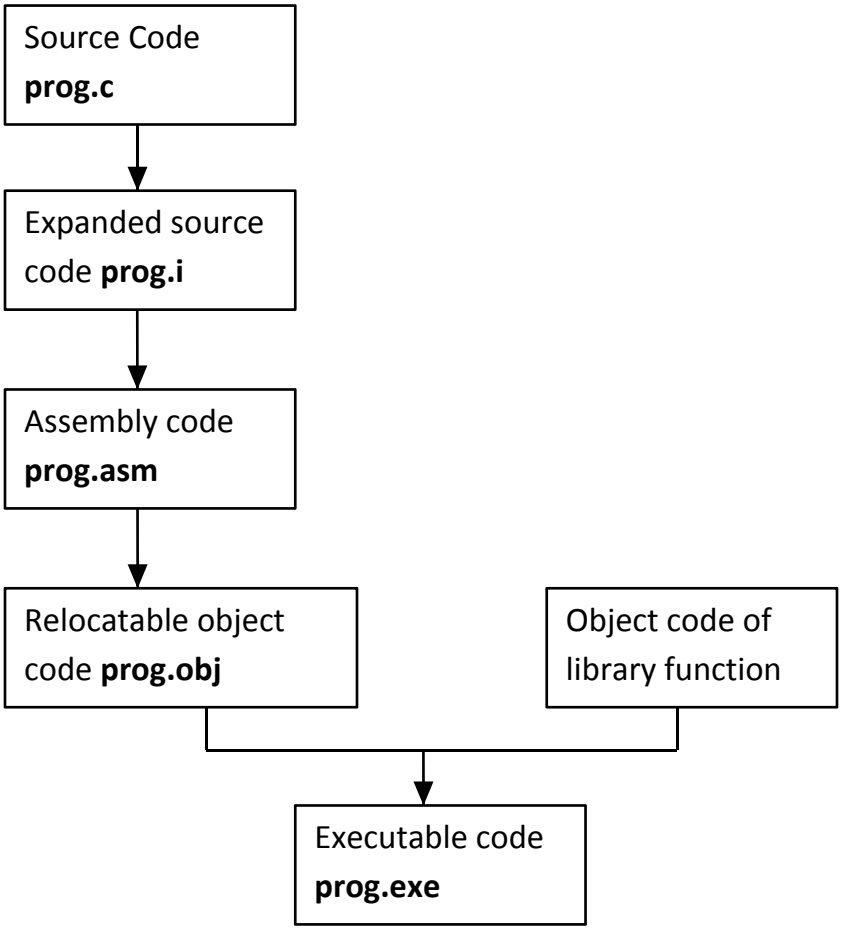
C PREPROCESSOR

- Preprocessor is exactly what its name implies.
- It is a program that process the source program before it is passed to the compiler.
- Preprocessor contains certain commands in it, to process the source code. These commands are known as **"DIRECTIVE"**.
- All these commands together, can be considered as a language with in c- language.
- Without having knowledge about preprocessor , all c- programmers depend on it. **This feature does not exist in other higher level languages.**



Each of the preprocessor directive begins with a # symbol.

There directives can be placed any where in the program, but generally these are written in the beginning of the program, before main() or before a particular function.

Some of the preprocessor directives are:

1. #include directive (file inclusion)
2. #define directive (macro substitution)
3. #undef directive
4. Conditional compilation directives

i) #define directive:

```
#define RANGE 10
main()
{
int k;
for(k=1;k<=RANGE; k++)
{
printf("k=%d\n",k);
}
}
```

Value is constant throughout the program.

In the above program, instead of writing 10, in for loop we are writing as RANGE

RANGE is also defined before main() using statement.

#define RANGE 10
The above statement is called as **MACRO DEFINATION**
During preprocessing, preprocessor replaces, each occurrence of word RANGE with 10.

```
#define A 25
main()
{
int i,j;
printf("enter i value");
scanf("%d",&i);
j=A*i;
printf("product=%d",j);
}
```

Here ; Range & A are called as “Macro templates”.
10 & 25 are called as “Macro Expansions”.

When we compile the program, before source code passes to compiler, it is evaluated by preprocessor.

- Preprocessors checks for the Macro definitions.
- According to the Macro definition given, preprocessor searches the entire for “Macro templates”.
- Whenever, it finds these “Macro templates”, preprocessor replaces them with “Macro expansions”.

- After completion of this procedure, program will be sent to compiler.

```
#define VALUE 3+5
main()
{
int i;
clrscr();
i=VALUE * VALUE;
printf("%d",i);
}
```

i=3+5*3+5
i=3+15+5
i=23

Reason: * has priority over +
First '*' is performed, and then '+' is performed.

NOTE:

(i) Generally, capital letters are used for writing Macro templates. (Small letters can also be used)

(ii) Macro template & Macro expansion are separated by space.

```
#define   RANGE   10
            ↓       ↓
          M.T     M.E
```

MACROS WITH ARGUMENTS:

Macros can have arguments, just like functions

```
#define AREA(x) (3.14*x*x)
main()
{
float i1=2.2,i2=3.4,j;
clrscr();
j=AREA(i1);
printf("Area=%f",j);
j=AREA(i2);
printf("Area=%f",j);
}
```

Preprocessors will replace AREA(x) with 3.14*x*x
Also i1&i2 will be substituted, based on function calls
J=AREA(i1) -> when this function call is executed, i1 is assigned to x.

J=AREA(i2) ->i2 will be assigned to x.

Difference between Macro and Functions:

In a Macro cal, preprocessor replaces, Macro template with its Macro expansion. Macro won't return any value to main().

In a function call, the control is passed to a function, with certain arguments and some value will be returned back to the calling function.

Macros makes programs to run faster, but increases the size of program, whereas, functions decreases the size of program.

```
#define AREA(x) (3.4*x*x)
main()
{
float i=2.2,t;
clrscr();
t=AREA(i);
printf("Area=%f",t);
}
```

1. Preprocessor will replace AREA(x) with 3.14*x*x
2. When function call AREA(i) is made, then i is assigned to x and result is calculated as 3.14*i*i

ii) File Inclusion:

- **It is a process of inserting external files, containing functions into the c-program**
- **This avoids re-writing those functions in the program.**

Suppose we have to use same function in 5 diff programs, then file inclusion technique can be used.

An external file can be loaded into the c- program using #include directive.

General form of file inclusion is:

```
#include filename
```

#include → preprocessor directive for file inclusion

Filename → name of the file containing the required function define to be included in a c-program.

If filename is declared as: #include<filename>

then, the specified file is searched only in standard directories.

If filename is declared as: #inlcude "filename"

then, the specified file is first searched in the current directories and then in the standard directories.

iii) #undef Directive

On some occasions it may be desirable to cause a defined name to become 'undefined'. This can be accomplished by means of the #undef directive. In order to undefined a macro which has been earlier #defined, the directive,

```
#undef macro-template
```

can be used.

EXAMPLE: #define PI 3.1414

```
Void main()
{
float a,r=3;
clrscr();
 a=PI*3*3;
 printf("AREA = %d",a);
 #undef PI
 a=PI*3*3;  /* Gives error undefined PI*/
  printf("Area =%d",a);
  getch();
}
```

For this program it gives error because the macro PI is undefined. After undefining macro we are using that macro in the program therefore it gives error.

iv) Conditional Compilation:

It is a process of selecting a particular segment of code for compilation based on condition.

```
main()
{
#define one
#ifdef 0one
printf("hi");
printf("hellow");
#else
printf("c-lab");
#endif
printf("welcome");
}
```

Output:
i) hi hellow welcome
ii) c-lab welcome (if #define is commented)

- c- preprocessor provides a compilation directive which is used to select alternate segment of code depending upon the condition.
- If there are 2 different versions of a program, it will take more memory to store the program. This problem can be solved by including both version in a single program.
- Then a particular version can be selected based on the requirement.
- In large programs, generally programmers writes comments for each and every statement. If we use comment, instead of conditional compilation directives, we would end up with nested comments. Sometimes, it is not possible to write nested comments in C.

Function Returning Non-integers:

C function returns a value of type ‘int’ as a default value, when no return type is specified.

Two steps must be performed ,to enables a calling function, to receive a non-integer value from the called function:

1. Return-type of the function should be specified in the function header.
2. The called function must be declared at the start of body, in the calling function, like any other variable. This is to tell the calling function, the type of data, that the function is actually returing,.

```
float m(float x, float y);
main()
{
float a,b, mul();
a=12.34;
b=7.32;
printf("%f", mul(a,b));
}
float mul(float x, float y)
{
return x*y;
}
```