Dr.SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE,

(AUTONOMOUS)

COIMBATORE-641049

Accredited by NAAC (Cycle III) with "A+" Grade Recognized

by UGC, Approved by AICTE, New Delhi andAffiliated to

Bharathiar University, Coimbatore.

DEPARTMENT OF COMPUTER APPLICATIONS

Course Code / Course Name: **23UCU401 /Programming in C**

YEAR	:**2023-2024**
CLASS	:**I BCA "A"**
STAFF NAME	:**Dr.A.DEVI**
Topic	: **Structure of C Language program**

Structure of C Language program

1) Comment line
2) Preprocessor directive
3) Global variable declaration
4) main function( )
       {

          Local   variables;
  Statements;
  }
  User defined function
  }
}

Comment line
It indicates the purpose of the program. It is represented as
/*…………………………..*/
Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed withinthe decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & characterconstant.

Preprocessor Directive:
    #include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition &declaration of system defined function such as printf( ), scanf( ), pow( ) etc. Generally printf() function used to display and scanf() function used to read value

Global Declaration:
This is the section where variable are declared globally so that it can be access byall the functions used in the program. And it is generally declared outside the function :

main()
It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces.
The main( ) function can be anywhere in the program but in general practice it isplaced in the first position.
Syntax   :
  main()
{
……..
……..

……..
}
The main( ) function return value when it declared by data type asint main( )
{
return 0   }
The main function does not return any value when void (means null/empty) asvoid main(void)
or void main()
{
 printf ("C language");
}
Output: C language
The program execution start with opening braces and end with closing brace.
And in between the two braces declaration part as well as executable part is mentioned. And at
the end of each line, the semi-colon is given which indicatesstatement termination.
/*First c program with return statement*/
#include <stdio.h>int
main (void)
{
printf ("welcome to c Programming language.\n");return 0;
}
Output: welcome to c programming language.

Steps for Compiling and executing the Programs
A compiler is a software program that analyzes a program developed in a particularcomputer language and
then translates it into a form that is suitable for execution on a particular computer system. Figure below
shows the steps that are involved inentering, compiling, and executing a
computer program developed in the C programming language and the typical Unixcommands that would
be entered from the command line.
   Step 1: The program that is to be compiled is first typed into a *file* on the computer system.
                       There are various conventions that are used for naming files, typically be
any name provided the last two characters are **".c"** or file with extension .c. So, the file name
prog1.c might be a valid filename for a C program.A text editor is usually used to enter the C
program into a file. For example, vi is apopular text editor used on Unix systems. The program
that is entered into the file is known as the ***source program*** because it represents the original form
of the program expressed in the C language.

   Step 2: After the source program has been entered into a file, then proceed to haveit compiled.
The compilation process is initiated by typing a special command on the system. When this
command is entered, the name of the file that contains the source program must also be specified.
For example, under Unix, the command to initiate program compilation is called **cc**. If we are using
the popular GNU C compiler, the command we use is **gcc**.
           Typing the line
                   gcc prog1.c or cc prog1.c
In the first step of the compilation process, the compiler examines each program
statement contained in the source program and checks it to ensure that it conforms to the syntax
and semantics of the language. If any mistakes are discovered by the compiler during this phase,
they are reported to the user and the compilation process ends right there. The errors then have to
be corrected in the source program(with the use of an editor), and the compilation process must be

restarted. Typical errors reported during this phase of compilation might be due to an expression that has unbalanced parentheses (**syntactic error**), or due to the use of a variable that is not "defined" (**semantic error**).

Step 3: When all the syntactic and semantic errors have been removed from the program, the compiler then proceeds to take each statement of the program and translate it into a "lower" form that is equivalent to assembly language program needed to perform the identical task.

Step 4: After the program has been translated the next step in the compilation process is to translate the assembly language statements into actual machine instructions. The assembler takes each assembly language statement and converts it into a binary format known as *object code*, which is then written into another file on the system. This file has the same name as the source file under Unix, with the last letter an "**o**" (**for** *object*) instead of a "**c**".
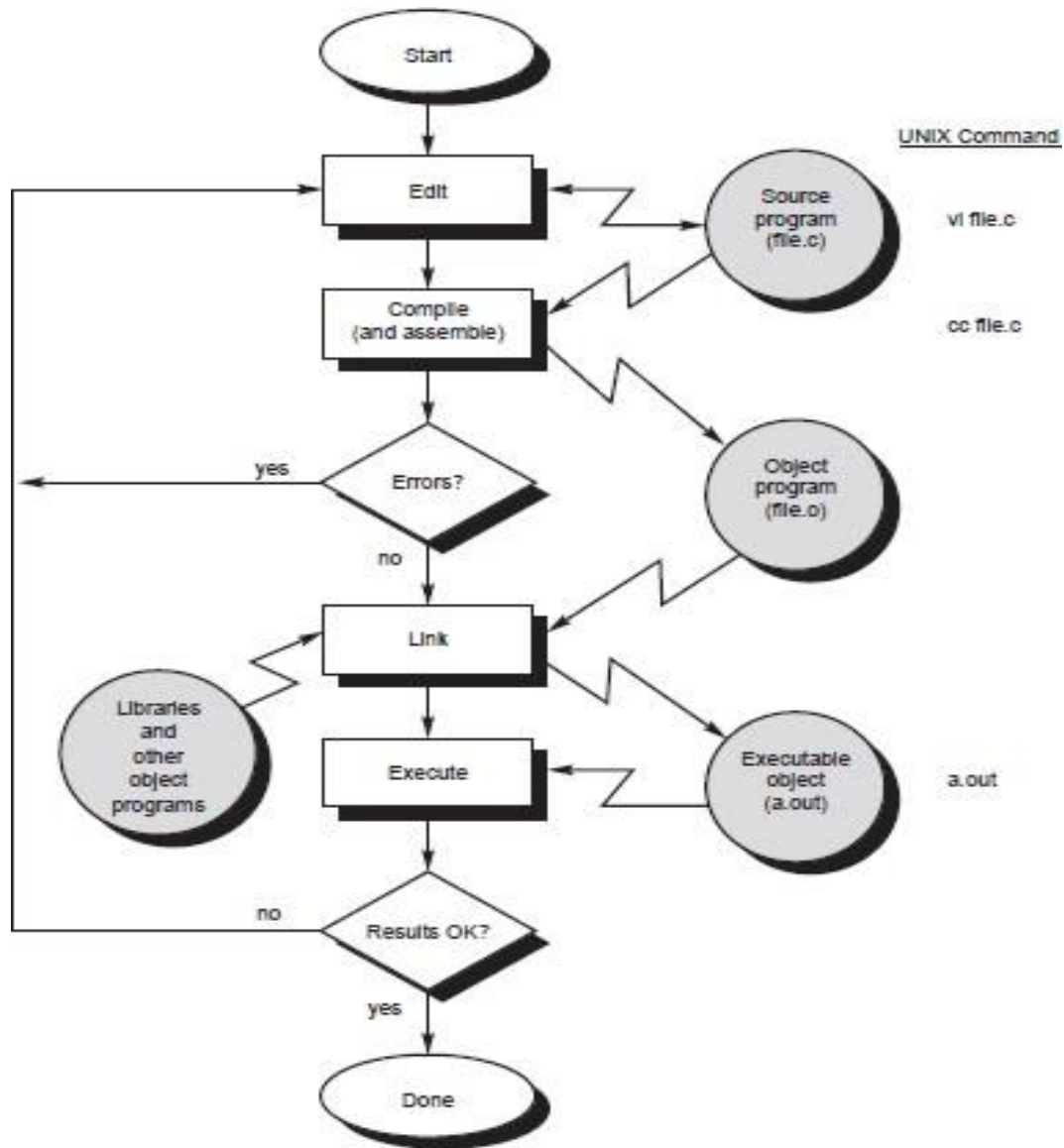
Step 5: After the program has been translated into object code, it is ready to be *linked.* This process is once again performed automatically whenever the cc or gcc command is issued under Unix. The purpose of the linking phase is to get the program into a final form for execution on the computer.

If the program uses other programs that were previously processed by the compiler, then during this phase the programs are linked together. Programs that are used from the system's program *library* are also searched and linked together with the object program during this phase.

The process of compiling and linking a program is often called *building.*

The final linked file, which is in an executable *object* code format, is stored in another file on the system, ready to be run or *executed.* Under Unix, this file is called **a.out** by default. Under Windows, the executable file usually has the same name as the source file, with the c extension replaced by an exe extension.

Step 6: To subsequently execute the program, the command **a.out** has the effect of *loading* the program called **a.out** into the computer's memory and initiating its execution.

When the program is executed, each of the statements of the program is sequentially executed in turn. If the program requests any data from the user, known as *input*, the program temporarily suspends its execution so that the input can be entered. Or, the program might simply wait for an *event*, such as a mouse being clicked, to occur. Results that are displayed by the program, known as *output*, appear in a window, sometimes called the *console*. If the program does not produce the desired results, it is necessary to go back and reanalyze the program's logic. This is known as the *debugging phase*, during which an attempt is made to remove all the known problems or *bugs* from the program. To do this, it will most likely be necessary to make changes to original source program.

Start

Edit ◄──── Source program (file.c) — UNIX Command — vi file.c

Compile (and assemble) ◄──── cc file.c

Errors? — yes ──► Object program (file.o)

no

Link ◄──── Libraries and other object programs

Execute ◄──── Executable object (a.out) — a.out

Results OK? — no / yes

Done

/* Simple program to add two numbers ....*/
#include <stdio.h> int
main (void)
{
int v1, v2, sum;   //v1,v2,sum are variables and int is data type declaredv1 = 150;
v2 = 25;
sum = v1 + v2;
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);return 0;
}
Output:
   The sum of 150 and 25 is=175