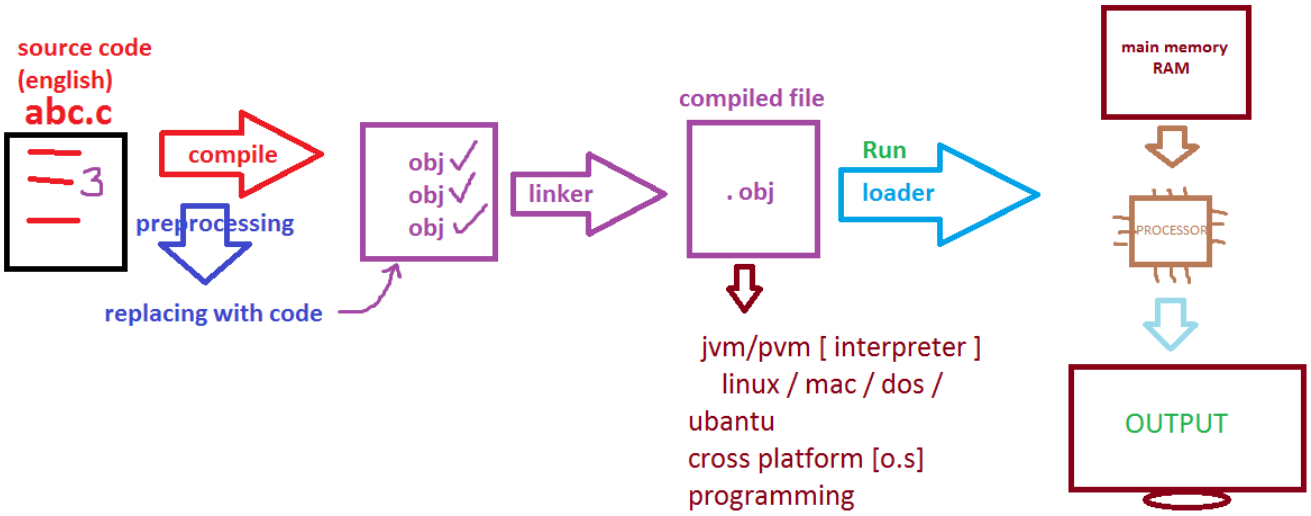
PREPROCESSOR

Preprocessor is an automated program, which will be passed for compilation, before the source program.

Compiling & Execution order of a c program:



Every preprocessor should be started with **#** symbol and it tells the compiler it is a preprocessor program.

We can declare the preprocessor at any place of our program. But it is recommended to declare at the top of program.

Preprocessor never ends with semicolon **[;]**

preprocessor programs are under control of 4 preprocessor directives. They are

1. File inclusion directives **[#include]**
2. Macro substitution directives[**#define**]
3. Conditional compilation directives[**#if, #else,..**]
4. Miscellaneous directives[**#error, #pragma,..**]

File inclusion directive [#include]

It is used to add other files to our source program.

By using this preprocessor directive, we can add the .h, .c, .cpp files etc.

Generally header files contain function definitions, macros, global variables etc.

Creating our own header file:

#include<stdio.h>

#include<conio.h>

```
void  welcome()
{
printf("Welcome to C Language\n");
}
void sum(int x ,  int y)
{
printf(“Sum=%d”,x+y);
}
```

Save the file with **.h** extension [not mandatory]

[**Eg**: abc.h]

Note: **H**eader file never contain main() .

Compile the file. Now it doesn’t shows any error. Try to run the file. It shows one error “**undefined symbol main()”**. Don’t bother about the error.

Using header file in our C- program:

Open a new c file.

Write the below program.

#include “abc.h” or #include <c:\tc\abc.h>

```
void  main()
{
welcome();
sum(10,20);
getch();
}
```

Run the program.

Note: We can use header files in **< >** or **“ “**.

When < > are used, compiler searches in include folder.

When **“ “** are used, compiler searches in current location.

When < > are used, it is better to provide path.

2. Macro substitution directive [#define]

By using this directive, we can create the symbolic constants / macro substitution text / macro's.

Whenever **#define** is occurred in our program, the identifier is replaced with the substitution text.

Here we should have to maintain at least one space in between the identifier and replacement text.

Syntax:

#define identifier replacementtext

Eg:

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

#define  p   printf

#define  pi  3.14

#define  imp  main()
```

```
void   imp
{
float  r = 10;
clrscr();
p("Circle area = %.2f", pi * r * r);
getch();
}
```

Save the file. [eg: xyz.c] and run.

Press alt+f for file menu → select OS / DOS shell

Now the dos prompt is displayed like this.

C:\TC>

C:\TC> cpp   xyz.c

Now it creates **xyz.i** file [intermediate]

C:\TC> type xyz.i enterkey

Note: The .i file contains information, which is participating in execution.

Type exit and press enter key to return to the

C-Editor.

Macro:

Macro is a simplified function.

When a function is completed within one or two lines, it is called simplified function.

Advantages:

They are faster than normal functions.

They doesn't occupy any physical memory.

In macros, in place of bouncing [jumping] process, substitution or replacement is occurred.

They reduce program size.

Drawbacks:

1. Control statements are not allowed.
2. No compilation issues. [no syntax checking]
3. No parameter type checking.
4. They can’t return any value.

Eg:

```
#include<stdio.h>
#include<conio.h>
#define    sum(a, b)     a + b
void main()
{
clrscr();
printf(“sum = %d\n”, sum(10, 20));
printf(“sum = %.2f”, sum(1.2, 5);
getch();
}
```

Output:

Sum = 30

Sum = 6.20

Eg: 2

```
#include<stdio.h>
#include<conio.h>
#define     max( a, b )        a>b ? a : b
void main()
{
clrscr();
printf(“Big = %d\n” , max(7, 25));
printf(“Big = %f”, max(2.3,7.7));
getch();
}
```

Eg: 3

```
#include<stdio.h>
#include<conio.h>
#define  sqr1(a)    a * a
#define  sqr2(a)   (a) * (a)
#define  cube(a)    sqr2(a) * (a)
void main()
{
clrscr();
printf(" sqr = %d\n", sqr1(5));  ➔ 25
printf("sqr = %d\n", sqr1(2+3));  ➔ 11
printf("sqr = %d\n", sqr2(2+3)); ➔ 25
printf("cube = %d", cube(5)); ➔ 125
getch();
}
```

Eg: 4

```
#include<stdio.h>
#include<conio.h>
#define  sqr(a)  a * a
void  main()
{
int  a = 64/sqr(8);
printf("a=%d",a);
getch();
}
```

Output: a=64

Explanation: a = 64/8*8 ➔ (64/8)*8 ➔ 8*8=64

Eg:

```
/* pre defined macros */
#include<stdio.h>
#include<conio.h>
```

```
void  main()
{
clrscr();
printf("File name: %s\n",__FILE__);
printf("Date: %s\n",__DATE__);
printf("Time: %s\n",__TIME__);
printf("Line No: %d",__LINE__);
getch();
}
```

3. Conditional compilation directives:

Conditional compilation directives are similar to if, else , else if in our programs.

In general if condition, regardless of condition, all the statements (both true & false) are participated in exe file creation. But in conditional compilation directives only the true part statements are

participated in .exe file. Due to this exe file size is reduced and execution speed is increased.

We are having several conditional compilation directives such as

#if, #else, #elif, #endif, #ifdef, #ifndef , #undef.

Eg:

```
#include<stdio.h>
#include<conio.h>
#define   a   10
#define   b   20
void main()
{
#if  a>b
printf("a is big");
#elif  b>a
```

printf("b is big");

#else

printf("Both are Equal");

#endif

getch();

}

Save the file [eg: test1.c]

Ctrl+f9 - run

Press alt+f and select os shell / dos shell.

C:\TC> cpp test1.c enterkey

C:\TC> Type test1.i enterkey

Eg:

#include<stdio.h>
#include<conio.h>

#define a 10

```
void main()
{
#ifdef   a
printf(“a is available\n”);
#undef  a
printf(“a is deleted\n”);
#endif
#ifndef  a
printf(“a is not available”);
#endif
getch();
}
```

Miscellaneous directives:

#error: It is used to create user defined errors.

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

```
#include<conio.h>
void  main()
{
#ifndef   pi
#error   pi not defined
#else
printf("pi is defined");
#endif
getch();
}
```

#pragma:

It is a miscellaneous directive.

It is completely depended on the compiler. i.e. some compilers supports and some not.

It is used to suppress warning messages.

it is also used to set the functions sequence.

```
#include<stdio.h>
#include<conio.h>
#pragma  warn   -rch        /*unreachable code*/
#pragma  warn   -rvl      /*return value*/
#pragma  warn   -par     /* parameter not used*/
int   fun(int  a)
{
printf("Good bye");
}
int  main()
{
clrscr();
fun(10);
return 0;
```

```
printf("Hello");
}
```

Eg:2

```
#include<stdio.h>
#include<conio.h>
void  begin()
{
clrscr();
printf("Welcome to C\n");
}
void  end()
{
printf("Good bye to C\n");
getch();
}
```

```
#pragma  startup  begin
#pragma  exit     bye
void  main()
{
printf("Thank you \n");
}
```