C Programming

Class- BCA IInd Semester

Dr. Dharm Raj Singh

Assistant Professor, (HOD)

Department of Computer Application

Jagatpur P. G. College, Varanasi

Affiliated to Mahatma Gandhi Kashi Vidhyapith Varanasi

Email- dharmrajsingh67@yahoo.com

Outline

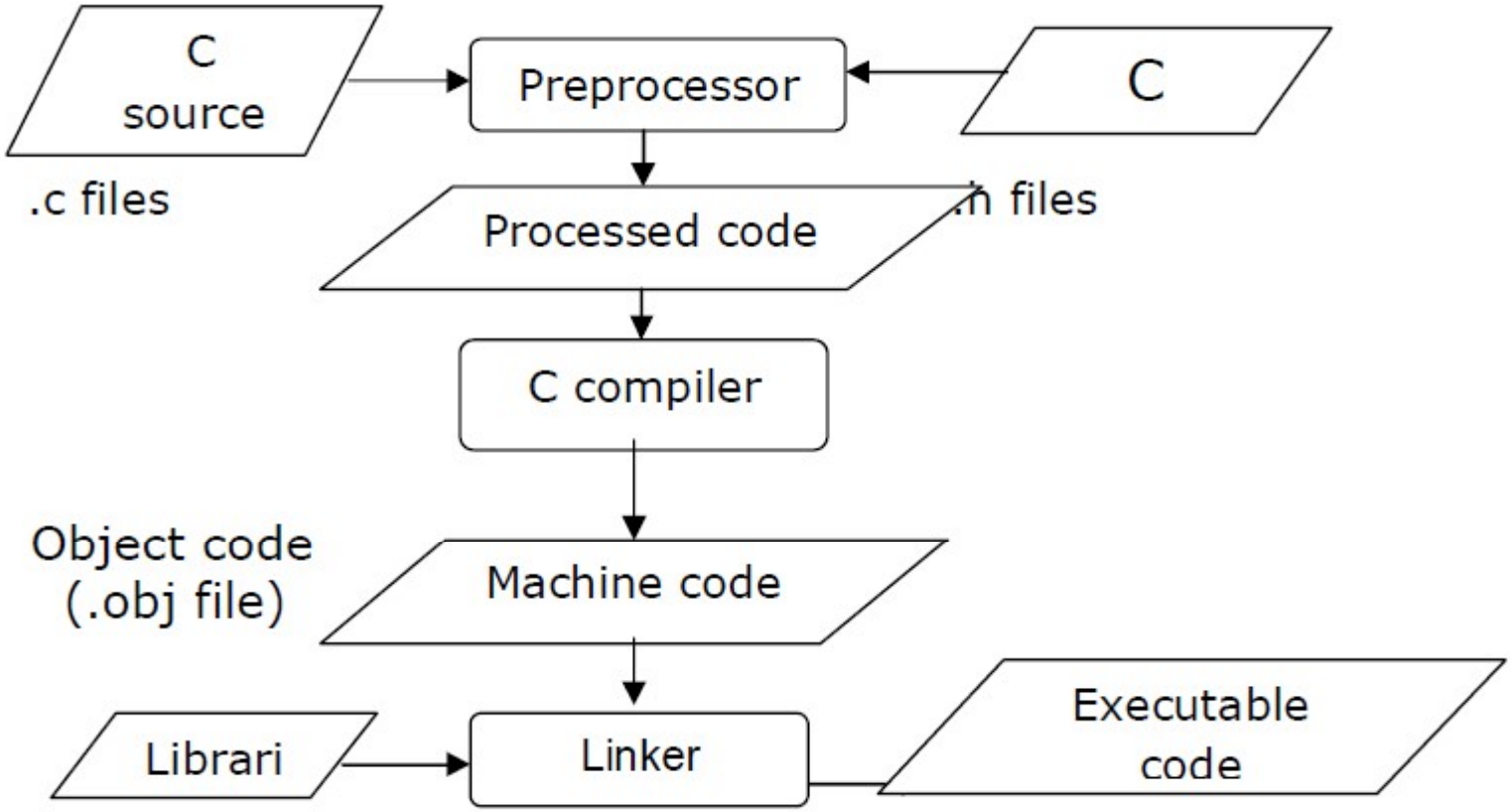
unit 5 : Preprocessors

- Preprocessors
- Macros with Arguments
- Macros versus Functions
- File Inclusion
- Conditional Compilation
- Miscellaneous Directives
- Token Pasting

5.1 Preprocessors

- A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make his program easy to read, modify, portable and more efficient.
- The C preprocessor is a collection of special statements, called directives that are executed at the beginning of the compilation process.
- Preprocessor directives follow the special syntax rules that are listed below.
  - Executed by the pre-processor.
  - Occurs before a program is compiled.
  - Begin with #.
  - Would not end with semicolon.
  - Can be placed anywhere in the program.
  - Normally placed at the beginning of the program or before any particular function.

The compilation process can be diagrammatically given as below.



We would learn the following preprocessor directives here:

(a) Macro expansion

(b) File inclusion

(c) Conditional Compilation

(d) Miscellaneous directives

Macro Expansion

```
#define UPPER 25
main( )
{
int i ;
for ( i = 1 ; i <= UPPER ; i++ )
printf ( "\n%d", i ) ;
}
```

- In this program instead of writing 25 in the **for loop we are writing** it in the form of UPPER, which has already been defined before **main( ) through the statement,**
  #define UPPER 25
- This statement is called '**macro definition**' or more commonly, just a '**macro**'.

- using **#define can produce more efficient and more easily** understandable programs**.**
- A **#define** directive is many a times used to define operators as shown below.

```
#define AND &&
#define OR ||
main( )
{
int f = 1, x = 4, y = 90 ;
if ( ( f < 5 ) AND ( x <= 20 OR y <= 45 ) )
printf ( "\nYour PC will always work fine..." ) ;
else
printf ( "\nIn front of the maintenance man" ) ;
}
```

- A **#define directive could be used even to replace a condition, as** shown below.

```
#define AND &&
#define ARANGE ( a > 25 AND a < 50 )
main( )
{
int a = 30 ;
if ( ARANGE )
printf ( "within range" ) ;
else
printf ( "out of range" ) ;
}
```

Macros with Arguments

- The macros that we have used so far are called simple macros. Macros can have arguments, just as functions can. Here is an example that illustrates this fact.

```
#define AREA(x) ( 3.14 * x * x )
main( )
{
float r1 = 6.25, r2 = 2.5, a ;
a = AREA ( r1 ) ;
printf ( "\nArea of circle = %f ", a ) ;
a = AREA ( r2 ) ;
printf ( "\nArea of circle = %f ", a ) ;
}
```

Here's the output of the program...

```
Area of circle = 122.656250
Area of circle = 19.625000
```

Macros versus Functions

- In a macro call the preprocessor replaces the macro template with

  its macro expansion, in a stupid, unthinking, literal way.
- As against this, in a function call the control is passed to a function along with certain arguments, some calculations are performed in the function and a useful value is returned back from the function.
- This brings us to a question: when is it best to use macros with arguments and when is it better to use a function?
- Usually macros make the program run faster but increase the program size, whereas functions make the program smaller and compact.

File Inclusion

- This directive causes one file to be included in another. The preprocessor command for file inclusion looks like this:

  #include "filename "

- If we have a very large program, the code is best divided into several different files, each containing a set of related functions. It is a good programming practice to keep different sections of a large program separate. These files are **#included** at the beginning of main program file.
- Actually there exist two ways to write **#include statement. These**

are:

#include "filename"

#include <filename>

- The meaning of each of these forms is given below:

#include "goto.c "

- This command would look for the file **goto.c** in the current directory as well as the specified list of directories as mentioned in the include search path that might have been set up.

#include <goto.c>

- This command would look for the file **goto.c** in the specified list of directories only.

Conditional Compilation

- if we want, have the compiler skip over part of a source code by inserting the preprocessing commands **#if and #endif,** which have the general form:

```
#if  macroname
statement 1 ;
statement 2 ;
#endif
```

If **macroname has been #defined, the block of code will be** executed as usual; otherwise not.

```
#include <stdio.h>
#include <conio.h>
 #define NUMBER  0
void main()
{
#if NUMBER==0
printf("Value of Number is: %d", NUMBER);
#endif
getch();
}
```

#else

- The #else preprocessor directive evaluates the expression or condition if condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.

- **Syntax:**

```
#if expression
Staement 1
 #else
 Staement 2
 #endif
```

Syntax with #elif

```
#if expression
    Staement 1
#elif expression
        Staement 2
#else
        Staement 2
#endif
```

Example

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 1
void main()
{
 #if NUMBER==0
printf("Value of Number is: %d",NUMBER);
#else
print("Value of Number is non-zero");
#endif
getch();
}
```

Miscellaneous Directives

- There are two more preprocessor directives available, though they are not very commonly used. They are:

(a) #undef

(b) #pragma

#undef

- To undefine a macro means to cancel its definition. This is done with the **#undef** directive.

Syntax: **#undef token**

```
#include <stdio.h>
 #define PI 3.1415
#undef PI
main()
{ printf("%f", PI);
}
```

Output: Compile Time Error: 'PI' undeclared

#pragma

#pragma startup and #pragma exit: These directives allow us to specify functions that are called upon program startup (before **main( )) or program exit (just before the program** terminates). Their usage is as follows:

- **Syntax: #pragma token**

```
#include<stdio.h>
#include<conio.h>
void func() ;
#pragma startup func
 #pragma exit func
void main(){
printf("\nI am in main");
 getch(); }
void func() {
printf("\nI am in func");
getch(); }
Output:  I am in func
                I am in main
                I am in func
```

Stringize (#)

- The stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant. This operator may be used only in a macro that has a specified argument or parameter list. For example:

```
#include <stdio.h>

#define  message_for(a, b)  \
    printf(#a " and " #b ": We love you!\n")

int main(void)
{
   message_for(Carole, Debra);
   return 0;
}
```

- it produces the following result:

Carole and Debra: We love you!

Token Pasting (##)

- The token-pasting operator (##) within a macro definition combines two arguments. It permits two separate tokens in the macro definition to be joined into a single token. For example:

```
#include <stdio.h>

#define tokenpaster(n) printf ("token" #n " = %d", token##n)

int main(void)
{
   int token34 = 40;

   tokenpaster(34);
   return 0;
}
```

- it produces the following result: token34 = 40

Exercise

1. Write a program to find out the area of circle with a micro substitution.
2. Write a program to find largest input any two numbers using a micro.
3. Write a program to declare the member of structure using a bit field data type and to display the contents of the structure.
4. Write a program to illustrate the bitwise operator.

References

- Kanetkar, Yashavant P. "Let UsC Fifth Edition." (2017).
- Kernighan, Brian W., and Dennis M. Ritchie. *The C programming language*. 2006.
- Ritchie, Dennis M., Brian W. Kernighan, and Michael E. Lesk. The C programming language. Englewood Cliffs: Prentice Hall, 1988.
- McGraw-Hill, Herbert Schildt Tata. "The Complete Reference C fourth Edition”." (2005).
- Griffiths, David, and Dawn Griffiths. Head First C: A Brain-Friendly Guide. " O'Reilly Media, Inc.", 2012.
- Programming in C-Balguruswamy
- Structured programming approach using C-Forouzah & Ceilberg Thomson learning Publication

Declaration

“The content is exclusively meant for academic purpose and for enhancing teaching and learning. Any other use for economic/commercial purpose is strictly prohibited. The users of the content shall not distribute, disseminate or share it with anyone else and its use is restricted to advancement of individual knowledge. The information provided in this e-content is authentic and best as per knowledge”.

Dr. Dharm Raj Singh
Assistant Professor, (HOD)
Department of Computer Application
JagatpurP. G. College, Varanasi

Thanks