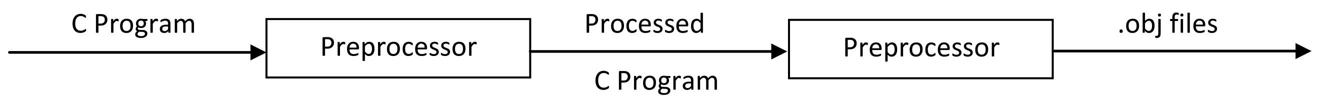
UNIT-5 (C PREPROCESSOR & BITWISE OPERATORS)
(RAM GOPAL GUPTA- http://ramgopalgupta.com/)

PART-1

C Preprocessor:

The C preprocessor is not a part of the compiler but is a separate step in compilation process. The preprocessor is a program that processes the source program before it is passed to the compiler. A 'C' preprocessor is a collection of special statements called directives that are executed before the starting of the compilation process. All preprocessor commands begin with a hash symbol (#).



The preprocessor directives can be divided into three categories:

1. Macro substitution
2. File inclusion
3. Conditional compilation

1. Macro Substitution:

Macro substitution has a name and replacement text, defined with **#define** directive. The preprocessor simply replaces the name of macro with replacement text from the place where the macro is defined in the source code. Macro substitution is commonly used to define symbolic constants.

Examples of macro:

```
#define PI 3.14
#define X 100
#define P printf
```

Program example1:

```
        /* Macro substitution */
Line 1-   #include<stdio.h>
Line 2-   #define NUM int
Line 3-   #define OUT printf
Line 4-   NUM main()
Line 5-   {
Line 6-   NUM a,b,c;
Line 7-   a=45;
Line 8-   b=25;
Line 9-   c=a+b;
Line 10-  OUT("Sum %d",c);
Line 11-  return 0;
Line 12-  }
```

Explanation:

- Macro has been defined at Line 1 & 2 named "NUM" & "OUT".
- "NUM" substitute int
- "OUT" substitute printf
- In line 4, 6 "NUM" is used in place of int.
- In line 10 "OUT" is used in place of printf.

Program example2:

```
/* Macro substitution */
Line 1-   #define PI 3.14
Line 2-   #include<stdio.h>
Line 3-   int main()
Line 4-   {
Line 5-   int rad;
Line 6-   float area,cir;
Line 7-   printf("Enter the radios:");
Line 8-   scanf("%d",&rad);
Line 9-   printf("Area %f",PI*rad*rad);
Line 10-  printf("\nCircumference %f",2*PI*rad);
Line 11-  return 0;
Line 12-  }
```

Explanation:

- Macro has been defined at Line1 “**PI**”.
- “**PI**” substitute the value 3.14
- In line 9, “**PI**” is used in place of 3.14

Program example3:

```
/* Macro substitution */
Line 1-   #include<stdio.h>
Line 2-   #define SQUARE(x) (x * x)
Line 3-   int main()
Line 4-   {
Line 5-    int num;
Line 6-    printf("Enter the number:");
Line 7-    scanf("%d",&num);
Line 8-    printf("Square of %d is %d\n",num, SQUARE(num));
Line 9-    return 0;
Line 10-  }
```

Explanation:

- Macro has been defined at Line 2 “**SQUARE(x) (x * x)**”.
- “**SQUARE(x) (x * x)**” substitute the process of (x * x) i,e, square of a given number
- In line 8, “**SQUARE(x) (x * x)**” is used to calculate the square of value store into variable num;

2. File Inclusion:

It is the second important preprocessor directive. It helps to import and attach the contents of one file into another automatically by the preprocessor. It is generally used on two purposes.

- If we have a very large program, then it is a better programming practice to divide different sections of the program as different files and linked together.
- If we have some commonly used functions, macros and constants then these can be defined as a separate file and then can be included into any program as we need.

We specify the file name with **#include** directive then the preprocessor automatically includes the contents of header file into the program (as simple as copy and paste)

Syntax:

#include statement can be written in two forms, that are

1. #include "file name"

and

2. #include <file name>

Program example4:

```
        /* File use.c */
Line 1-     #define NUM 20
Line 2-     #include<stdio.h>
Line 3-     void show(){
Line 4-     printf("void show() called from use.c file");
Line 5-     }
```

```
        /* File Inclusion example: finclusion.c */
Line 1-     #include "use.c"
Line 2-     int main()
Line 3-     {
Line 4-     printf("int main() called from finclusion.c file\n");
Line 5-     printf("NUM = %d\n",NUM);
Line 6-     show();
Line 7-     return 0;
Line 8-     }
```

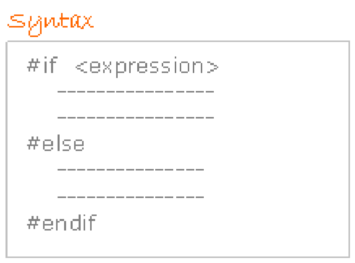
Explanation:

- In the above program example 4: there are two source code files. One is “use.c” another is “finclusion.c”.
- In source code “use.c”, we have created a Macro names “NUM” and a function “show()”. Included “stdio.h” because we are using printf(…) inside show() function.
- In source code “finclusion.c”: at line 1, we have used the file inclusion, included file “use.c” therefore code of “use.c” is included into “finclusion.c” file. No need to write code of header file (stdio.h), macro “NUM” and function “show()” in file “finclusion.c” but we can call them as we did at line 5 & 6.

3. Conditional Compilation:

As the name implies, it is the process of selecting the source code conditionally from the program and sending to the compiler using preprocessor statements like #if, #ifdef, #ifndef, #else, #elif, #endif.

#if, #else & #endif statement:



```
#if  <expression>
   ----------------
   ----------------
#else
   ---------------
   ---------------
#endif
```

Here the preprocessor selection statement #if checks the condition. If it is true or a non-zero then statements between #if and #else are sent to the compiler. If it is false or a zero then statements between #else and #endif are sent to the compiler.

Program example5:

```
/* File main.c */
Line 1-   #include <stdio.h>
Line 2-   int main()
Line 3-   {
Line 4-   int a,b;
Line 5-   printf("Enter two numbers:\n");
Line 6-   scanf("%d%d",&a,&b);
Line 7-   #if 5>10
Line 8-   printf("Sum %d",a+b);
Line 9-   #else
Line 10-  if(a==b)
Line 11-  printf("Eqauls");
Line 12-  else if(a>b)
Line 13-  printf("Biggest number %d",a);
Line 14-  else
Line 15-  printf("Biggest number %d",b);
Line 16-  #endif
Line 17-  return 0;
Line 18-  }
```

Output:

```
Enter two numbers:
45
30
Biggest number 45
```

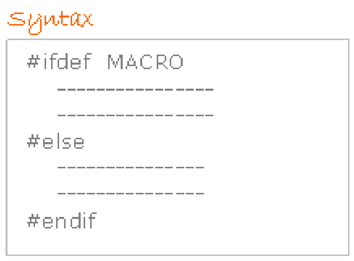
Explanation:

In the example as 5>10 gives an integer constant 0 (false), the preprocessor sends the code between #else and #endif to the compiler. Now we can realize this by looking at it’s expanded source

```
/* main.c */
main.c 2: int main()
main.c 3: {
main.c 4: int a,b;
main.c 5: printf("Enter two numbers:\n");
main.c 6: scanf("%d%d",&a,&b);
main.c 7:
main.c 8:
main.c 9:
main.c 10: if(a==b)
main.c 11: printf("Eqauls");
main.c 12: else if(a>b)
main.c 13: printf("Biggest number %d",a);
main.c 14: else
main.c 15: printf("Biggest number %d",b);
main.c 16:
main.c 17: return 0;
main.c 18: }
```

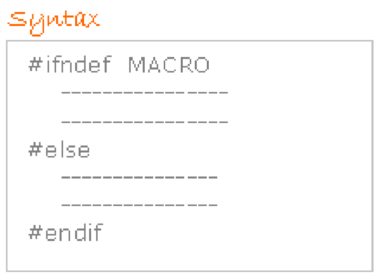
#ifdef, #ifndef preprocessor conditional statements:

#ifdef checks whether a macro with the name is defined or not. It returns true if in case macro with the name is defined otherwise returns false.
#ifndef is opposite of #ifdef.


Syntax

```
#ifdef  MACRO
   ----------------
   ----------------
#else
   ---------------
   ---------------
#endif
```

Here if the MACRO is defined with #define statement within the scope then code between #ifdef and #else is send to the compiler otherwise statements between #else and #endif are send to the compiler.


Syntax

```
#ifndef  MACRO
   ----------------
   ----------------
#else
   ---------------
   ---------------
#endif
```

Here statements between #ifndef and #else are send to the compiler if the MACRO is not defined, statements between #else and #endif are send to the compiler if MACRO is defined within the scope.

Program example6:

```
#include <stdio.h>
#define MAX
int main()
{
#ifdef MAX
printf("Hello");
#else
printf("World");
#endif
printf("\n");
#ifndef MIN
printf("Fox");
#else
printf("Duck");
#endif
return 0;
}
```

Output:

```
Hello
Fox
```

Explanation:
In this program as the macro MAX is defined "printf("Hello");" is sent to the compiler.

As the macro MIN is not defined "printf("Fox");" is sent to the compiler