UNIT :→ 5

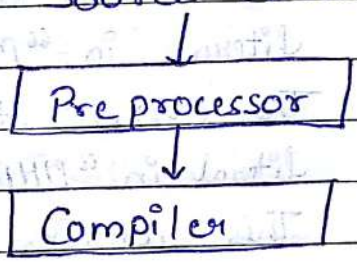
Date 6 / 4 / 18

☆ Introduction to C Preprocessor :→

'C' has a special feature of preprocessor which makes it different from other high level language that don't have his type of fecility.
Some advantage of using preprocessor are :→

1. Readability of the program is increase.
2. Program modifications eassy.
3. Makes the program portable and efficient.

We know that the code will write is translated into object code by the compiler but before being compile, The code is pass to the 'C' preprocessor. The 'C' preprocessor scan the whole source code and modify it, which is then given to the compiler.



The line starting with the symbol # are known as preprocessor directives. His some features of preprocessor directries are :→

1. Each preprocessor drictive start with a # symbol.
2. There can be only one driective on a line.
3. There is no semi-colon (;) at the end of directive.
4. The preprocessor driective can be plane anywhere

Date / /

in a program (inside or outside functions) But they are ~~program~~ usually written at the begining of the program.

☆ Derective & Description :→

Sr. No.	Directive	Discription
01	#define	Substitute a preprocessor macros
02	#include:	Insert a particular hyder from another file
03	#undef	Undefine a processor macros
04	#if	Test if a compile time condition is true
05	#else	The alternative for #if
06	#endef	end preprocessor conditioned

* Predefined Macros :→

Sr. No.	Macros	Discription
01	-DATE-	The current data as a character literal in "MMDD YY" format.
02	-TIME-	The current time as a character literal in "HHMMSS" format
03	-FILE-	This contains the current file name as a string literal.
04	-LINE-	This contains the current line numbers as the decimal constant.
05	-STDC-	Defined as 1, when the compiler compiles with the ANSI Standard.

Let us try the following example :-

```
#include<stdio.h>
main()
{
```

Date / /

```
printf("File :%s \n", __FILE__);
printf("Date :%s \n", __DATE__);
printf("Time :%s \n", __TIME__);
printf("Line :%d \n", __LINE__);
printf("ANSI :%d \n", __STDC__);
```

When the above code in a file test.c is compile and executed it produce the following result.

Output :→ File : Test.c
Date : Apr. 10 2018
Time : 10:20:15
Line : 8
ANSI : 1

☆ Preprocessor Operator :→

① Macro Continuation (\) Operator :→ This Operator is generally used to continue a macro that is too long for a single line.

② STRINGIZE (#) Operator :→ It is use with in a macro difination, converts a macro parameter into a string constant.

③ TOKEN Passing (##) Operator :→ It is use with in a macro definition & combine two arguements it permits two separate token in the macro definition to be join into a single token.

④ Including files :→ The preprocessor directive #include is use to include a file into the saurce code the

Date / /

file name should be within (< >) brackets or (" ") double quotes.

The Syntax is :→ #include <file name>
#include "file name"

★ Bitwise Operator :→

① Bitwise AND Operator (&) :→ The Output of bitwise AND is one if the corresponding bits of two operants is one (1). If either bit of an operant is 0 ; the result of corresponding bit is evaluated to 0 (zero).

Let us suppose bitwise and operation of two integer 12 and 25.

Ex:→ 12 — 0 0 0 0 1 1 0 0 (in Binary)
25 — 0 0 0 1 1 0 0 1 (in Binary)

Bitwise operation of 12 & 25 :-

```
  0 0 0 0 1 1 0 0
& 0 0 0 1 1 0 0 1
-----------------
  0 0 0 1 1 0 0 0   = 8 (in Decimal)
```

Example of Program :→

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
    int a=12, b=25;
  printf("Output = %d", a&b);
    return 0;
    getch();
  }
```

Date / /

Output :→ 8

② Bitwise OR Operator :→ The Output of bitwise OR is 1 (one), if atleast one corresponding bits of two operants is 1. In C programming bitwise OR Operator is denoted by bileline (|).

12 - 0 0 0 0 1 1 0 0 (in Binary)
25 - 0 0 0 1 1 0 0 1 (in Binary)

Bitwise OR Operation of 12 and 25

```
  0 0 0 0 1 1 0 0
  0 0 0 1 1 0 0 1
  ---------------
  0 0 0 1 1 1 0 1  = 29 (in Decimal)
```

Program Example :→

```
#include<stdio.h>
#include<conio.h>
  void main()
 {
  clrscr();
 int a=12, b=25;
printf("Output = %d", a|b);
  getch();
 }
```

Output :→ 29

③ Bitwise Compliment Operator (~) :→ Bitwise compliment Operator is an unary operator (work only one operant) its change 1 to 0 & 0 to 1. It is denoted by (~).

35 = 100011 (in Binary)

Date / /

Bitwise Compliment Operation of 35.

(~) 100011 Complement 011100

IInd Complement +1

011101

(4) Bitwise XOR (Exclusive OR) Operator (^) :→

The result of Bitwise XOR operator is one if the corresponding bit of two operants are opposite it is denoted by (^) sign.

12 - 00001100 (in Binary)

25 - 00011001 (in Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

00010101 = 21 (in decimal)

Program Example :→

```
#include <stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=12, b=25;

printf("Output = %d", a^b);
getch();
}
```

Output :→ 21