dcomment

A General Purpose File Preprocessor For use with the LCS series of codes Version 1.10

Jeffrey V. Siebers
Loma Linda University Medical Center
Loma Linda, California 92354
siebers@llumc.edu

April 18, 1997

1 Introduction

A general purpose file preprocessor has been developed to create input files for the LCS series of codes. The preprocessor was written to allow the programs that require simple input files (LAHET, MCNP, HTAPE, SABRINA, ...) to have alternative input files that have extended comments, defined values, logical processing of the input, long file names, and to allow the input to be split among multiple input files. Use of the preprocessor allows several input files to share a common sub-set of data (*ie* LAHET, MCNP, and SABRINA can all use the exact same files for the geometry deck).

The preprocessor was originally written for use with LAHET Code System input files, however, it can be used a a general preprocessor. The preprocessor is similar to a c/C++ compiler preprocessor. Many of the major functions of the preprocessor c/C++ are included, but minor functions (in the view of the author) have not been coded.

2 General Usage

The syntax for using the preprocessor is:
dcomment *(input_name) (output_name) (-d name) (-v name value) (-u) (-l).*

input_name This is the name of the input file to be processed. This argument is optional. If the user fails to include the *input_name* on the command line, the user is prompted for the *input_name* by the program.

output_name This is the name of the output file to be produced. This argument is also optional. If the user fails to include the *output_name* on the command line, the user id prompted for the *output_name* by the program.

-d name This flag allows the user to define *name* from the command line. This flag is for defines without an associated with a value. See Section 5.2.

-v name value This flag allows the user to define *name* to be equal to *value* from the command line. See Section 5.2.

-u This flag will result in all output to be written in upper-case.

-l This flag will result in all output to be written in lower-case.

3 Comments

Many computer codes (particularly Monte Carlo transport codes such as LAHET, HTAPE, MCNP) require a input files in a pre-determined format, with limited comments. This preprocessor allows verbose comments, which are stripped from the input file while making the output file (the output file from the preprocessor is the input file for the other computer code). The format for the comments are the same as for a standard C++ preprocessor. Comments start with /* and all text is skipped until a */ is encountered. Additionally, // can be used to delimit input that is to be skipped until the end of the line. The code

```
/* The dcomment preprocessor allows verbose comments
   the comments can span over several lines */
this line is not a comment
/* or they can be limited to single lines */
this line begins as text // and ends as a comment
this line ends with a multi-line comment /* that can be used to describe your
                                            input file */
the end
```

expands out to

```
this line is not a comment
this line begins as text
this line ends with a multi-line comment
the end
```

4 Mathematical Operations Allowed in Input File

Mathematical operations of addition(+), subtraction(-), and multiplication(*) are allowed in the input file line. Division(/) is not allowed since it can not be differentiated from the LCS end of line delimiter (/).

Parenthesis are NOT allowed. Operations are evaluated from left to right, with multiplication being performed, then addition/subtraction.

```
/* math_test.inp */
1+2+3+4 = 10
1-2+3-4 = -2
2*3-4*6 = -18
6*5.01 = 30.06
4*6+5.01 = 29.01    /* (4*6) + 5.01 = 29.01*/
3-4*6+5.01 = -15.99 /* 3 - (4*6) + 5.01 = -15.99 */
2*3-4*6+5.01 = -12.99  /* (2*3) - (4*6) +5.01 = -12.99 */
/* You can use defines (see section below) in math operations */
#define PI 3.14159265
#define RADIUS 2.0
#define AREA PI*RADIUS*RADIUS
```

```
A Circle of radius RADIUS has an area of AREA
```

evaluates to

```
10.000000 = 10
-2.000000 = -2
-18.000000 = -18
30.060000 = 30.06
29.010000 = 29.01
-15.990000 = -15.99
-12.990000 = -12.99
A Circle of radius 2.0 has an area of 12.566371
```

5 Preprocessor directives

5.1 #include

The **#include** directive inputs the contents of another file into the file being processed. The included file is also pre processed, so it can contain comments and directives. The **#include** directive operates in a similar to the **#include** directive for c and c++ compilers. The syntax of the **#include** directive is:
#include <*filename.ext*>
#include *"filename.ext"*

The difference between the <*filename.ext*> and *"filename.ext"* formats is the search algorithm used to find the include file. When **#include** *"filename.ext"* is specified, the file is searched for first in the current directory. If the file is not found in the current directory, the directories listed in the environment variable DC_PATH are searched in order until the file is found. If the file is not found, an error message is produced. When **#include** <*filename.ext*> is specified, the file is searched for in the directories listed by the environment variable DC_PATH. If the file is not found in these directories, an error message is produced.

```
#include "file1.ext" /* searches for the file in the local directory,
                        then in the DC_PATH directory */
```

```
#include <file2.ext> /* searches for the file in the DC_PATH directory
                      stops searching when first match is found. */
```

5.2 #define

The **#define** directive is used to define values for substitution and for **#ifdef** directives (section 5.4). Values defined using an **#define** directive are substituted into the input file when the value found. The syntax for a **#define** directive is:
#define *Name Value*
where *Name* is a character string, and *Value* is a character string, integer, or floating point value. The defined value will be substituted into the file where ever *Name* is encountered. *Name* can also be used in **#ifdef** directives. When the command line option *-v Name Value* is used, values are defined in this way.

If the syntax **#define** *Name* is used for the directive, then the *Name* is defined for use with **#ifdef** directives only. When the command line option *-d Name* is used, values are defined in this way. Use of *Name* outside of an **#ifdef** directive will result in *NAME_DEFINED_WITHOUT_A_VALUE* will be substituted in for the *Name*.

Using the **#define** directive, the code

```
/* define_example.inp */
#define ENERGY 100    // define a value
#define ETEMP  10    // define another value
#define EMARGIN 1.1   // define yet another value
#define ECUT ENERGY*EMARGIN    // define a value in terms of other defines
for input e=ENERGY with margin EMARGIN the cut off is ECUT
note the mathematical operations of:
addition:       ETEMP+EMARGIN
subtraction:    ETEMP-EMARGIN
multiplication: ETEMP*EMARGIN
division:       ETEMP/EMARGIN
Note, division is not allowed since it would be confused with the LAHET
end of line character/
#define TEST_VALUE // a value that should exist only for an ifdef
TEST_VALUE was not defined with a value.
```

expands out to

```
for input e=100 with margin 1.1 the cut off is 110.000000
note the mathematical operations of:
addition:       11.100000
subtraction:    8.900000
multiplication: 11.000000
division:       10/1.1
Note, division is not allowed since it would be confused with the LAHET
end of line character/
TEST_VALUE_DEFINED_WITHOUT_A_VALUE was not defined with a value.
```

Unlike the c and C++ preprocessors, the **#define** directive in this preprocessor cannot be used to define macros.

5.3 #undef

The **#undef** directive is used to remove values defined by **#define** directives from the defined substitution list.

```
/* undef_example.inp */
#define ATEST 1000
atest is defined to be equal to ATEST
#undef ATEST
ATEST is no longer defined
```

evaluates to

```
atest is defined to be equal to 1000
ATEST is no longer defined
```

5.4 #ifdef - #ifndef - #elseif - #elif - #else - #endif

The **#ifdef**, **#ifndef**, **#elseif**, **#elif**, **#else**, and **#endif** are the logical directives of the preprocessor. These logical operations determine the fate of information processed the preprocessor. The syntax of the conditional directives are:
#ifdef *Name0*

lines to evaluate if Name0 is defined
#elseif *Name1*
lines to evaluate if Name1 is defined, and Name0 is **Not** *defined*
...
#elif *NameN*
lines to evaluate if NameN is defined, and Name0-Name(N-1) is **Not** *defined*
#else
lines to evaluate if Name0-NameN are **Not** *defined*
#endif

and
#ifndef *Name0*
lines to evaluate if Name0 is **Not** *defined*
#elseif *Name1*
lines to evaluate if Name1 is defined, and Name0 **is** *defined*
#else
lines to evaluate if Name0 **is** *defined and* Name1 *is* **Not** *defined*
#endif

The expression **#ifdef** *Name* evaluates to true in *Name* has been **#define**d. In this case (true), the input file lines are processed until a **#elseif**, **#else**, or **#endif** directive is encountered. The **#elseif**, **#else**, and **#endif** directives are valid only following a **#ifdef** directive. If the **#ifdef** directive evaluated to be true, t hen these directives (**#elseif**, **#else**, and **#endif**) stop processing of the input lines until a **#endif** directive is encounter. Thus, the code

```
/* simple_ifdef.inp */
#define TEST1
#define TEST2
#ifdef TEST1
test1 is defined, test2 and test3 are unchecked
#elseif TEST2
test1 is not defined, test2 is defined, test3 is unchecked
#elseif TEST3
test1 and test2 are not defined, test3 is defined
#else
```

```
test1, test2, test3 are not defined.
#endif
/* check for TEST2 alone */
#ifdef TEST2
test2 is defined
#endif
```

evaluates to be

```
test1 is defined, test2 and test3 are unchecked
test2 is defined
```

If an **#ifdef** is evaluated to be false, then the **#elseif** directive is processed as if it was an **#ifdef** directive. If all **#elseif** directives evaluate to be false, the **#else** directive is processed if it exists. Input is evaluated until an **#endif** directive is encountered. A **#endif** directive ends the **#ifdef** processing.

#ifdef directives can be nested, however, all directives in a given logical level must exist in a single file.

The following sample code showing use of the **#ifdef** directive chain with nested **#ifdef**s.

```
/* ifdef_example.inp */
#define TEST1  100 // define a variable.
#define TEST2
/* process the input based on the what was defined above */
#ifdef TEST1
TEST1 (test1) is defined
#ifdef TEST2   // another check (nested within the first)
The Value TEST2 (test2) is defined
#else          // else for #ifdef TEST2
TEST1 (test1) is defined and TEST2 is not defined
#endif        // #endif for #ifdef TEST2
#elseif TEST2
TEST1 is NOT Defined,  and TEST2 (test2) is defined
#else
TEST1 and TEST2 are not defined
#endif
```

It evaluates to be equal to

```
100 (test1) is defined
The Value TEST2_DEFINED_WITHOUT_A_VALUE (test2) is defined
```

The expression **#ifndef** *Name* evaluates to true in *Name* has not been **#define**d. Processing continues similar to an *ifdef* directive.

The **#elif** is functionally equivalent to the **#elseif** directive. These directives are interchangeable.

5.5 #echo - #error

The **#echo** and **#error** directives allows outputs the remainder of the input line to the standard input device. It was developed to allow informational messages, warning, and error messages to be presented to the user when processing the input file. used in

```
/* simple echo example */
#define ONE
#ifdef ONE                // Check of ONE is defined
#echo ONE is defined      // Output Informational message
#elseif TWO
#echo TWO is defined
#else
#error ***** WARNING ***** Neither ONE of TWO is defined ******
#endif
```

This results in "ONE is defined" being echoed to the users screen.

6 C Preprocessor Directives NOT implemented

The following directives have not been implemented in this preprocessor.

- #if
- #line
- #pragma

7 Installation Guidelines

7.1 Environment Variables

The environment variable *DC_PATH* is used to determine the search path for the **#include** directive. On Unix system, different directories are colon separated, *DC_PATH=/home/path1:path2:/path3*. The environment variable can be set in your .login, .cshrc, or in a number of other places.

On OS/2 and DOS file system the path variable are semi-colon separated, *DC_PATH=c:\home\path1;d:\path2;c:\path3*. Environment variables can be set in config.sys files, autoexec.bat, and, in WINNT, in the system configuration.

7.2 Compile time flags

When compiling for DOS, OS/2, or Windows, define MSDOS before you compile.

If you want the "old style" of comments, define OLD_COMMENTS. This will change the comment format so that a **c** at the beginning of a line starts a comment and input on a line is ignored after a **$** is encountered. This comment format was only used for a early unreleased version of the code, so you should be able to ignore this. I've kept it in the code for reverse compatablilty. The code dcomment_convert.cpp will convert comments from the pre-1.10 to 1.10 version formats.

8 Uses for dcomment

dcomment has many uses. I've listed a few here:

- Improve readability of input decks
- Defining Input variables on the fly (use *-v* and *-d* command line options with **#ifdef**'s)
- Using alternative names for input deck items to improve understanding (*ie* give materials names, then use the names on the cell cards)

```
/* materials.inh
  1/16/97: JVS:

  MATERIAL DESCRIPTIONS */
#define LEAD 1          // Material Number
#define PLEAD -11.69    // Density
0,1,1/              // Material 1: Lead
82,207,0.03299,22/  // Lead
//
// Polycarbonate: (C16_H14_03)
#define POLYCARBONATE 2         // Material Number
#define PPOLYCARBONATE -1.20    // Density
0.039789,5/       // Hydrogen
6,12,0.044973/        // Carbon-12:       0.989
6,13,0.0005/         // Carbon-13:       0.011
8,16,0.008506/        // Oxygen-16      0.99762
8,17,3.24E-05/        // Oxygen 17       0.0038
8,18,1.71E-05/        // Oxygen-18        0.002
/* .....
   many more materials are listed here */
/* end of material descriptions */
```

- Use **#define** and math operations to create geometry

```
#define START
#define THICKNESS
100 pz  START
110 pz  START+THICKNESS
120 pz  START+2*THICKNESS
...
```

- Changing in/out standard geometry parts

```
#ifdef IC_IN         /* check if IC is defined */
#include ``ion_chamber_cells.inh'' /* include ion chamber */
#endif
```

- Defining standard geometry parts (see above, can use ion_chamber in many different problems)
- Using an input file that has a common geometry for all LCS series problems through use of **#include**
- Single input file for several LCS codes by using **#ifdef**, **#define**, and the other logical directives.

```
/* sample to show different includes */
/* header to include depends on the code */
#ifdef LAHET
#include ``lcs_stage1_header.inh''
#elseif MCNP
#include ``mcnp_header.inp''
#elif SABRINA
#include ``sabrina_header.inh''
#endif
/* include the same geometry for all problems */
#include ``problem_geometry.inh''
/* can also have ifdef's for footer information */
```

- Creating complex geometrical objects
- ....