```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - character string to be added to new structure  (input)
 - -      index - index for character string                    (input)
 - -      value - definition of surface                         (input)
 - -   mnemonic - surface definition mnemonic                   (input)
 - - equivalent_label
 - -            - label of
 - -          p - pointer to memory locations                    (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - - Structured Variable(s)
 - -  p_new - pointer to new structure
*/
  surface_usage_list *p_new;
/* - Check for Initial Member of String - - - - - - - - - - - - - - - - */
  if(p == NULL){
/* - Initial Member */
    p = memory_surface_usage_list(1,p);
    p->last = NULL;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
/* - Addition to Linked List */
  else{
    p_new = memory_surface_usage_list(1,p_new);
    p->next = p_new;
    p_new->last = p;
    p = p_new;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
}
```

Function memory_ascii_record

```
#include<stdio.h>

#include<malloc.h>
#include<stdlib.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
```

```
                                } a_record;

a_record *memory_ascii_record(int operation,int number, a_record *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_ascii_record - manages memory requests for character storage
 - -                       for variables of the ascii_record structure
 - -                       type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures for which to allocate      (input)
 - -             memory
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(a_record);
    p = (a_record *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += number*sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    size = number*sizeof(ascii_string);
    storage_c -= size;
```

```
    free((void *) p);
    p = NULL;
    return p;}
}
```

Function memory_fg_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct fuel_geometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

fg_list *memory_fg_list(int operation, int number, fg_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_fg_list - manages memory requests for storage for
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to allocate                (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -               requested
 - -  storage_rt - maximum bytes of single-precision real storage
```

```
 - -                requested
*/
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (fg_list *)calloc((size_t) number,(size_t) sizeof(fg_list));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += 12*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    storage_c += 12*sizeof(char)+sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_i -= 2*sizeof(int);
    storage_r -= 12*sizeof(float);
    storage_c -= 12*sizeof(char)+sizeof(ascii_string);
    return p;}
}
```

Function memory_float

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

float *memory_float(int operation, int number, float *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for single-precision real
 - -                  storage
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
```

```
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_r - current bytes of float storage requested
 - -  storage_rt - maximum bytes of float storage requested
*/
  extern int storage_r, storage_rt;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (float *)calloc((size_t)number,(size_t) sizeof(float));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_r += number;
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    storage_r -= number;
    free((void *) p);
    p = NULL;
    return p;};
}
```

Function memory_integer

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

int *memory_integer(int operation, int number, int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for integer storage
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (int *)calloc((size_t)number,(size_t) sizeof(int));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += number;
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    storage_i -= number;
    free((void *) p);
    p = NULL;
    return p;};
}
```

Function memory_lattice_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef struct all{
```

```
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
                    struct all *next;
                   } augmented_lattice_list;

augmented_lattice_list *memory_lattice_list(int operation
 ,augmented_lattice_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_lattice_list - manages memory requests for storage for vari-
 - -                       ables of the augmented_lattice_list structure
 - -                       type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(augmented_lattice_list);
    p = (augmented_lattice_list *)calloc((size_t) 1
     ,(size_t) sizeof(augmented_lattice_list));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
```

```
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_i -= 2*sizeof(int);
    return p;}
}
```

Function memory_s_material

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;


ll_material *memory_s_material(int operation,int number,ll_material *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_s_material - manages memory requests for storage for vari-
 - -                     ables of the s_material structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to process                 (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -              requested
 - -  storage_rt - maximum bytes of single-precision real storage
 - -              requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
```

```
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(ll_material);
    p = (ll_material *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += 3*number*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += 2*number*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += number*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= 3*number*sizeof(char);
    storage_i -= 2*number*sizeof(int);
    storage_r -= number*sizeof(float);
    return p;}
}
```

Function memory_surface_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct su_list{
                      struct su_list *last;
                      int index;
```

```
                    ascii_string label;
                    ascii_string value;
                    char mnemonic[4];
                    ascii_string equivalent_label;
                    struct su_list *next;
                  } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int operation
 , surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_surface_usage_list - manages memory requests for storage for
 - -                             variables of the surface_usage_list
 - -                             structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(surface_usage_list);
    p = (surface_usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n",errno);
      }
```

```
      lines(2);
      abort();
    }
    storage_c += 3*sizeof(ascii_string)+4*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= (3*sizeof(ascii_string)+4*sizeof(char));
    storage_i -= sizeof(int);
    return p;}
}
```

Function memory_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct u_list{
                       struct u_list *last;
                       int index;
                       ascii_string label;
                       struct u_list *next;
                      } usage_list;

usage_list *memory_usage_list(int operation, usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_usage_list - manages memory requests for storage for vari-
 - -                     ables of the usage_list structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
```

```
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(usage_list);
    p = (usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= sizeof(ascii_string);
    storage_i -= sizeof(int);
    return p;}
}
```

Function search_surface_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int operation
```

```
 , surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_surface_usage_list - manages memory requests for storage for
 - -                             variables of the surface_usage_list
 - -                             structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(surface_usage_list);
    p = (surface_usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n",errno);
      }
      lines(2);
      abort();
    }
    storage_c += 3*sizeof(ascii_string)+4*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
```

```
  else {
    free((void *) p);
    p = NULL;
    storage_c -= (3*sizeof(ascii_string)+4*sizeof(char));
    storage_i -= sizeof(int);
    return p;}
}
```

2.3. Input Routines

Subroutine ldlv

```
      SUBROUTINE LDLV(NBUNDLE,NAXP1,LU,LVECT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  L D L V *  Reads Vectors giving Lattice Assignments to Fuel
C - -                 Assemblies
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -      NBUNDLE - Number of Fuel Assemblies in Core          (input)
C - -        NAXP1 - Number of Axial Nodes in Core +1           (input)
C - -           LU - Logical Unit from which to Read Vectors    (input)
C - -        LVECT - Lattice Loading Vectors for each Fuel     (output)
C - -                Assembly
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - Character String for Messages
      CHARACTER*132 MESSAGE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
      DIMENSION LVECT(NAXP1,NBUNDLE)
C - - Skip Header Record - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(1X)')
      DO NB=1,NBUNDLE
        READ(LU,*,END=500) (LVECT(I,NB),I=1,NAXP1)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - -
  500 MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. LDLV --'
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Premature End-of-File Encountered on Logical Unit'')')
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' '',I2)') LU
      CALL lines(5)
      CALL fortran_message(MESSAGE)
      CALL abort()
      END
```

Subroutine lodct

```
      SUBROUTINE LODCT(NCOLP,NROWP,NAXP1,NLATTICG,NLATTICM,GMAP,MMAP
     2 ,LGVECT,LMVECT,CTABLE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  L O D C T *  Load Correspondence Table Matching Lattice
C - -                  Geometry Types with Lattice Material Types
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        NCOLP - Number of Columns in Core Fuel Assembly   (input)
C - -                Map
C - -        NROWP - Number of Rows in Core Fuel Assembly Map  (input)
C - -        NAXP1 - Number of Axial Nodes + 1                 (input)
C - -     NLATTICG - Number of Unique Lattice Geometry Types   (input)
C - -     NLATTICM - Number of Unique Lattice Material Types   (input)
C - -         GMAP - Fuel Assembly Geometry Index Map          (input)
```

```
C - -          MMAP - Fuel Assembly Material Index Map            (input)
C - -        LGVECT - Vectors Provides Lattice Geometry Assign-   (input)
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Geometry Index
C - -        LMVECT - Vectors Provides Lattice Material Assign-   (input)
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Material Index
C - -        CTABLE - Correspondence Table Providing Lattice     (output)
C - -                 Geometry Index for a Given Lattice Mater-
C - -                 ial Index
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MATCH - Flag to Indicate that a Lattice Material/Lattice
C - -           Geometry Pair has Already been Added to CTABLE
      LOGICAL MATCH
      INTEGER GMAP(NCOLP,NROWP),CTABLE(2,NLATTICM)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MMAP(NCOLP,NROWP),LGVECT(NAXP1,NLATTICG)
     2 ,LMVECT(NAXP1,NLATTICM)
C - - Sweep Over Entire Core - - - - - - - - - - - - - - - - - - - - -
      CALL INVALI((2*NLATTICM),CTABLE,0)
      IC = 0
      DO J = 1,NROWP
        DO I = 1,NCOLP
          IF(MMAP(I,J) .NE. 0) THEN
C - - Examine Each Node of Fuel Assembly Loading Vector
            DO K = 2,NAXP1
              LGV = LGVECT(K,GMAP(I,J))
              LMV = LMVECT(K,MMAP(I,J))
              IF(IC .EQ. 0) THEN
                IC = IC+1
                CTABLE(1,IC) = LMV
                CTABLE(2,IC) = LGV
              ELSE
                MATCH = .FALSE.
                DO N = 1,IC
                  IF((CTABLE(1,N) .EQ. LMV) .AND.
     2               (CTABLE(2,N) .EQ. LGV)) MATCH = .TRUE.
                ENDDO
                IF(.NOT. MATCH) THEN
                  IC = IC+1
                  CTABLE(1,IC) = LMV
                  CTABLE(2,IC) = LGV
                ENDIF
              ENDIF
            ENDDO
          ENDIF
        ENDDO
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine rblade

```
      SUBROUTINE RBLADE(LENFN,BLADE_DB,LUGEOM,NTUBE,CBSPAN,ATID,ATOD
     2 ,CBTHICK,TRSPAN,TRTHICK,WSTHICK,CBLENGTH,NCS,CSOFF,CSWIDTH
     3 ,CBPMAT,ATMAT,CBSMAT,CBTRMAT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R B L A D E  *  Reads Dataset Containing Control Blade
C - -                     Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                          (input)
C - -      BLADE_DB - Name of File Containing Appropriate Core-   (input)
C - -                 layout Database
C - -       LUGEOM - Logical Unit Number for Blade Geometry       (input)
C - -                 Dataset
C - -        NTUBE - Number of Absorber Tubes in All Four        (output)
C - -                 Wings of Blade
C - -       CBSPAN - Control Blade Span (Tie Rod Center to       (output)
C - -                 Tip)
C - -         ATID - Absorber Tube Inner Diameter (cm)           (output)
C - -         ATOD - Absorber Tube Outer Diameter (cm)           (output)
C - -      CBTHICK - Blade Wing Thickness (cm)                   (output)
C - -       TRSPAN - Tie Rod Span (cm)                           (output)
C - -      TRTHICK - Tie Rod Thickness (cm)                      (output)
C - -      WSTHICK - Wing Thickness (cm)                         (output)
C - -     CBLENGTH - Active Absrober Length (cm)                 (output)
C - -          NCS - Number of Central Stiffeners per Wing       (output)
C - -        CSOFF - Central Stiffener Offset                    (output)
C - -      CSWIDTH - Central Stiffener Width                     (output)
C - -       CBPMAT - Identifier for Blade Poison Material        (output)
C - -        ATMAT - Identifier for Blade Absorber Tube Mat-     (output)
C - -                 erial
C - -       CBSMAT - Identifier for Blade Sheath Material        (output)
C - -      CBTRMAT - Identifier for Blade Tie Rod Material       (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*6 CBPMAT,ATMAT,CBSMAT,CBTRMAT
      CHARACTER*(LENFN) BLADE_DB
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /BLADE/ NTUBE,CBSPAN,ATID,ATOD,CBTHICK,TRSPAN,TRTHICK
     2 ,CBLENGTH,WSTHICK,NCS,CSOFF,CSWIDTH,CBPMAT,ATMAT,CBSMAT,CBTRMAT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(LUGEOM,BLADE_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,BLADE)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine readin

```
      SUBROUTINE READIN(FIN,LENFN,CORE_DB,NIN,LUCGEOM,NAXIAL,AFL,NROW
     2 ,NCOL,NROWP,NCOLP,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR,TCGR,BLTPR
```

```
     3 ,BCPR,LPREFIX,FPREFIX,CORE_MTLS,MVESSEL,MSHROUD,MTG,MCP,MUTP
     4 ,MLTP,CORE_F,BLADE_DB,MIGT,DTOD,DTID,RHO,RHOBYP,TEMPK)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R E A D I N  *  Reads NAMELIST input to program -- including
C - -                     the core geometry file (defined as function
C - -                     for compatibility with C invocation)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          FIN - File Name for Input File                    (input)
C - -        LENFN - Length of File Name                         (input)
C - -      CORE_DB - Name of File Containing Appropriate Core-   (output)
C - -                layout Database
C - -          NIN - Logical Unit Number for Input File          (input)
C - -      LUCGEOM - Logical Unit Number for Core Geometry       (input)
C - -                Dataset File
C - -       NAXIAL - Number of Axial Nodes in Active Fuel        (output)
C - -          AFL - Active Fuel Length (cm)                     (output)
C - -         NROW - Number of Rows in Full Core                 (output)
C - -         NCOL - Number of Columns in Full Core              (output)
C - -        NROWP - Number of Rows in Core Represented in the  (output)
C - -                Problem
C - -        NCOLP - Number of Columnes in the Core Repre-      (output)
C - -                sented in the Problem
C - -       APITCH - Assembly Pitch (cm)                        (output)
C - -          SOD - Outer Radius of Core Shroud (cm)           (output)
C - -       STHICK - Core Shroud Thickness (cm)                 (output)
C - -          VOD - Pressure Vessel Outer Radius (cm)          (output)
C - -       VTHICK - Pressure Vessel Thickness (cm)             (output)
C - -        TUTPR - Top of the Upper Tie Plate Region (cm)     (output)
C - -         TCGR - Top of the Core Grid Region (cm)           (output)
C - -        BLTPR - Bottom of the Lower Tie Plate Region (cm)  (output)
C - -         BCPR - Bottom of the Core Plate Region (cm)       (output)
C - -      LPREFIX - File Description Prefix for Lattice Geo-   (output)
C - -                metry Files
C - -      FPREFIX - File Description Prefix for Fuel Materi    (output)
C - -                al Intermediate Datasets
C - -    CORE_MTLS - File Specification for Core Materials      (output)
C - -                Dataset
C - -      MVESSEL - Material Identifier for Vessel             (output)
C - -      MSHROUD - Material Identifier for Core Shroud        (output)
C - -          MTG - Material Identifier for Core Top Guide     (output)
C - -                Region
C - -          MCP - Material Identifier for Core Plate         (output)
C - -                Region
C - -         MUTP - Material Identifier for Upper Tie Plate    (output)
C - -                Region
C - -         MLTP - Material Identifier for Lower Tie Plate    (output)
C - -                Region
C - -       CORE_F - Fraction of Core in Problem                (output)
C - -                (1 - full, 2 - half, 4 - quarter)
C - -     BLADE_DB - Dataset for Control Blade used in the      (output)
C - -                Problem
C - -         MIGT - Material Identifier for Incore Guide Tube (output)
C - -         DTOD - Incore Guide Tube Outer Diameter (cm)     (output)
C - -         DTID - Incore Guide Tube Inner Diameter (cm)     (output)
```

```
C - -          RHO - In-channel Water Density (g/cc)            (output)
C - -       RHOBYP - Bypass Water Density (g/cc)                (output)
C - -        TEMPK - Problem Temperature (Kelvin)               (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - -     NIN - Logical Unit Number for Input File
C - - LUCGEOM - Logical Unit Number for Core-geometry Dataset File
      INTEGER NIN,CORE_F
      CHARACTER*6 MVESSEL,MSHROUD,MTG,MCP,MUTP,MLTP,MIGT
      CHARACTER*132 CORE_DB,LPREFIX,FPREFIX,CORE_MTLS,BLADE_DB
      CHARACTER*(LENFN) FIN
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /LINKIN/ CORE_DB,NAXIAL,AFL,NROWP,NCOLP,LPREFIX,FPREFIX
     2 ,CORE_MTLS,CORE_F,BLADE_DB,RHO,RHOBYP,TEMPK,MUTP,MLTP
      NAMELIST /CORE/ NCOL,NROW,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR
     2 ,TCGR,BLTPR,BCPR,MVESSEL,MSHROUD,DTOD,DTID,MTG,MCP,MIGT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(NIN,FIN)
C - - Skip Record Containing Case Title
      READ(NIN,'(1X)')
C - - Read NAMELIST Input Block
      READ(NIN,LINKIN)
C - - Process Core-geometry File Namelist Input Section - - - - - - - -
      CALL FTOPEN(LUCGEOM,CORE_DB)
C - - Skip Header Record
      READ(LUCGEOM,'(1X)')
C - - Read NAMELIST Input
      READ(LUCGEOM,CORE)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine rlattice

```
      SUBROUTINE RLATTICE(LENFN,FUEL_DB,LUGEOM,CTHICK,ASIN,WGAP,NGAP
     2 ,CRADIUS,FSRD,CFSRD,RPITCH,COD,CLD,POD,FRCMAT,FCMAT,LATDIM,NWR
     3 ,WROD,WRTH)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R L A T T I C E *  Reads Dataset Containing Lattice
C - -                        Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                          (input)
C - -      FUEL_DB - Name of File Containing Appropriate          (input)
C - -                Lattice-layout Database
C - -       CTHICK - Channel Thickness (cm)                      (output)
C - -         ASIN - Inner Span of Channel (cm)                  (output)
C - -         WGAP - Wide Gap Thickness (cm)                     (output)
C - -         NGAP - Narrow Gap Thickness (cm)                   (output)
C - -      CRADIUS - Inner Radius of Channel Corner (cm)         (output)
C - -         FSRD - Clad Surface to Clad Surface Separation     (output)
C - -                (cm)
C - -        CFSRD - Clad Surface to Inner Channel Surface       (output)
C - -                Separation (cm)
C - -       RPITCH - Fuel Rod Pitch (cm)                         (output)
```

```
C - -           COD - Cladding Outer Diameter (cm)                (output)
C - -           CLD - Cladding Thickness (cm)                     (output)
C - -           POD - Fuel Pellet Outer Diameter (cm)             (output)
C - -        FRCMAT - Material Identifier for Fuel Rod Cladding   (output)
C - -         FCMAT - Material Identifier for Channel             (output)
C - -        LATDIM - Lattice Dimensionality                      (output)
C - -           NWR - Number of Water Rods                        (output)
C - -          WROD - Water Rod Outer Diameter (cm)               (output)
C - -          WRTH - Water Rod Thickness (cm)                    (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      REAL NGAP
      CHARACTER*6 FRCMAT,FCMAT
      CHARACTER*(LENFN) FUEL_DB
C - - Initialization - - - - - - - - - - - - - - - - - - - - - - - - - -
      WROD = 0.0
      WRTH = 0.0
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /FUEL/ CTHICK,ASIN,WGAP,NGAP,CRADIUS,FSRD,CFSRD,RPITCH
     2 ,COD,CLD,POD,LATDIM,NWR,FRCMAT,FCMAT,WROD,WRTH
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(LUGEOM,FUEL_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,FUEL)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

2.4. Input Editing Routines

Function edit_ct

```
#include <stdio.h>
#include <string.h>

void edit_ct(int nlatticm,int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  e d i t _ c t  *  Edits Contents of Correspondence Table, which
 - -                      Relates Lattice Geometry Indices to Lattice
 - -                      Material Indices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - Number of Unique Lattice Material Indices      (input)
 - -          p - Pointer to Correspondence Table               (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  n - index for loop
 - - p2 - pointer to second member of correspondence table
*/
  int n, *p2;
/* - File Variable(s)
 - -  nout - output stream
*/
  extern FILE *nout;
/* - Edit Correspondence Table - - - - - - - - - - - - - - - - - - - - */
  lines(7);
  fprintf(nout,"0Correspondence Table for Lattice Material and Geometry");
  fprintf(nout," Indices\n");
  fprintf(nout,"0Index Material Geometry\n\n");
  p2 = p+1;
  for( n = 1; n <= nlatticm; n++){
    if(n > 7) lines(1);
    fprintf(nout," %3i   %6i   %6i\n",n,*p,*p2);
    p++; p++;
    p2 = p+1;}
}
```

Function edit_spacer

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void edit_spacer(int nbundlg,int *ptr_n_spacer
 ,float *ptr_spacer_location,ascii_string *ptr_spacer_material){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  e d i t _ s p a c e r  *  Edits Input Variables Defining
 - -                               Fuel Assembly Spacers
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -    nbundlg - number of fuel assembly geometrical types    (input)
```

```
 - - ptr_n_spacer
 - -             - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -             - pointer to array containing location of spa- (input)
 - -               cers for each fuel assembly type
 - - ptr_spacer_material
 - -             - pointer to array containing spacer material  (input)
 - -               mnemonic for each fue assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -       i - utility index variable
 - -       j - utility index variable
 - -  ptr_n - pointer to integer vector
*/
  short int i,j;
  int *ptr_n = ptr_n_spacer;
/* - Float Variable(s)
 - - ptr_f - pointer to floating point array
*/
  float *ptr_f = ptr_spacer_location;
/* - Character Variable(s)
 - -  ptr_c - pointer to ascii_string vector
 - - buffer - character variable for processing output
*/
  ascii_string buffer, *ptr_c = ptr_spacer_material;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - */
  header();
  lines(5+nbundlg);
  fprintf(nout
   ,"0Fuel Assembly Spacer Information\n");
/* - Number of Spacers for Each Unique Fuel Assembly Geometry */
  fprintf(nout,"0Index  Number of Spacers  Material Mnemonic\n\n");
  for(i = 1;i <= nbundlg;i++){
    strcpy(buffer,"");
    strncpy(buffer,*ptr_c,7);
    fprintf(nout," %3i          %2i                %s\n"
     ,i,*ptr_n,buffer);
    ptr_n++;
    ptr_c++;}
/* - Spacer Coordinates for Each Type */
  lines(3+nbundlg);
  fprintf(nout,"0Index Axial Distance from BAF (cm)\n\n");
  ptr_n = ptr_n_spacer;
  for(i = 1;i <= nbundlg;i++){
    fprintf(nout," %3i",i);
    for(j = 1;j <= *ptr_n;j++){
      fprintf(nout," %7.3f",*ptr_f);
      *ptr_f++;
    }
```

```
    fprintf(nout,"\n");
    ptr_n++;
  }
}
```

Function fgds_edt

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct fuel_gometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

  void fgds_edt(int nlatticg,char lprefix[],ascii_string *lgdsnam
   ,fg_list *ptr_lg_ds){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - fgds_edt - edits contents of fuel geometry datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -   lprefix - prefix for lattice geometry datasets            (input)
 - -   lgdsnam - file descriptions for lattice geometry datasets (input)
 - - ptr_lg_ds - pointer to first element in list of fuel geo-   (input)
 - -             datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   i - loop control variable
*/
  short int i;
/* - Character Variable(s)
 - -     fn - filename for lattice geometry dataset
 - - buffer - string used to manage output
*/
  ascii_string fn, buffer;
/* - FILE Variable(s)
```

```
 - -  nout - stream for output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  p - pointer to current element in list of fuel geometry datasets
 - -  l - pointer to current element in list of lattice gometry dataset
 - -       names
*/
  fg_list *p = ptr_lg_ds;
  ascii_string *l = lgdsnam;
/* - Loop Over all the Lattice Geometry Types - - - - - - - - - - - - - - */
  for( i = 1; i <= nlatticg; i++,p++,l++){
    if(p->wrod != 0)
      lines(20);
    else
      lines(22);
    strcpy(fn,lprefix);
    strcat(fn,*l);
    fprintf(nout,"0Values from Lattice Geometry Dataset:\n  %s\n",fn);
    fprintf(nout,"0 Variable      Value    Definition\n\n");
    sprintf(buffer,"    LATDIM    %6i",p->latdim);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Lattice Dimensionality\n");
    fprintf(nout,buffer);
    sprintf(buffer,"       NWR    %6i",p->nwr);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Number of Water Rods\n");
    fprintf(nout,buffer);
    sprintf(buffer,"    CTHICK     %5.2f",p->cthick);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      ASIN     %5.2f",p->asin);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Interior Span (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      WGAP     %5.2f",p->wgap);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Wide Gap Half-thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      NGAP     %5.2f",p->ngap);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Narrow Gap Half-thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"   CRADIUS     %5.2f",p->cradius);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Corner Inner Radius (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      FSRD     %5.2f",p->fsrd);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Clad Surface to Clad Surface Separation (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"     CFSRD     %5.2f",p->cfsrd);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer
```

```
     ,"Clad Surface to Inner Channel Surface Separation (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"    RPITCH     %5.2f",p->rpitch);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Rod Pitch (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"       COD     %5.2f",p->cod);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Rod Cladding Outer Diameter (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"       CLD     %5.2f",p->cld);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Rod Cladding Thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"       POD     %5.2f",p->pod);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Pellet Outer Diameter (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"    FRCMAT     %s",p->frcmat);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Rod Cladding Material Identifier\n");
    fprintf(nout,buffer);
    sprintf(buffer,"     FCMAT     %s",p->fcmat);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Channel Material Identifier\n");
    fprintf(nout,buffer);
    if(p->wrod != 0.0){
      sprintf(buffer,"      WROD     %5.2f",p->wrod);
      bufferpad(buffer,strlen(buffer),23);
      strcat(buffer,"Water Rod Outer Diameter (cm)\n");
      fprintf(nout,buffer);
      sprintf(buffer,"      WRTH     %5.2f",p->wrth);
      bufferpad(buffer,strlen(buffer),23);
      strcat(buffer,"Water Rod Thickness (cm)\n");
      fprintf(nout,buffer);}
  }
}
```

2.5. Deck Generation Routines

Function add_surface

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                       struct su_list *last;
                       int index;
                       ascii_string label;
                       ascii_string value;
                       char mnemonic[4];
                       ascii_string equivalent_label;
                       struct su_list *next;
                      } surface_usage_list;

surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);

void add_surface(int operation,ascii_string label,char mnemonic[4]
 ,FILE *lu8,FILE *lu9,int *index,ascii_string value
 ,surface_usage_list *ptr_surface_usage,int *nsurface){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ s u r f a c e  *  adds surface to MCNP model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  operation - operation to perform                          (input)
 - -              (0 - add new surface card,
 - -               1 - do not add new surface card)
 - -      label - Label Surface                                 (input)
 - -   mnemonic - MCNP mnemonic for surface (first edit only)   (input)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-  (input)
 - -              initions
 - -        lu9 - Pointer to Scratch File Stream for Surface    (input)
 - -              Definitions
 - -      index - Index for Surface                               (i|o)
 - -      value - String Containing Values for Defining Surface (input)
 - - ptr_surface_usage
 - -            - Points to Surface Usage Linked List          (input)
 - -   nsurface - total number of surfaces                        (i|o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        multiplier - multiplier on index for cell card
 - -  equivalent_index - index for equivalent surface
*/
  int multiplier = 1.0, equivalent_index;
/* - Character Variable(s)
 - -          buffer - buffer for processing lines of output
```

```
 - -          ptr_buf - pointer to buffer
 - - equivalent_label - label for surface with identical definition
*/
  ascii_string buffer, equivalent_label = "";
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Structured Variable(s) - - - - - - - - - - - - - - - - - - - - - -
 - -  ptr_sl - pointer to surface usage list
*/
  surface_usage_list *ptr_sl;
/* - For New Surfaces,  Write to Surface Accumulation Scratch File */
  if(*index < 0) {
    multiplier = -1;
    *index = -(*index);}
  if(operation == 0){
/* - Search for Equivalent Surface that has already been Defined */
    search_surface_usage_list(2,"",&equivalent_index,value,mnemonic
     ,equivalent_label,ptr_surface_usage);
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    if(equivalent_index == 0){
      (*nsurface)++;
      *index = *nsurface;}
    else
      *index = equivalent_index;
    ptr_sl = load_surface_usage_list(label,*index,value,mnemonic
     ,equivalent_label,ptr_sl);
    if(equivalent_index == 0){
      fprintf(lu9,"c     %s\n",label);
      fprintf(lu9,"%5i %s %s\n",*index,mnemonic,value);}
  }
/* - Write Surface Definition to Cell Card in MCNP Input Deck - - - -  */
  fprintf(lu8," %i",(multiplier*(*index)));
/* - Write to Edit */
  sprintf(buffer,"       %6i %s",(multiplier*(*index)),mnemonic);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
}
```

Function augment_lattice_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct all{
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
```

```
                    struct all *next;
                   } augmented_lattice_list;


augmented_lattice_list *memory_lattice_list(int
 ,augmented_lattice_list *);

void augment_lattice_list(int *nlatticm,int *lmvect,int *lgvect
 ,int nlatticm_ref,int nlatticg,int nbundlm,int nbundlg,int naxial
 ,int *ptr_n_spacer,int *ptr_spacer_node
 ,augmented_lattice_list *additional_lattices
 ,int *ptr_correspondence_table){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * a u g m e n t _ l a t t i c e _ l i s t  *  adds lattices incor-
 - -                                                porating spacer grids
 - -                                                into material lattice
 - -                                                loading vectors
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - total number of unique material lattice indices  (i&o)
 - -     lmvect - material lattice loading vectors                 (i&o)
 - -     lgvect - geometrical lattice loading vectors             (input)
 - -  nlatticm_ref
 - -              - total number of unique material lattice indi-  (input)
 - -                ces without augmentation for lattices incor-
 - -                porating spacers
 - -   nlatticg - total number of unique geometrical lattice in- (input)
 - -              dices
 - -    nbundlm - number of unique fuel assembly materials        (input)
 - -    nbundlg - number of unique fuel assembly geometries       (input)
 - -     naxial - number of axial nodes in problem                (input)
 - - ptr_n_spacer
 - -              - number of spacer grids associated with each    (input)
 - -                geometrically unique fuel assembly
 - - ptr_spacer_node
 - -              - nodal location of spacer grids for each geo-   (input)
 - -                metric fuel assembly type
 - - additional_lattices
 - -              - linked list of additional lattices generated  (output)
 - -                to accomodate spacer grids
 - - ptr_correspondence_table
 - -              - table providing relationship between lattice   (input)
 - -                material indices and lattice geometrical in-
 - -                dices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -               k - axial index for looping
 - -               n - uility loop variable
 - -           nglat - lattice geometry index
 - -           ngasm - fuel assembly geometry index
 - -             lmv - pointer to lattice material index list
 - -             lgv - pointer to lattice geometry index list
 - - spacer_present - flag to indicate whether a spacer grid is present
```

```
 - -                  at this location
*/
  short int k,n;
  int ngasm, spacer_present, nglat;
  int *lmv = lmvect, *lgv;
/* - Structured Variable(s)
 - -  ptr_al - pointer to additional lattice linked list
*/
  augmented_lattice_list *ptr_al = additional_lattices;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Sweep through Material Assembly Types - - - - - - - - - - - - - - - */
  for(n = 1;n <= nbundlm;n++){
/* - Skip Fuel Assembly Index */
    lmv++;
/* - Sweep from Bottom of Assmbly to Top */
    for(k = 1;k <= naxial;k++,lmv++){
/* - Determine Corresponding Lattice Geometrical Index */
      { short int i;
        int *p = ptr_correspondence_table;
        while(*p != *lmv) {p++;p++;}
        nglat = *(p+1);
      }
/* - Find Fuel Assembly Geometrical Type of which the Lattice Geometry
 - - is a Member */
      lgv = lgvect;
      { short int i,bk,ns;
        int *ptr = ptr_n_spacer;
        int *ptr_n = ptr_spacer_node;
        ngasm = 0;
        for(i = 1;i <= nbundlg;i++)
          if(ngasm == 0){
/* - Skip Fuel Assembly Index */
            lgv++;
            for(bk = 1;bk <= naxial;bk++,lgv++)
              if(*lgv == nglat){
                ngasm = i;
                break;}
          }
          else break;
/* - Compare Nodes at which Spacer Grids are Located with Current Axial
 - - Node Index */
        spacer_present = 0;
        for(i = 1;i < ngasm;i++,ptr++)
          for(ns = 1;ns <= *ptr;ns++,ptr_n++);
        for(bk = 1;bk <= *ptr;bk++,ptr_n++)
          if(k == *ptr_n){
            spacer_present = 1;
            break;}
      }
/* - Determine whether this Lattice Incorporating a Spacer Grid has
 - - already been defined */
      if(spacer_present == 1){
```

```
        { short int found = 0, first = 0;
          augmented_lattice_list *ptr_next;
          ptr_al = additional_lattices;
          do{
              if(first != 0) ptr_al = ptr_al->next;
              first = 1;
              if(ptr_al->basis_lattice_material_index == *lmv){
                found = 1;
                *lmv = ptr_al->lattice_material_index;
                break;}
            }
          while(ptr_al->next != NULL);
/* - Spacer Grid not already Defined, Augment Lattice List */
          if(found == 0){
            ptr_al = additional_lattices;
/* - Check for First Entry in Linked List */
            if(ptr_al->lattice_material_index != NULL){
              while(ptr_al->next != NULL) ptr_al = ptr_al->next;
              ptr_next = memory_lattice_list(1,ptr_next);
              ptr_next->last = ptr_al;
              ptr_al->next = ptr_next;
              ptr_al = ptr_next;}
            (*nlatticm)++;
            ptr_al->basis_lattice_material_index = *lmv;
            ptr_al->lattice_material_index = *nlatticm;
            ptr_al->next = NULL;
            *lmv = *nlatticm;
          }
        }
      }
    }
  }
}
```

Function build_assemblies

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void lines(int);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
```

```
void build_assemblies(int *ncell,int *nuniverse,FILE *lu8,FILE *lu9
 ,surface_usage_list *ptr_surface_usage,int **ptr_ufa
 ,int nbundlm,int *lmvect,int naxial,int *ptr_ufl,int *nsurface
 ,int *ptr_n_node,float *ptr_node_boundaries,int *lgvect
 ,int *ptr_correspondence_table,int nlatticm,int nbundlg){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  b u i l d _ a s s e m b l i e s  *  builds fuel assemblies from
 - -                                        constituent lattices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  Argument(s):
 - -       ncell - number of cells in MCNP model                  (input)
 - -    nuniverse - number of "universes" in MCNP model           (input)
 - -          lu8 - stream pointer for scratch file for cell      (input)
 - -                definition accumulation
 - -          lu9 - stream pointer for scratch file for surface   (input)
 - -                definition accumulation
 - - ptr_surface_usage
 - -              - linked list for surfaces already defined in   (input)
 - -                the problem
 - -      ptr_ufa - vector with universe indices for unique fuel (output)
 - -                assembly types
 - -      nbundlm - number of unique fuel assembly types          (input)
 - -       lmvect - vector containing assignment of lattice mat-  (input)
 - -                erial indices to fuel assembly material indi-
 - -                ces
 - -       naxial - number of axial nodes                         (input)
 - -      ptr_ufl - vector with universe indices for unique lat-  (input)
 - -                tice types
 - -     nsurface - total number of surfaces                     (input)
 - -   ptr_n_node - number of axial nodes for each geometrical   (input)
 - -                fuel assembly type
 - - ptr_node_boundaries
 - -              - locations of tops of axial nodes for each    (input)
 - -                geometrical fuel assembly type
 - -       lgvect - vector containing assignment of lattice geo- (input)
 - -                metrical indices to fuel assembly material
 - -                indices
 - - ptr_correspondence_table
 - -              - tables relating lattice material indices to  (input)
 - -                lattice geometrical indices
 - -     nlatticm - number of unique material lattice types      (input)
 - -      nbundlg - number of unique geometrical fuel assembly   (input)
 - -                types
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -           i - utility loop variable
 - -           n - loop variable
 - -           k - loop control for axial location
 - -       index - utility index variable
 - -       ngasm - index for geometical assembly type
 - -         lmv - pointer to lmvect vector
 - -     ptr_lmv - second pointer to lmvect vector
```

```
 - -  ptr_table - pointer to correspondence table
 - -      ptr_n - pointer to vector containing number of axial nodes
 - -              for each geometrically unique fuel assembly
 - -          p - pointer to **ptr_ufa vector
*/
  short int i,n,k;
  int index, ngasm, *lmv = lmvect, *ptr_lmv = lmvect;
  int *ptr_lgv = lgvect;
  int *ptr_n = ptr_n_node;
  int *ptr_table = ptr_correspondence_table;
  int *p = *ptr_ufa;
/* - Float Variable(s)
 - -     ptr_nb - pointer to array containing the axial locations for
 - -              tops of axial nodes for each geometrically unique
 - -              fuel assembly
*/
  float *ptr_nb = ptr_node_boundaries;
/* - Character Variable(s)
 - -    label - label for cell, surface or material
 - - mnemonic - mnemonic for surfaces
 - - material - name of material
 - -    value - string for defining surface definition
*/
  char mnemonic[4];
  ascii_string label, material, value;
/* - FILE Variable(s)
 - -  nout - pointer to code output file
*/
  extern FILE *nout;
/* - Loop Over Unqiue Fuel Assemblies in the Core - - - - - - - - - -  */
  for(n = 1;n <= nbundlm;n++){
    (*nuniverse)++;
/* - Determine the Fuel Assembly Geometrical Index Corresonding to this
 - - Fuel Assembly Material Index; Start by Locating the Bottom Node
 Material Lattice Index in Correspondence Table */
    for(i = 1;i < n;i++)
      for(k = 1;k <= naxial;k++) ptr_lmv++;
    for(i = 1;i <= nlatticm;i++)
      if(*ptr_table == *ptr_lmv) break;
      else {ptr_table++; ptr_table++;}
/* - Determine the Fuel Assembly Geometrical Type of which the Lattice
 - - Geometrical Type is a Member */
    ngasm = 0;
    ptr_lgv = lgvect;
    for(i = 1;i <= nbundlg;i++){
      for(k = 1;k <= naxial;k++)
        if(*ptr_lgv == *ptr_table){
          ngasm = i;
          break;}
        else
          ptr_lgv++;
      if(ngasm != 0) break;
    }
    ptr_n = ptr_n_node;
    ptr_nb = ptr_node_boundaries;
```

```
    for(i = 1;i < ngasm;i++){
      for(k = 1;k <= *ptr_n;k++) ptr_nb++;
      ptr_n++;}
/* - Loop Over all the Nodes in an Assembly */
    for(k = 1;k <= naxial;k++){
/* - Cells for Axial Nodes */
      sprintf(label,"Axial Node %i for Fuel Assembly %i",k,n);
      strcpy(material,"void");
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
/* - Surfaces for Node Boundaries */
      if(k != 1) {
        sprintf(label,"Top of Node %i",(k-1));
        search_surface_usage_list(1,label,&index,"","",""
         ,ptr_surface_usage);
        if(index == 0){
          lines(3);
          fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
          fprintf(nout," populate_control_cells --\n");
          fprintf(nout
           ," Surface not Found in Linked List, label = %s\n"
           ,label);}
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
      if(k != naxial){
        sprintf(label,"Top of Node %i",k);
        search_surface_usage_list(1,label,&index,"","",""
         ,ptr_surface_usage);
        if(index == 0){
          strcpy(mnemonic,"pz");
          sprintf(value,"%10.4E",*ptr_nb);
          index = -1;
          add_surface(0,label,mnemonic,lu8,lu9,&index,value
           ,ptr_surface_usage,nsurface);}
        else{
          index = -index;
          add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
           ,nsurface);}
        ptr_nb++;
      }
      fprintf(lu8,"\n");
      { short int nn;
        int *l = lmv;
        int *p = ptr_ufl;
        l++;
        for(nn = 1;nn < k;nn++,l++) continue;
        for(nn = 1;nn < *l;nn++,p++) continue;
        fprintf(lu8,"      fill= %i\n",*p);
        fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
      }
    }
/*    **ptr_ufa = *nuniverse; */
    *p = *nuniverse;
    p++;
    for(k = 1;k <= (naxial+1);k++) lmv++;
  }
```

```
}
```

Function ge7x7_lattice

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge7x7_lattice(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
```

```
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 7 x 7 _ l a t t i c e  *  Create Lattice Model for GE 7x7
 - -                                  Lattice Model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      cthick - Channel Thickness (cm)                          (input)
 - -        asin - Inner Span of Channel (cm)                      (input)
 - -        wgap - Wide Gap Thickness (cm)                         (input)
 - -        ngap - Narrow Gap Thickness (cm)                       (input)
 - -     cradius - Inner Radius of Channel Corner (cm)             (input)
 - -        fsrd - Clad Surface to Clad Surface Separation         (input)
 - -               (cm)
 - -       cfsrd - Clad Surface to Inner Channel Surface          (input)
 - -               Separation (cm)
 - -      rpitch - Fuel Rod Pitch (cm)                            (input)
 - -         cod - Cladding Outer Diameter (cm)                   (input)
 - -         cld - Cladding Thickness (cm)                        (input)
 - -         pod - Fuel Pellet Outer Diameter (cm)                (input)
 - -      frcmat - Material Identifier for Fuel Rod Cladding      (input)
 - -       fcmat - Material Identifier for Channel                (input)
 - -      latdim - Lattice Dimensionality                         (input)
 - -         nft - Number of Unique Fuel Rod Types                (input)
 - -     lattice - Fuel Rod Type Map                              (input)
 - -       ncell - number of MCNP cells                             (i&o)
 - -  nmaterial - number of MCNP materials                         (i&o)
 - -         lu8 - stream pointer to scratch file for cell defin- (input)
 - -               itions
 - -         lu9 - stream pointer to scratch file for surface     (input)
 - -               definitions
 - -        lu10 - stream pointer to scratch file for material    (input)
 - -               definitions
 - - ptr_surface_usage
 - -             - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -             - list of defined materials                      (input)
 - - inchannel_density
 - -             - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -             - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                         (i&o)
 - - fp_density - density for fuel pellets                        (input)
 - -   matdsnam - name for fuel intermediate material dataset     (input)
 - -       fmids - linked list for fuel material composition      (input)
 - - ptr_core_mtls
 - -             - pointer to linked list containing core mat-    (input)
 - -               erial definitions
 - -        nlat - index for unique lattice for which to create   (input)
 - -               MCNP lattice model
 - -         ufl - index for universe that contains the entire    (input)
```

```
 - -              lattice representation
 - -   nsurface - total number of surfaces                          (input)
 - - inchannel_material
 - -             - mnemonic for inchannel moderator material       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
```

```
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -            dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
```

```
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 0;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    if(index == 0){
      { usage_list *ptr_ml = ptr_material_usage;
        while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
        index = (ptr_ml->index)+1;
        ptr_ml = load_usage_list(material,index,ptr_ml);
      }
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
```

```
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
```

```
      n_entries = 0;
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
```

```
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n",label);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        fprintf(lu8,"%3i ",(*plattice+ufr-1));
        fprintf(nout,"%3i ",(*plattice+ufr-1));}
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
```

```
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",yambig2);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"):(");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",wgcowy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",wgciwy);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
```

```
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
```

```
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
```

```
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," ):\n      (");
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
```

```
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (NE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (SE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"     Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function ge8x8_lattice_lcwr

```
#include <stdio.h>
#include <string.h>
```

```
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry{
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                             } fg_list;


void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
```

```
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge8x8_lattice_lcwr(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []
 ,float wrod,float wrth,int *ncell_tr){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 8 x 8 _ l a t t i c e _ l c w r  *  Create Lattice Model for
 - -                                            GE 8x8 Lattice with a
 - -                                            Large Central Water
 - -                                            Rod
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     cthick - Channel Thickness (cm)                          (input)
 - -       asin - Inner Span of Channel (cm)                      (input)
 - -       wgap - Wide Gap Thickness (cm)                         (input)
 - -       ngap - Narrow Gap Thickness (cm)                       (input)
 - -    cradius - Inner Radius of Channel Corner (cm)             (input)
 - -       fsrd - Clad Surface to Clad Surface Separation         (input)
 - -              (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface           (input)
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)                            (input)
 - -        cod - Cladding Outer Diameter (cm)                   (input)
 - -        cld - Cladding Thickness (cm)                        (input)
 - -        pod - Fuel Pellet Outer Diameter (cm)                (input)
 - -     frcmat - Material Identifier for Fuel Rod Cladding      (input)
 - -      fcmat - Material Identifier for Channel                (input)
 - -     latdim - Lattice Dimensionality                         (input)
 - -        nft - Number of Unique Fuel Rod Types                (input)
 - -    lattice - Fuel Rod Type Map                              (input)
 - -      ncell - number of MCNP cells                             (i&o)
 - -  nmaterial - number of MCNP materials                        (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin- (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface     (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material    (input)
```

```
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                          (input)
 - - ptr_material_usage
 - -            - list of defined materials                         (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)           (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)               (input)
 - -  nuniverse - number of MCNP "universes"                          (i&o)
 - - fp_density - density for fuel pellets                          (input)
 - -   matdsnam - name for fuel intermediate material dataset       (input)
 - -      fmids - linked list for fuel material composition         (input)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-       (input)
 - -              erial definitions
 - -       nlat - index for unique lattice for which to create      (input)
 - -              MCNP lattice model
 - -        ufl - index for universe that contains the entire       (input)
 - -              lattice representation
 - -   nsurface - total number of surfaces                          (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material         (input)
 - -       wrod - outer diameter for water rod (cm)                 (input)
 - -       wrth - water rod thickness (cm)                          (input)
 - -   ncell_tr - number of MCNP cells for translated cells          (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
 - -     uwrnw - universe index for northwest quadrant of water rod
 - -     uwrne - universe index for northeast quadrant of water rod
 - -     uwrse - universe index for southeast quadrant of water rod
 - -     uwrsw - universe index for southwest quadrant of water rod
 - -    ncwrnw - cell index for water inside water rod for northwest
 - -             quadrant of water rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan, uwrnw, uwrne, uwrse, uwrsw;
  int cchan, cwic;
```

```
  int ncpellet, ncgap, ncclad, ncwater, ncwrnw;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - -    wror - radius for outer surface of water rod
 - -    wrir - radius for inner surface of water rod
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset,wrir,wror;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
```

```
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  wror = wrod/2;
  wrir = wror-wrth;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
```

```
          ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 0;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    if(index == 0){
      { usage_list *ptr_ml = ptr_material_usage;
        while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
        index = (ptr_ml->index)+1;
        ptr_ml = load_usage_list(material,index,ptr_ml);
      }
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
```

```
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
      n_entries = 0;
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
```

```
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - -  */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
```

```
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Water Rod for Lattice - - - - - - - - - - - - - - - - - - */
/* - Water Rod in Northwest Quadrant */
/* - Water */
  (*nuniverse)++;
  uwrnw = *nuniverse;
  sprintf(label,"Water Rod Center (NW Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
  ncwrnw = *ncell;
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset-ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Cladding */
  sprintf(label,"Water Rod (NW Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
```

```
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset-ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (NW Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Water Rod Section in Northeast Quadrant */
/* - Inner Water */
  (*nuniverse)++;
  uwrne = *nuniverse;
  sprintf(label,"Water Rod Center (NE Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset-ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Cladding */
  sprintf(label,"Water Rod (NE Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
```

```
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset-ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (NE Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
```

```
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Water Rod Section in Southeast Quadrant */
/* - Inner Water */
  (*nuniverse)++;
  uwrse = *nuniverse;
  sprintf(label,"Water Rod Center (SE Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset+ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Cladding */
  sprintf(label,"Water Rod (SE Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
```

```
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset+ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (SE Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Water Rod Section in Southwest Quadrant */
```

```
/* - Inner Water */
  (*nuniverse)++;
  uwrsw = *nuniverse;
  sprintf(label,"Water Rod Center (SW Quadrant), L%i",nlat);
    strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset+ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrsw);
/* - Cladding */
  sprintf(label,"Water Rod (SW Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset+ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrsw);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (SW Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrsw);
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
```

```
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int nwr = 0, uwr;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        if(*plattice > 0){
          fprintf(lu8,"%3i ",(*plattice+ufr-1));
          fprintf(nout,"%3i ",(*plattice+ufr-1));}
```

```
        else{
          switch (nwr) {
            case 0:
              uwr = uwrnw;
              break;
            case 1:
              uwr = uwrne;
              break;
            case 2:
              uwr = uwrsw;
              break;
            case 3:
              uwr = uwrse;
          }
          nwr++;
          fprintf(lu8,"%3i ",uwr);
          fprintf(nout,"%3i ",uwr);}
      }
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
```

```
sprintf(value,"%10.4E",wgcowx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",wgciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",yambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",yambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):(");
sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",wgcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",wgciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
```

```
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
```

```
      ," Surface Not Found in Linked List, label = %s\n",label);
     abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
```

```
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8," ):\n      (");
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
```

```
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (NW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
```

```
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (NE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
```

```
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (SE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
```

```
   abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (SW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,")\n");
fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
lines(1);
fprintf(nout,"     Filled with Universe %i\n",ufrl);
```

```
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
```

```
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
```

```
    ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
    ,nsurface);
  sprintf(label
    ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
    ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function ge8x8_lattice_swr

```
#include <stdio.h>

#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
```

```
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry{
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              char frcmat[6];
                              char fcmat[6];
                             } fg_list;


void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge8x8_lattice_swr(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
```

```
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 8 x 8 _ l a t t i c e _ s w r  *  Create Lattice Model for
 - -                                          GE 8x8 Lattice with Water
 - -                                          Rods the same Size as the
 - -                                          Fuel Rods
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      cthick - Channel Thickness (cm)                          (input)
 - -        asin - Inner Span of Channel (cm)                      (input)
 - -        wgap - Wide Gap Thickness (cm)                         (input)
 - -        ngap - Narrow Gap Thickness (cm)                       (input)
 - -     cradius - Inner Radius of Channel Corner (cm)             (input)
 - -        fsrd - Clad Surface to Clad Surface Separation         (input)
 - -               (cm)
 - -       cfsrd - Clad Surface to Inner Channel Surface          (input)
 - -               Separation (cm)
 - -      rpitch - Fuel Rod Pitch (cm)                            (input)
 - -         cod - Cladding Outer Diameter (cm)                   (input)
 - -         cld - Cladding Thickness (cm)                        (input)
 - -         pod - Fuel Pellet Outer Diameter (cm)                (input)
 - -      frcmat - Material Identifier for Fuel Rod Cladding      (input)
 - -       fcmat - Material Identifier for Channel                (input)
 - -      latdim - Lattice Dimensionality                         (input)
 - -         nft - Number of Unique Fuel Rod Types                (input)
 - -     lattice - Fuel Rod Type Map                              (input)
 - -       ncell - number of MCNP cells                             (i&o)
 - -   nmaterial - number of MCNP materials                         (i&o)
 - -         lu8 - stream pointer to scratch file for cell defin- (input)
 - -               itions
 - -         lu9 - stream pointer to scratch file for surface     (input)
 - -               definitions
 - -        lu10 - stream pointer to scratch file for material    (input)
 - -               definitions
 - - ptr_surface_usage
 - -             - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -             - list of defined materials                      (input)
 - - inchannel_density
 - -             - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -             - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                         (i&o)
 - - fp_density - density for fuel pellets                        (input)
 - -   matdsnam - name for fuel intermediate material dataset     (input)
 - -       fmids - linked list for fuel material composition      (input)
 - - ptr_core_mtls
 - -             - pointer to linked list containing core mat-    (input)
 - -               erial definitions
 - -        nlat - index for unique lattice for which to create   (input)
 - -               MCNP lattice model
 - -         ufl - index for universe that contains the entire    (input)
```

```
 - -              lattice representation
 - -   nsurface - total number of surfaces                          (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
 - -       uwr - universe index for water rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan, uwr;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
```

```
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -            dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
```

```
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 1;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    { usage_list *ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(material,index,ptr_ml);
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
```

```
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
      n_entries = 0;
```

```
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - -  */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
```

```
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Water Rod for Lattice - - - - - - - - - - - - - - - - - */
/* - Water */
  (*nuniverse)++;
  uwr = *nuniverse;
  sprintf(label,"Water Rod Center, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Cladding */
  sprintf(label,"Water Rod Cladding, L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
```

```
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
```

```
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        if(*plattice > 0){
          fprintf(lu8,"%3i ",(*plattice+ufr-1));
          fprintf(nout,"%3i ",(*plattice+ufr-1));}
        else{
          fprintf(lu8,"%3i ",uwr);
          fprintf(nout,"%3i ",uwr);}
      }
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - - */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
```

```
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",yambig2);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"):(");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",wgcowy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",wgciwy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
```

```
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
```

```
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
```

```
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8," ):\n      (");
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (NW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Channel Corner Inner Radius (SE), L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (SW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"     Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
```

```
      ," Surface Not Found in Linked List, label = %s\n",label);
     abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function generate_lattice_model

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
```

```
                              } a_record;
typedef struct s_material{
                             struct s_material *last;
                             int atomic_number;
                             int mass_number;
                             float weight_percentage;
                             char library_suffix[5];
                             struct s_material *next;
                            } ll_material;
typedef struct u_list {
                           struct u_list *last;
                           int index;
                           ascii_string label;
                           struct u_list *next;
                          } usage_list;

typedef struct su_list {
                            struct su_list *last;
                            int index;
                            ascii_string label;
                            ascii_string value;
                            char mnemonic[4];
                            ascii_string equivalent_label;
                            struct su_list *next;
                           } surface_usage_list;

typedef struct fuel_geometry {
                                 ascii_string gds_name;
                                 int latdim;
                                 int nwr;
                                 float cthick;
                                 float asin;
                                 float wgap;
                                 float ngap;
                                 float cradius;
                                 float fsrd;
                                 float cfsrd;
                                 float rpitch;
                                 float cod;
                                 float cld;
                                 float pod;
                                 float wrod;
                                 float wrth;
                                 char frcmat[6];
                                 char fcmat[6];
                                } fg_list;


void ge7x7_lattice(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void ge8x8_lattice_swr(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
```

```
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void ge8x8_lattice_lcwr(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[],float,float,int *);
void rollup_llm(ll_material *);
int *memory_integer(int,int,int *);
ll_material *load_fuel_material(int *,char[],char[],int,int **
 ,float **);
int mchar(int *,char[]);

void generate_lattice_model(int *ncell,int *nmaterial,FILE *lu8
 ,FILE *lu9,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,int *ptr_correspondence_table
 ,fg_list *ptr_lg_ds,ascii_string *lmdsnam,int nlat,char fprefix[]
 ,float inchannel_density,float bypass_density,int *nuniverse
 ,a_record *ptr_core_mtls,int *ufl,int *nsurface
 ,char inchannel_material[],int *ncell_tr){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e n e r a t e _ l a t t i c e _ m o d e l  *  Create Lattice
 - -                                                    Model for a
 - -                                                    Unique Node
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ncell - number of MCNP cells                              (i&o)
 - -   nmaterial - number of MCNP materials                         (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin-  (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -            - list of defined materials                      (input)
 - - ptr_correspondence_table
 - -            - table relating lattices with unique material   (input)
 - -              definitions with geometry definitions
 - -  ptr_lg_ds - list of lattice geometry datasets              (input)
 - -    lmdsnam - list of material intermediate dataset names    (input)
 - -       nlat - index for unique lattice for which to create   (input)
 - -              MCNP lattice model
 - -    fprefix - prefix for fuel material intermediate datasets (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-    (input)
 - -              erial definitions
```

```
 - -        ufl - universe definition for unique lattices       (output)
 - -   nsurface - total number of surfaces                        (i&o)
 - - inchannel_material
 - -            - menmonic for inchannel moderator material     (input)
 - -   ncell_tr - number of MCNP cells for translated cells       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  latdim - lattice dimensionality
 - -     nwr - number of water rods
 - -   nglat - index for lattice geometry dataset corresponding to
 - -            lattice material dataset
 - - lattice - fuel rod type map for lattice
 - -     nft - number of fuel rod types
*/
  int latdim, nwr;
  int nglat, *lattice, nft;
/* - Float Variable(s)
 - -     cthick - Channel Thickness (cm)
 - -       asin - Inner Span of Channel (cm)
 - -       wgap - Wide Gap Thickness (cm)
 - -       ngap - Narrow Gap Thickness (cm)
 - -    cradius - Inner Radius of Channel Corner (cm)
 - -       fsrd - Clad Surface to Clad Surface Separation (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)
 - -        cod - Cladding Outer Diameter (cm)
 - -        cld - Cladding Thickness (cm)
 - -        pod - Fuel Pellet Outer Diameter (cm)
 - -       wrod - Water Rod Outer Diameter (cm)
 - -       wrth - Water Rod Thickness (cm)
 - - fp_density - Density Vector for Lattice
*/
  float cthick, asin, wgap, ngap, cradius, fsrd, cfsrd, rpitch
   ,cod, cld, pod, wrod, wrth;
  float *fp_density;
/* - Character Datasets
 - -  matdsnam - name for material dataset
 - -    frcmat - Material Identifier for Fuel Rod Cladding
 - -     fcmat - Material Identifier for Channel
 - -       gds - name for lattice geometry dataset
*/
  char frcmat[6], fcmat[6];
  ascii_string matdsnam, gds;
/* - Structured Variable(s)
 - -  ptr_fg - pointer to proper fuel material dataset
 - -   fmids - pointer to fuel material dataset
*/
  fg_list *ptr_fg = ptr_lg_ds;
  ll_material *fmids;
/* - Adjust Character String Read with FORTRAN for Processing in C -  */
  { short int n;
    int len = 132, length;
```

```
    ascii_string *p = lmdsnam;
    for( n = 1; n < nlat; n++,p++) continue;
    strncpy(matdsnam,*p,132);
    length = mchar(&len,matdsnam);
    matdsnam[length] = '\0';
  }
/* - Find Corresponding Lattice Geometry Dataset */
  { short int n;
    int *p = ptr_correspondence_table;
/*    do{
      p++;p++;}
    while(*p != nlat);
*/
    while(*p != nlat) {p++;p++;}
    nglat = *(p+1);
    for( n = 0; n < (nglat-1); n++) ptr_fg++;
    latdim = ptr_fg->latdim;
    strcpy(gds,ptr_fg->gds_name);
  }
/* - Allocate Memory for Lattice Map */
  lattice = memory_integer(1,(latdim*latdim),lattice);
/* - Load Material Dataset for Lattice - - - - - - - - - - - - - - - - */
  fmids = load_fuel_material(&nft,fprefix,matdsnam,latdim,&lattice
   ,&fp_density);
/* - Create Lattice Model - - - - - - - - - - - - - - - - - - - - - - */
   cthick = ptr_fg->cthick;    asin = ptr_fg->asin;
     wgap = ptr_fg->wgap;      ngap = ptr_fg->ngap;
  cradius = ptr_fg->cradius;   fsrd = ptr_fg->fsrd;
    cfsrd = ptr_fg->cfsrd;   rpitch = ptr_fg->rpitch;
      cod = ptr_fg->cod;        cld = ptr_fg->cld;
      pod = ptr_fg->pod;
     wrod = ptr_fg->wrod;      wrth = ptr_fg->wrth;
  strcpy(frcmat,ptr_fg->frcmat);
  strcpy(fcmat,ptr_fg->fcmat);
  if(strstr(gds,"ge7x7") != NULL)
    ge7x7_lattice(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
     ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
     ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
     ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
     ,nlat,ufl,nsurface,inchannel_material);
   else
    if(wrod == 0.0)
      ge8x8_lattice_swr(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
       ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
       ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
       ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
       ,nlat,ufl,nsurface,inchannel_material);
     else
      ge8x8_lattice_lcwr(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
       ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
       ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
       ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
       ,nlat,ufl,nsurface,inchannel_material,wrod,wrth,ncell_tr);
/* - Return Memory */
  { short int n;
```

```
    ll_material *p = fmids, *p_last;
    for( n = 0; n < (nft-1); n++,p++) continue;
    p_last = p;
    for( n = nft; n > 0; n--){
      p = p_last;
      p_last--;
      rollup_llm(p);}
  }
  lattice = memory_integer(-1,(latdim*latdim),lattice);
}
```

Function material_match

```
#include<stdio.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

ll_material *memory_s_material(int,int,ll_material *);


ll_material *material_match(a_record *p,char name[],float *density
 ,int *n_entries){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * m a t e r i a l _ m a t c h *  Matches Material Identifiers
 - -                                   with Materials on Linked List
 - -                                   for Core Materials
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -          p - pointer to first element in linked list       (input)
 - -       name - string containing name of material sought     (input)
 - -    density - density of material                          (output)
 - -  n_entries - number of elements/isotopes of which the     (output)
 - -              material is composed
 - -    m_first - pointer to first element in material defin-  (output)
 - -              ition link list
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -           n - counter for processing elements within material
 - -               definition
 - -  n_elements - number of elements of which a material is composed
 - -  n_isotopes - number of isotopes for a particular element in a
```

```
 - -                material
 - -    a_number - atomic number from material in core materials
 - -                linked list
 - -    m_number - mass number of isotope
*/
  short int n;
  int n_elements, n_isotopes, a_number, m_number;
/* - Float Variable(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - -   ewp - elemental weight-percentage from core materials linked
 - -         list
 - -   iwp - isotopic weight-percentage from core materials linked
 - -         list
*/
  float ewp, iwp;
/* - Character Variable(s)
 - -  db_name - string to hold name of material from linked list
 - -     line - buffer to hold "line" from linked list
 - - l_suffix - library suffix from element or isotope on core mater-
 - -            ial linked list
 - -  element - elemental name from element on core material linked
 - -            list
*/
  char db_name[6], l_suffix[5], element[15];
  ascii_string line;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structure Variable(s) - - - - - - - - - - - - - - - - - - - - - -
 - -  pm_current - pointer to current position in linked list for
 - -                material definition
 - -     pm_next - pointer to next position in linked list for material
 - -                definition
 - -     m_first - first element in linked list to material definition
*/
  ll_material *pm_current, *pm_next, *m_first = NULL;
/* - Sweep through the linked list seeking material identifier match */
  while(p->next != 0){
    strcpy(line,p->line);
    if(strstr(line,name) != NULL){          /* - match on identifier */
      *n_entries = 0;
      sscanf(line,"%s %f %i",db_name,density,&n_elements);
      for(n = 1; n <= n_elements; n++){     /* - sweep through ele - */
        p = p->next;                         /* - ment definitions -  */
        sscanf(p->line,"%s %f %i %s %i"
         ,element,&ewp,&a_number,l_suffix,&n_isotopes);
        pm_next = memory_s_material(1,1,pm_next);
        if(m_first == NULL) {
          m_first = pm_next;
          pm_current = pm_next;
          pm_current->last = NULL;}
        else{
          pm_current->next = pm_next;
          pm_next->last = pm_current;
          pm_current = pm_next;
```

```
        }
        pm_current->next = NULL;
        if(strlen(l_suffix) == 1){
          { short int i;
            *n_entries += n_isotopes;
            for(i = 1;i <= n_isotopes;i++){
              p = p->next;
              sscanf(p->line,"%i %f %s",&m_number,&iwp,l_suffix);
              iwp = ewp*iwp/100.0;
              pm_current->atomic_number = a_number;
              pm_current->mass_number = m_number;
              pm_current->weight_percentage = iwp;
              strcpy(pm_current->library_suffix,l_suffix);
              if(i < n_isotopes){
                pm_next = memory_s_material(1,1,pm_next);
                pm_current->next = pm_next;
                pm_next->last = pm_current;
                pm_current = pm_next;}
            }
          }}
        else{
          (*n_entries)++;
          pm_current->atomic_number = a_number;
          pm_current->mass_number = 0;
          pm_current->weight_percentage = ewp;
          strcpy(pm_current->library_suffix,l_suffix);
          { short int i;
            for(i = 1;i <= n_isotopes; i++) p=p->next;}
        }
      }
      return m_first;
    }
    p = p->next;
  }
/* - No match found -- fatal error - - - - - - - - - - - - - - - - - - */
  lines(4);
  fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
  fprintf(nout," material match:\n");
  fprintf(nout,"0No Match Found for Input Material %s\n",name);
  abort();
}
```

Function source_specification

```
#include <stdio.h>

void source_specification(int nsrck,float rkk,int ikz,int kct,FILE *lu10
 ,int ncolcc,int nrowcc,int *ccmap,int *ccmapw,float apitch,float afl
 ,int core_f,int ndm,int mct,int ndmp,int ncell,float tempk){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  s o u r c e _ s p e c i f i c a t i o n  *  Adds Source Specif-
 - -                                                 ication to Input
 - -                                                 Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      nsrck - nominal source size per cycle in MCNP          (input)
```

```
 - -        rkk - initial guess for eigenvalue in MCNP            (input)
 - -        ikz - number of cycles to be skipped before begin-    (input)
 - -              ning tally accumulation in MCNP
 - -        kct - number of cycles to be performed in MCNP        (input)
 - -       lu10 - stream pointer for material scratch file        (input)
 - -     ncolcc - Number of Columns in Control Cell Map           (input)
 - -     nrowcc - Number of Rows in Control Cell Map              (input)
 - -      ccmap - Map of Control Cell Universe Number             (input)
 - -     ccmapw - Map Containing Contents of Each Control Cell    (input)
 - -     apitch - fuel assembly pitch (cm)                        (input)
 - -        afl - active fuel length (cm)                         (input)
 - -     core_f - fraction of core modeled in problem             (input)
 - -              (0 - full,
 - -               1 - half,
 - -               4 - quarter)
 - -        ndm - periodicity of dumps to TPE file                (input)
 - -        mct - flag to creation of MCTAL file                  (input)
 - -       ndmp - maximum number of dumps on file                 (input)
 - -      ncell - total number of cells in the problem            (input)
 - -      tempk - problem temperature (Kelvin)                    (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - loop variable for columns in map
 - -        j - loop variable for rows in map
 - -        n - utility loop variable
 - -     nsrc - number of source fields
 - -  ptr_map - pointer to ccmap array
 - - ptr_mapw - pointer to ccmapw array
 - -   i_zero - column for origin of problem
 - -   j_zero - row of origin of problem
*/
  short int i,j,n,i_zero,j_zero;
  int nsrc = 0;
  int *ptr_map = ccmap, *ptr_mapw = ccmapw;
/* - Float Variable(s)
 - -   x - abscissa for center of source body
 - -   y - ordinate for center of source body
*/
  float x,y;
/* - Print Program Control and Source Cards - - - - - - - - - - - - - - */
/* - The thermal scattering kernal temperature in MeV is computed from*/
/* - the formula given on page 3-108 of the MCNP-4B User's Manual - - */
  fprintf(lu10,"c     Control Cards\n");
  fprintf(lu10,"c\n");
  fprintf(lu10," tmp1 %10.4E %ir\n",(8.617E-11*tempk),(ncell-1));
  fprintf(lu10,"kcode  %5i %5.3f %3i %5i\n",nsrck,rkk,ikz,kct);
  fprintf(lu10,"print 40\n");
  fprintf(lu10," lost 20 20 \n");
  fprintf(lu10,"prdmp %i %i %i %i\n",kct,ndm,mct,ndmp);
/* - Source Fields */
  fprintf(lu10," sdef  erg=d1 pos=d2 axs= 0 0 1 rad=d3 ext=d4\n");
  fprintf(lu10,"  sp1  -3\n");
  fprintf(lu10,"  si2  L");
```

```
  for( j = 1; j <= nrowcc; j++){
    for( i = 1; i <= ncolcc; i++,ptr_map++){
/* - Check for Null Control Cell */
      if(*ptr_map == 0){
        for(n = 1;n <= 6; n++) ptr_mapw++;
        continue;}
/* - Compute Coordinates of Source Body Center */
      switch (core_f){
      case 4:
        i_zero = ncolcc;
        j_zero = nrowcc;
        break;
      case 1:
        i_zero = ncolcc/2;
        j_zero = nrowcc/2;
        break;
      case 2:
        i_zero = ncolcc;
        j_zero = nrowcc/2;
        break;
      }
      if((i == i_zero) || (j == j_zero)) continue;
      nsrc++;
      x = (i-i_zero)*2*apitch;
      y = (j_zero-j)*2*apitch;
      if(nsrc == 1)
        fprintf(lu10," %10.4E %10.4E %10.4E\n",x,y,(afl/2));
      else
        fprintf(lu10,"         %10.4E %10.4E %10.4E\n",x,y
         ,(afl/2));
    }
  }
  fprintf(lu10,"  sp2");
  { int nmax=10,nmin=1;
    do{
      for(n = nmin;n <= nmax;n++)
        fprintf(lu10," 1");
      fprintf(lu10,"\n");
      if(n <= nsrc) fprintf(lu10,"     ");
      nmin += 10;
      nmax += 10;
      if(nmax > nsrc) nmax = nsrc;
    }while(n <= nsrc);
  }
  fprintf(lu10,"  si3 %10.4E\n",((2*apitch)-0.01));
  fprintf(lu10,"  si4 %10.4E\n",(afl/2));
}
```

Function spacer_location

```
void spacer_location(int nbundlg,int *ptr_n_spacer
 ,float *ptr_spacer_location,int *ptr_spacer_node,int *ptr_n_node
 ,float *ptr_node_boundaries){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  s p a c e r _ l o c a t i o n  *  Determines Nodal Locations
 - -                                       of each Spacer
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -    nbundlg - number of unique fuel assembly geometries     (input)
 - - ptr_n_spacer
 - -            - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -            - pointer to array containing location of spa- (input)
 - -              cers for each fuel assembly type (cm)
 - - ptr_spacer_node
 - -            - pointer to array containing space node       (output)
 - -              locations for each fuel assembly type
 - - ptr_n_node - number of node boundaries for each unique     (input)
 - -              fuel assembly type
 - - ptr_node_boundaries
 - -            - axial locations of node boundaries for each  (input)
 - -              unique fuel assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - utility index variable
 - -        j - utility index variable
 - -        k - utility index variable
 - -        n - utility index variable
 - -    ptr_n - pointer to integer vector
 - -   ptr_sn - pointer to integer vector containing spacer nodal
 - -            locations
 - - ptr_node - pointer to integer vector containing the number of
 - -            nodes for each unique fuel assembly type
 - -    found - flag to indicate that a node has been found in which
 - -            a spacer is resident
*/
  short int i,j,k,n,found;
  int *ptr_n = ptr_n_spacer, *ptr_sn = ptr_spacer_node;
  int *ptr_node = ptr_n_node;
/* - Float Variable(s)
 - -       ptr_f - pointer to floating point array
 - -      ptr_nb - pointer to floating point vector containing
 - -               the node boundaries
*/
  float *ptr_f = ptr_spacer_location;
  float *ptr_nb = ptr_node_boundaries;
/* - Sweep over the Unique Fuel Assembly Geometry Types - - - - - -  */
  for(i = 1;i <= nbundlg;i++,ptr_n++,ptr_node++){
/* - Loop over the Spacers in this Geometry Type */
    for(j = 1;j <= *ptr_n;j++,ptr_f++){
/* - Loop over the Axial Nodes in this Fuel Assembly Type */
      found = 0;
      ptr_nb = ptr_node_boundaries;
      for(n = 1;n < i;n++)
        for(k = 1;k <= *ptr_node;k++) *ptr_nb++;
      for(k = 1;k <= *ptr_node;k++,ptr_nb++){
        if((*ptr_f <= *ptr_nb) && (found != 1)){
          *ptr_sn = k;
          ptr_sn++;
```

```
            found = 1;}
        }
        if(found != 1){
          *ptr_sn = (*ptr_node)+1;
          ptr_sn++;}
      }
    }
}
```

Function vessel_generation

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                       } usage_list;

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;


ll_material *material_match(a_record *,char[],float * ,int *);
ll_material *memory_s_material(int, ll_material *);
void rollup_llm(ll_material *);
usage_list *memory_usage_list(int,usage_list *);
surface_usage_list *memory_surface_usage_list(int
 ,surface_usage_list *);
usage_list *load_usage_list(char label[],int index,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
```

```
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
int mchar(int *,char[]);
void header();
void lines(int);
void bufferpad(char[],int,int);
void abort();
void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_symmetry_surfaces(int,FILE *,FILE *,surface_usage_list *
 ,int);
void add_material(FILE *,int *,char[],int,ll_material *);

void vessel_generation(float apitch,float vod,float vthick,float sod
 ,float sthick,float tutpr,float tcgr,float bltpr,float bcpr
 ,FILE *lu8, FILE *lu9, FILE *lu10,int *ncell,int *nsurface
 ,int *nmaterial,a_record *ptr_core_mtls,char mvessel[],char mshroud[]
 ,char mtg[],char mcp[],char mutp[],char mltp[]
 ,int core_f,surface_usage_list *ptr_surface_usage
 ,usage_list **ptr_material_usage,float bypass_density,float afl){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * v e s s e l _ g e n e r a t i o n  *  Generates Vessel Compo-
 - -                                          nents in MCNP Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     apitch - fuel assembly pitch                              (input)
 - -        sod - shroud outer radius                              (input)
 - -     sthick - shroud thickness                                 (input)
 - -        vod - pressure vessel outer radius                     (input)
 - -     vthick - pressure vessel thickness                        (input)
 - -      tutpr - Top of Upper Tie Plate Region                    (input)
 - -       tcgr - Top of Core Grid Region                          (input)
 - -      bltpr - Bottom of Lower Tie Plate Region                 (input)
 - -       bcpr - Bottom of Core Plate Region                      (input)
 - - incore_loc - positions for incore instrumentation             (input)
 - -        lu8 - pointer to scratch file for cell representa-     (input)
 - -              tations
 - -        lu9 - pointer to scratch file for surface repre-       (input)
 - -              tations
 - -       lu10 - pointer to scratch file for material repre-      (input)
 - -              sentations
 - -      ncell - number of MCNP gometrical cells in the in-       (output)
 - -              put card images thus far
 - -   nsurface - number of MCNP gometrical surfaces in the       (output)
 - -              input card images thus far
 - -  nmaterial - number of MNCP materials in the problem         (output)
 - -              thus far
 - -  ptr_core_mtls
 - -            - pointer to linked list of core materials       (input)
 - -    mvessel - material identifier for vessel                  (input)
 - -    mshroud - material identifier for core shroud             (input)
 - -        mtg - material identifier for top grid region         (input)
 - -        mcp - material identifier for core plate region       (input)
```

```
 - -       mutp - material identifier for upper tie plate       (input)
 - -              region
 - -       mltp - material identifier for lower tie plate       (input)
 - -              region
 - -     core_f - fraction of core in problem                   (input)
 - -              (1 - full, 2 - half, 4 - quarter)
 - -  ptr_surface_usage
 - -            - pointer to linked list for surface labels    (output)
 - -              and indices
 - -  ptr_material_usage
 - -            - pointer to linked list for material labels   (output)
 - -              and indices
 - -  bypass_density
 - -            - density for bypass water in problem (g/cc)    (input)
 - -        afl - active fuel length (cm)                       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          ns - counter for number of "virtual" surfaces accmulated
 - -   n_entries - number of elements and isotopes in a particular
 - -               material
 - -       index - index used for searching usage lists
 - -        zero - zero value;
 - -         pid - process identifier of job creating input deck
 - -     version - version number for linkage code
*/
  short int ns;
  int n_entries,index,zero = 0;
  extern short version;
  extern long pid;
/* - Float Variable(s)
 - -  density - material density
*/
  float density;
/* - Character Variable(s)
 - -     buffer - string variable used to load entire lines for print
 - -      label - label for editing
 - -    ptr_buf - pointer to buffer;
 - -       zaid - MCNP element/isotope identifier
 - -   mnemonic - surface type mnemonic
 - - case_title - title for job
 - -      value - definition of surface
 - -     codemn - name of linkage code
 - -      cdate - date of linkage code execution
 - -     crtime - time of linkage code execution
 - - modification_level
 - -            - modification level for linkage code
*/
  char zaid[11], mnemonic[4];
  extern char case_title[];
  ascii_string buffer, label, value;
  char *ptr_buf;
  extern char codenm[5];
  extern char cdate[9];
```

```
  extern char crtime[9];
  extern char modification_level;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition link list
 - -       pm - working pointer to material definition link list
 - -   ptr_sl - pointer to current record in surface usage link list
 - -   ptr_ml - pointer to current record in material usage link list
*/
  ll_material *ptr_mtl, *pm;
  usage_list *ptr_ml = NULL;
  surface_usage_list *ptr_sl = NULL;
/* - Write Title for Case and Header Records for Cell Representations */
  fprintf(lu8,case_title);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Input Deck Generated by %s, Version %2i:%c\n"
   ,codenm,version,modification_level);
  fprintf(lu8,"c     Generated on %s at %s by Process %i\n"
   ,cdate,crtime,pid);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cell Cards\n");
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cells Defining Problem Domain\n");
  fprintf(lu8,"c\n");
/* - Write Header Records for Surface Representations - - - - - - - - */
  fprintf(lu9,"c     Surface Cards\n");
  fprintf(lu9,"c\n");
  fprintf(lu9,"c     Surfaces for Problem Domain\n");
/* - Write Header Records for Material Representations - - - - - - - - */
  fprintf(lu10,"c     Material Cards\n");
/* - Write Header for Deck Generation Edit Table */
  header();
  lines(5);
  fprintf(nout
   ,"0MCNP Input Card Representation Generation Summary Table\n");
  fprintf(nout
   ,"0 Cell Surface(s)     Material/Index/Density    Universe");
  fprintf(nout," Definition\n");
  fprintf(nout,"\n");
/* - Vessel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell Cards */
  (*nmaterial)++;
  { int len = 6, length;
    length = mchar(&len,mvessel);
    mvessel[length] = '\0';
  }
  ptr_mtl = material_match(ptr_core_mtls,mvessel,&density
   ,&n_entries);
  *ptr_material_usage = load_usage_list(mvessel,1,*ptr_material_usage);
  ptr_ml = *ptr_material_usage;
  strcpy(label,"Pressure Vessel");
  add_cell(label,mvessel,lu8,ncell,*nmaterial,-density,&zero);
```

```
/* - Surface Cards */
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",(tcgr+30.0));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",(bcpr-30.0));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Maximum Radial Extent of Problem */
  strcpy(label,"Maximum Radial Extent of Problem");
  strcpy(mnemonic,"cz");
  index = -1;
  sprintf(value,"%10.4E",(vod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Boundaries, if Present */
  if((core_f == 2) || (core_f == 4)){
    (*nsurface)++;
    strcpy(label,"X-Z Plane");
    strcpy(mnemonic,"py");
    strcpy(value,"0.0");
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    ptr_sl = load_surface_usage_list(label,*nsurface,value
     ,mnemonic,"",ptr_sl);
    fprintf(lu8," %i",*nsurface);
    fprintf(lu9,"c     %s\n",ptr_sl->label);
    fprintf(lu9,"   *%1i %s 0.0\n",*nsurface,mnemonic);
    sprintf(buffer,"       %6i",*nsurface);
    bufferpad(buffer,strlen(buffer),59);
    ptr_buf = buffer+59*sizeof(char);
    sprintf(ptr_buf,"%s\n",label);
    lines(1);
    fprintf(nout,buffer);
    if(core_f == 4){
      (*nsurface)++;
      strcpy(label,"Y-Z Plane");
      strcpy(mnemonic,"px");
      strcpy(value,"0.0");
      ptr_sl = load_surface_usage_list(label,*nsurface,value
       ,mnemonic,"",ptr_sl);
      fprintf(lu8," %i",-(*nsurface));
      fprintf(lu9,"c     %s\n",ptr_sl->label);
      fprintf(lu9,"   *%1i %s 0.0\n",*nsurface,mnemonic);}
      sprintf(buffer,"       %6i %s",-(*nsurface),mnemonic);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s\n",label);
```

```
      lines(1);
      fprintf(nout,buffer);
  }
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",((vod/2)-vthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n= 1.0\n");
/* - Material Cards */
  add_material(lu10,nmaterial,mvessel,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - "Outside" World - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"""Outside"" World");
  add_cell(label,"void",lu8,ncell,0,0.0,&zero);
/* - Edit List of Surfaces */
  fprintf(lu8," (");
/* - Maximum Radial Extent of Problem */
  index = 3;
  search_surface_usage_list(0,label,&index,"","","",""
   ,ptr_surface_usage);
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  index = 1;
  search_surface_usage_list(0,label,&index,"","","",""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  index = 2;
  search_surface_usage_list(0,label,&index,"","","",""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4)){
    strcpy(label,"X-Z Plane");
    search_surface_usage_list(1,label,&index,"","","",""
     ,ptr_surface_usage);
    fprintf(lu8,":");
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);
    if(core_f == 4){
      strcpy(label,"Y-Z Plane");
      search_surface_usage_list(1,label,&index,"","","",""
       ,ptr_surface_usage);
      fprintf(lu8,":");
```

```
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
  }
  fprintf(lu8,") ");
  fprintf(lu8,"\n      imp:n= 0.0\n");
/* - Jet Pump Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  (*nmaterial)++;
  strcpy(label,"Jet Pump Region");
  add_cell(label,"Bypass Water",lu8,ncell,*nmaterial,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",(sod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
```

```
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/*  - Material(s) */
  { short int i, nloc;
    char *cp;
    strcpy(label,"Bypass Water");
    fprintf(lu10,"c     %s\n",label);
    ptr_ml = load_usage_list(label,*nmaterial,ptr_ml);
    strcpy(zaid,"001001.50c");
    nloc = 0;
    if(*nmaterial < 1000) nloc++;
    if(*nmaterial < 100) nloc++;
    if(*nmaterial < 10) nloc++;
    strcpy(label,"2.0");
    { short int ip;
      for(ip = 1;ip <= nloc;ip++)
        fprintf(lu10," ");
        fprintf(lu10,"m%i %s %s\n"
         ,*nmaterial,zaid,label);
    }
    strcpy(zaid,"008016.50c");
    strcpy(label,"1.0");
    fprintf(lu10,"      %s %s\n"
      ,zaid,label);
    { short int ip;
      for(ip = 1;ip <= (nloc-1);ip++) fprintf(lu10," ");
      fprintf(lu10,"mt%i lwtr.01\n",*nmaterial);
    }
  }
/* - Core Shroud - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"Core Shroud");
/* - Load Core Materials Database */
  { int len = 6, length;
    length = mchar(&len,mshroud);
    mshroud[length] = '\0';
  }
  search_usage_list(1,mshroud,&index,*ptr_material_usage);
  if(index == 0){
    (*nmaterial)++;
    index = *nmaterial;
    ptr_mtl = material_match(ptr_core_mtls,mshroud,&density
     ,&n_entries);
    ptr_ml = *ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(mshroud,index,ptr_ml);}
```

```
    else
      n_entries = 0;
  add_cell(label,mshroud,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  sprintf(value,"%10.4E",((sod/2)-sthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
```

```
/*  - Material(s) */
  if(n_entries != 0){
    add_material(lu10,nmaterial,mshroud,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
  }
/* - Upper Tie Plate Region - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mutp);
    mutp[length] = '\0';
  }
  strcpy(label,mutp);
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mutp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mutp,index,ptr_ml);
  strcpy(label,"Upper Tie Plate Region");
  add_cell(label,mutp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Active Fuel */
  strcpy(label,"Top of Active Fuel");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",afl);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",tutpr);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
```

```
  add_material(lu10,nmaterial,mutp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Core Grid Region - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mtg);
    mtg[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mtg,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mtg,index,ptr_ml);
  strcpy(label,"Core Grid Region");
  add_cell(label,mtg,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",tcgr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mtg,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Upper Plenum Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Upper Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
/* - Lower Tie Plate Region - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mltp);
    mltp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mltp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mltp,index,ptr_ml);
  strcpy(label,"Lower Tie Plate Region");
  add_cell(label,mltp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Active Fuel */
  strcpy(label,"Bottom of Active Fuel");
  strcpy(mnemonic,"pz");
  sprintf(value,"0.0");
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bltpr);
```

```
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mltp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Fuel Support/Core Plate Region - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mcp);
    mcp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mcp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mcp,index,ptr_ml);
  strcpy(label,"Fuel Support/Core Plate");
  add_cell(label,mcp,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Fuel Support/Core Grid Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
```

```
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bcpr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mcp,n_entries,ptr_mtl);
/* - Lower Plenum Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  sprintf(label,"Lower Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Fuel Support/Core Plate Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
```

```
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
}
```