Leaning Objective:

In this Module you will be learning the following:

- command–line arguments
- Preprocessor directives

Introduction:

If any input value (or commands)is passed through the command prompt at the time of running of program is known as command line argument. It is a concept to passing the arguments to the main() function by using command prompt.

Preprocessor directives are lines included in a program that being with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation.

Material:

main() function of a C program accepts arguments from command line or from other shell scripts by using the concept of command–line arguments.

Syntax: int main(int argc, char *argv[])

Here argc counts the number of arguments on the command line and argv[ ] is a pointer array which holds pointers of type char which points to the arguments passed to the program.

In real time application, it will happen to pass arguments to the main program itself. These arguments are passed to the main () function while executing binary file from command line.

When Use Command Line Argument

When you need to developing an application for DOS operating system then in that case command line arguments are used. DOS operating system is a command interface operating system so by using command we execute the program. With the help of command line arguments we can create our own commands.

Example: C Program to add two numbers using Command Line Arguments.

```
#include<stdio.h>
void main(int argc, char * argv[]) {
        int i, sum = 0;
        if (argc != 3) {
             printf("You have forgot to type numbers.");
             exit(1);
        }
        printf("The sum is : ");
        for (i = 1; i < argc; i++)
             sum = sum + atoi(argv[i]);
        printf("%d", sum);
}
```

Output: The sum is: 30

Compile and run CMD programs

Command line arguments are not compile and run like normal C programs, these programs are compile and run on command prompt. To Compile and Link Command Line Program we need TCC Command.

First open command prompt
Follow you directory where your code saved.
For compile -> C:/TC/BIN>TCC mycmd.c
For run -> C:/TC/BIN>mycmd 10 20

Explanation: Here mycmd is your program file name and TCC is a Command. In "mycmd 10 20" statement we pass two arguments.

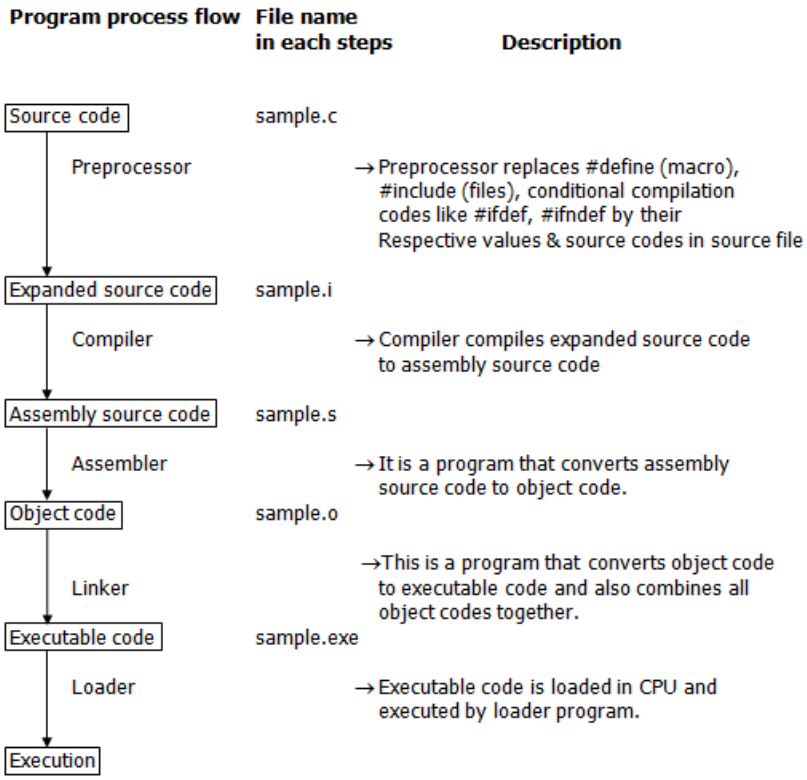
Preprocessor directives

- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.
- Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol.

Below is the list of preprocessor directives that C programming language offers.

Preprocessor	Syntax/Description
Macro	**Syntax:** #define This macro defines constant value and can be any of the basic data types.
Header file inclusion	**Syntax:** #include <file_name> The source code of the file "file_name" is included in the main program at the specified place.
Conditional compilation	**Syntax:** #ifdef, #endif, #if, #else, #ifndef Set of commands are included or excluded in source program before compilation with respect to the condition.
Other directives	**Syntax:** #undef, #pragma #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program.

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.



There are 4 regions of memory which are created by a compiled C program. They are,

1. **First region** – This is the memory region which holds the executable code of the program.
2. **2nd region** – In this memory region, global variables are stored.
3. **3rd region** – stack
4. **4th region** – heap

EXAMPLE PROGRAM FOR #DEFINE, #INCLUDE PREPROCESSORS IN C LANGUAGE:

- #define **–** This macro defines constant value and can be any of the basic data types.
- #include <file_name> **–** The source code of the file "file_name" is included in the main C program where "#include <file_name>" is mentioned.

```
#include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

void main(){
        printf("value of height    : %d \n", height );
        printf("value of number : %f \n", number );
        printf("value of letter : %c \n", letter );
        printf("value of letter_sequence : %s \n", letter_sequence);
        printf("value of backslash_char  : %c \n", backslash_char);
}
```

Output

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

EXAMPLE PROGRAM FOR #IFDEF, #ELSE AND #ENDIF IN C:

- "#ifdef" directive checks whether particular macro is defined or not. If it is defined, "If" clause statements are included in source file.
- Otherwise, "else" clause statements are included in source file for compilation and execution.

```
#include <stdio.h>
#define RAJU 100

int main() {
  #ifdef RAJU
        printf("RAJU is defined. So, this line will be added in this C file\n");
  #else
        printf("RAJU is not defined\n");
  #endif
  return 0;
}
```

OUTPUT:

```
RAJU is defined. So, this line will be added in this C file
```

EXAMPLE PROGRAM FOR #IFNDEF AND #ENDIF IN C:

- #ifndef exactly acts as reverse as #ifdef directive. If particular macro is not defined, "If" clause statements are included in source file.
- Otherwise, else clause statements are included in source file for compilation and execution.

```
#include <stdio.h>
#define RAJU 100
void main(){
  #ifndef SELVA
  {
    printf("SELVA is not defined. So, now we are going to define here\n");
    #define SELVA 300
  }
  #else
       printf("SELVA is already defined in the program”);
   #endif
  }
```

OUTPUT:

SELVA is not defined. So, now we are going to define here

EXAMPLE PROGRAM FOR UNDEF IN C LANGUAGE:

This directive undefines existing macro in the program.

```
#include <stdio.h>
 #define height 100
void main() {
  printf("First defined value for height    : %d\n",height);
  #undef height          // undefining variable
  #define height 600     // redefining the same for new value
  printf("value of height after undef \& redefine:%d",height);
}
```

OUTPUT:

First defined value for height : 100
value of height after undef & redefine : 600

EXAMPLE PROGRAM FOR PRAGMA IN C LANGUAGE:

Pragma is used to call a function before and after main function in a C program.

```
#include <stdio.h>

void function1( ) {
  printf("\nFunction1 is called before main function call");
}
void function2( ) {
  printf ( "\nFunction2 is called just before end of " \
         "main function" ) ;"
}

#pragma startup function1
#pragma exit function2

int main( ){
  printf ( "\n Now we are in main function" ) ;
  return 0;
}
```

OUTPUT:

Function1 is called before main function call
Now we are in main function
Function2 is called just before end of main function

Problem Set

1. Write C Program to find out the factorial of a given number using command–line arguments?
2. Write C Program to perform all arithmetic operations (+,-,*,/,%)using command–line arguments?
   Note: send two number and operator using command–line arguments.
3. Write C Program to display the message (“Welcome to Prepo”)
4. Write C Program to find out the square of a given number using macros?
5. Write C Program to find out the maximum of two given numbers using macros?