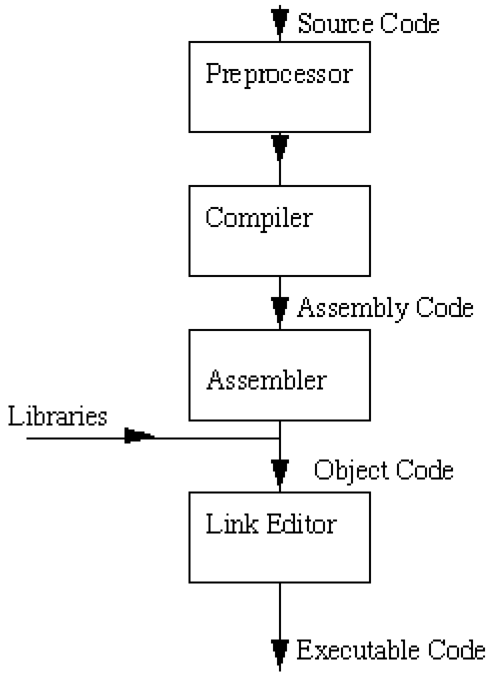
C Preprocessor

Prabhat Kumar Padhy

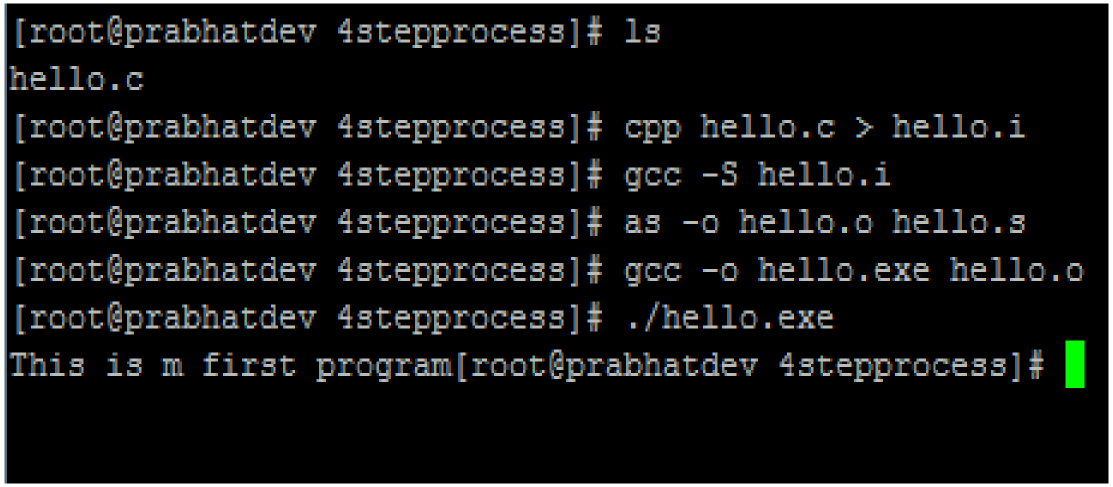
C Preprocessor?

- Creating C program, Compiling and Runnings.
  - Create using some editor
  - Compilation → gcc test.c (or) gcc –o test test.c
  - Running → ./test
- C Preprocessor
  - The preprocessor offers various features called
    - Preprocessor directive
    - Begin with "#" symbol
    - These can be placed anywhere in the program
- Use of preprocessor
  - programs easier to develop
  - easier to read
  - easier to modify
  - C code more transportable between different machine architectures.



4 step process C?

- Preprocessing
  - cpp hello.c > hello.i
- Compilation
  - gcc -S hello.i
- Assembly
  - as -o hello.o hello.s
- Linker
  - gcc -o hello.exe hello.o



```
[root@prabhatdev 4stepprocess]# ls
hello.c
[root@prabhatdev 4stepprocess]# cpp hello.c > hello.i
[root@prabhatdev 4stepprocess]# gcc -S hello.i
[root@prabhatdev 4stepprocess]# as -o hello.o hello.s
[root@prabhatdev 4stepprocess]# gcc -o hello.exe hello.o
[root@prabhatdev 4stepprocess]# ./hello.exe
This is m first program[root@prabhatdev 4stepprocess]#
```

Macro expansion?

- Macro expansion
  - A simple macro is kind of abbreviation
  - It is a name which stands for a fragment of code.
  - It can also be referred as *manifest constants*.
- Ex1

```
foo = X;
#define X 4
bar = X;
```

- Ex2

```
#define BUFSIZE 1020
#define TABLESIZE BUFSIZE
```

- EX3 – for finding min of two numbers

```
#define min(X, Y)  ((X) < (Y) ? (X) : (Y))
```

Macros?

- Standard Macros

```
#define PI 3.1415
int main ()
{
  float r=6.25;
  float area;
  area = PI*r*r;
  printf("\nArea =
%f",area);
  return 0;
}
```

- Macros with Arguements

```
#define PI 3.1415
#define AREA(x) (PI*x*x)
int main ()
{
  float r1=6.25, r2=2.5,a;
  a= AREA(r1);
  printf("\nArea = %f",a);
  a= AREA(r1);
 printf("\nArea = %f",a);
  return 0;
}
```

- Predefined Macro
  - Standard Predefined Macro

    __FILE__ → corresponds to the name of current input file

    __LINE__ → current input line no

    Usefule in generating error messages

    __DATE__, __TIME__ etc..
  - Non-standard predefined Macro

    unix, BSD, sun etc… are example of some non-standard predefined macro for the machine names..

Macro Vs Function?

- Macro call the preprocessor and replaces macro template with macro expansion
- In function call, control is passed to a function along with certain arguments, some calculation is performed and value is returned.
- Which is better ? Macros make the program run faster but increases program size, whereas functions make the program smaller and compact but increases execution time.
- Programmer should do judicious job, by doing a trade off between these two.
- Too many macros will unnecessarily make the code size longer.

File Inclusion?

- File inclusion
  - #include <file> or #include “file”

header.h, file contains

```
char *test ();
```

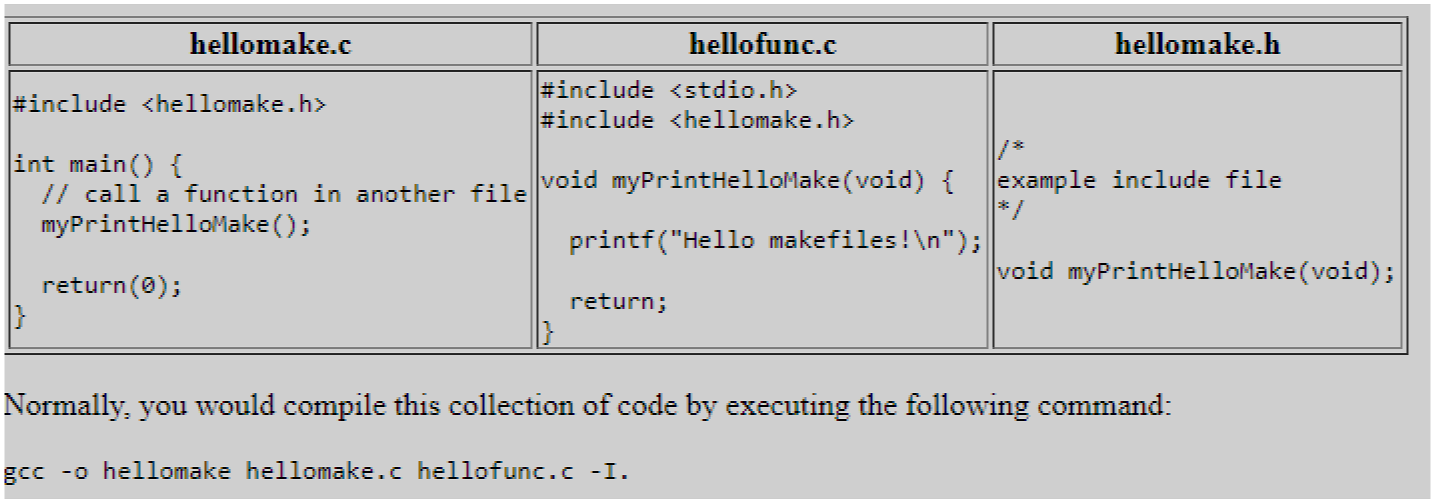
program.c, file contains

```
int x;
#include "header.h"
main ()
{
  printf (test ());
}
```

- the output generated by the C preprocessor for `program.c' as input would be

```
int x;
char *test ();
main ()
{
  printf (test ());
}
```

Macros?



hellomake.c

```
#include <hellomake.h>

int main() {
  // call a function in another file
  myPrintHelloMake();

  return(0);
}
```

hellofunc.c

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

void myPrintHelloMake(void) {

  printf("Hello makefiles!\n");

  return;
}
```

hellomake.h

```
/*
example include file
*/

void myPrintHelloMake(void);
```

Normally, you would compile this collection of code by executing the following command:

```
gcc -o hellomake hellomake.c hellofunc.c -I.
```

- In the above example, hellomake.h file is included which is a user defined header file. Similarly depending on the need many files can be defined as part of any project

Conditional Compilation?

- Conditional Compilation
  - In a macro processor, a *conditional* is a directive that allows a part of the program to be ignored during compilation, on some conditions.
- Usage ??
  - A program may need to use different code depending on the machine or operating system it is to run on.
  - You may want to be able to compile the same source file into two different programs.
  - A conditional whose condition is always false is a good way to exclude code from the program but keep it as a sort of comment for future reference.
- Syntax of conditional Usage ??
  - The “#if” directive
  - The `#else' Directive
  - The `#elif' Directive

Conditional Compilation?

- The "#if" directive

  #if *expression controlled*

  text

  #endif /* *expression* */

- The "#else" directive

```
#if expression
text-if-true
#else /* Not expression */
text-if-false
#endif /* Not expression */
```

- The "#elif" directive

```
#if X == 1
..
#elif X == 2
...
#else /* X != 2 and X != 1*/
...
#endif /* X != 2 and X != 1*/
```

Conditonal and Macro?

- EX1
  - #if BUFSIZE == 1020 printf ("Large buffers!\n"); #endif /* BUFSIZE is large */
- EX2
  - #if defined (vax) || defined (ns16000)
- EX3

#ifdef name
is equivalent to ` #if defined (name)'.
#ifndef name
is equivalent to ` #if ! defined (name)'

Miscellaneous Directive?

- #undef Directive
  - #include <file> or #include “file”

#def A 3

#undef A

- #pragma Directive: This is another special purpose directive which can be used to turn on / off certain features.
  - It varies from compiler to compiler
  - For ex: write assembly language statements in C program.

Miscellaneous?

- Miscellaneous directives such as line control, erroring out etc..
- EX1

```
#ifdef __FILE__
#error "Won't work on this file. "
#endif
```

- EX2

```
#if HASH_TABLE_SIZE % 2 == 0 || HASH_TABLE_SIZE % 3 == 0 \
    || HASH_TABLE_SIZE % 5 == 0
#error HASH_TABLE_SIZE should not be divisible by a small prime
#endif
```

Examples / Questions?

- Ex-1

```
#define NO
#define YES
int main ()
{
  int i=5,j;
  if(i>5)
     j = YES;
  else
     j = NO;
printf("%d",j);
  return 0;
}
```

- EX - 2

```
#define HELLO(m) printf("m")
#define NEXTLINE printf("\n"):
int main ()
{
  HELLO("The World !!");
  NEXTLINE;
  HELLO("It is going to End");
}
```

- Ex-3

```
#define ISUPPER(x) (x>=
65 && x<= 90)
#define ISLOWER(x) (x>=
97 && x<= 122)
#define ISALPPHA(x)
(ISUPPER(x) ||
ISLOWER(x))
int main()
{
char ch = '+';
if(ISALPHA(ch))
          printf("ch contains an
alphabet");
else
           printf("ch does not
contain an alphabet");
}
```

short number,sum; int bignumber,bigsum; char letter;   main() {   }

Examples / Questions?

- Ex-4

```
#define THIS
#define THAT
int main ()
{
  #ifdef THIS
        #ifdef THAT
        printf("Truth is hard to
digest");
        #else
         printf("But once realized,
hard to forget");
   #endif
return 0;
}
```

- EX - 5

```
#define CUBE(x) (x*x*x)
int main ()
{
  int a,b=3;
  a= CUBE(b++)/b++;
Printf("a=%d b=%d",a,b);
}
#define CUBE(x) (x*x*x)
int main ()
{
  int a,b=3;
  a= CUBE(++b)/++b;
Printf("a=%d b=%d",a,b);
}
```

- Ex-6

```
#define THIS
#define THAT
int main ()
{
  #ifdef THIS && THAT
          printf("Truth is hard
to digest");
  #else
          printf("But once
realized, hard to forget");
   #endif
return 0;
}
```

short number,sum; int bignumber,bigsum; char letter;   main() {   }

Examples / Questions?

- Ex-7

```
#define COND (a >==65 &&
a<= 90)
int main ()
{
  char a='R';
  if(COND)
        printf("UPPER CASE");
else
         printf("LOWER CASE");

return 0;
}
```

- EX - 8

```
#define AAA(format, var)
printf("var=%format\n",var);

int main ()
{
  int i=3;
  float a=3.14;
  AAA(d,i);
  AAA(f,a);
}
```

- Ex-9

```
#define DATATYPE char far
*
int main ()
{
  DATATYPE s;
  s= 0xb8000000;
*s = 'A';
         return 0;
}
```

Questions?