MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1112 (Problem Solving Using C)

Block 1: Foundations of Programming

Unit 2: Fundamentals of Programming

Structure	
2.0	Introduction
2.1	Objectives
2.2	Concept of C Language
2.3	Programming Format of 'C'
2.4	Creating a C Program
2.5	Compilation process in C Program
2.6	Link and Running C Program
2.7	Diagrammatic Illustration
2.8	Conclusion
2.9	Questions
2.10	References

2.0 Introduction

The C programming language stands as one of the most influential and enduring languages in the history of computer science. Developed by Dennis Ritchie at Bell Labs in the early 1970s, C quickly gained popularity due to its simplicity, efficiency, and powerful features. It has since become the foundation upon which many modern programming languages are built.

C is a general-purpose, procedural programming language known for its versatility and portability. It provides low-level access to memory, making it suitable for system programming, embedded systems, and developing operating systems. At the same time, its rich set of standard libraries and syntax simplicity make it an excellent choice for application development.

The significance of C lies not only in its widespread adoption but also in its influence on other programming languages. Many modern languages, including C++, Java, and Python, have borrowed syntax and concepts from C. Therefore, mastering C not only equips programmers with a powerful tool but also lays a solid foundation for understanding other languages and computer systems.

In this "Introduction to C Language" course, we will explore the fundamental concepts of C programming, from basic syntax to advanced features. We will cover topics such as variables, data types, control structures, functions, arrays, pointers, and memory management. Through hands-on examples and

exercises, students will gain practical experience and develop problem-solving skills essential for software development.

Whether you are a novice programmer looking to start your journey into the world of coding or an experienced developer seeking to deepen your understanding of computer systems, this course will provide you with the knowledge and skills needed to write efficient, reliable, and maintainable C code.

2.1 Objectives

After completing this unit, you will be able to:

- Designed to provide complete knowledge of C language.
- Help to develop logic which will help them to create programs and applications in C.
- Learning the basic programming constructs they can easily switch over to any other language in future.
- It aims to train the basic concepts of the C-programming language.
- This unit involves a component which is designed to give the student hands-on experience with the concepts.

2.2 Concept of C Language

The Structure of 'C' Program

A 'C' program's basic structure is separated into six components, making it easier to read, edit, document, and comprehend in a specific manner. In order to build and execute effectively, the C program must adhere to the guidelines outlined below. In a well-structured C program, debugging is easy.

1. **Sections of the C Program:** A program's proper execution is the responsibility of six basic components. The following sections are mentioned:
   a) **Documentation:** This part includes a program description, the program's name, and the program's inception date and time. It is supplied in the form of comments at the beginning of the program. Documentation can be represented in the following ways:

      // description, name of the program, programmer name, date, time etc.

      OR

      /* description, name of the program, programmer name, date, time etc. */

      Anything placed in comments will be viewed as program documentation and will not interfere with the supplied code. Essentially, it provides the reader with an overview of the software.

   b) **Preprocessor Section:** All of the program's header files will be defined in the program's preprocessing section. Header files allow us to incorporate better code from others into our own. Before the compilation process, a copy of these various files is placed into our software. Example:

      #include<stdio.h>

      #include<math.h>

   c) **Definition:** Preprocessors are programs that process our source code prior to compilation. There are several processes involved in the creation and implementation of the program. Preprocessor instructions begin with the symbol '#'. The #define preprocessor is used to declare

a constant that will be used throughout the program. When the compiler encounters this term, it replaces it with the actual piece of declared code.

Example: #define long long ll

d) **Global Declaration:** Global variables, function declarations, and static variables are all included in the global declaration section. Variables and functions specified in this scope can be utilized throughout the application.

Example: int num = 18;

e) **Main() Function:** A main function must be present in every C program. This section contains the program's main() function. Declaration and implementation are carried out within the curly braces in the main program. The return type of the main() function can be either int or void. The compiler is informed by void() main that the program will not produce any value. The main() function instructs the compiler that the code will return a numerical value.

Example: void main() OR int main()

f) **Sub Programs:** In this area of the program, user-defined functions are invoked. When they are called from the main() function or from outside the main() function, control of the program is transferred to the called function. These are specified in accordance with the programmer's needs.

Example: int sum(int x, int y) {return x+y;}

2. Structure of the C Program with Example:

Example: Below C program to find the sum of 2 numbers:

```
// Documentation
/**
* file: sum.c
* description: program to find sum.
*/
// Link
#include <stdio.h>
// Definition
#define X 20
// Global Declaration
int sum(int y);
// Main() Function
int main(void)
{
int y = 55;
printf("Sum: %d", sum(y));
return 0;
```

```
}
// Subprogram
int sum(int y)
{
return y + X;
}
```

Output:

Sum: 75

Explanation of the aforementioned Program:

The aforesaid software is explained in detail below.

Sections	Description
/ * file: sum.c * author: you * description: program to find sum. */**	It is the comment section and is part of the description section of the code.
#include<stdio.h>	Header file, which is used for standard input-output. This is the preprocessor section.
#define X 20	This is the definition section. It allows the use of constant X in the code.
int sum(int y)	This is the Global declaration section includes the function declaration that can be used anywhere in the program.
int main()	main() is the first function that is executed in the C program.
{…}	These curly braces mark the beginning and end of the main function.
printf("Sum: %d", sum(y));	printf() function is used to print the sum on the screen.
return 0;	We have used int as the return type so we have to return 0 which states that the given program is free from the error and it can be exited successfully.
int sum(int y) { return y + X; }	This is the subprogram section. It includes the user-defined functions that are called in the main() function.

Compilation and the execution of a C program involves the following steps are Program Creation, Compilation of the program, Execution of the program and output of the Program.

The following are some of the elements we acquired about the structure of the C Program in this article:

- A C program's basic structure is separated into six components, making it easier to read, edit, document, and comprehend in a specific manner.
- In a well-structured C program, debugging is easy.
- A C program is divided into six sections: Documentation, Preprocessor Section, Definition, Global Declaration, Main() Function, and Sub Programs.
- The following stages are taken during the compilation and execution of a C program:
  a) Development of a Program
  b) Putting together the program
  c) The initiative is being carried out.
  d) Result of the program

3. Syntax of C Program:

The C standard specifies a formal grammar for C. Line ends are not normally relevant in C; nevertheless, line boundaries are during the preprocessing step. Comments can appear within the delimiters /* and /, or (since C99) after // until the end of the line. Comments delimited by / and */ do not nest, and if they occur inside string or character literals, they are not interpreted as comment delimiters.

The declarations and defined functions are found in C source files. Declarations and statements are included in function definitions. Declarations either declare new types with keywords like struct, union, and enum, or assign types to and maybe reserve storage for new variables by writing a type followed by the variable name. Built-in types are specified via keywords that include char and int. Braces (and, sometimes known as "curly brackets") are used to limit the scope of declarations and to serve as a single statement for structuring controls.

C, being an imperative language, specifies actions using statements. The most frequent statement is an argument statement, which consists of an argument to be evaluated preceded by a semicolon; as a result of the assessment, functions and variables may be called and new values set. C includes many control-flow statements designated by reserved keywords to change the regular sequential execution of statements. If... [otherwise] conditional execution and do... while, while, and for sequential execution (looping) provide structured programming. The for statement has distinct initialization, evaluation, and reinitialization expressions, which can be omitted in any order. Within the loop, break and continue can be utilized. Break is used to exit the innermost contained loop statement, while continue skips to its reinitialisation. There is also a non-structured go to statement that goes directly to the function's assigned label. The value of an integer expression is used to pick a case to be performed by switch. Unlike numerous other languages, the control process will continue to the next case until interrupted by a break.

Expressions in C can employ a variety of integrated operators and call function, without no restriction on the sequence of evaluations. All side effects, including variable storage, occur before the next "sequence point," which includes the conclusion of each expression statement as well as the entrance to and return from each function call. Sequence points can also appear during the assessment of expressions that contain specific operators (&&, ||,?:, and the comma operator). This allows the compiler to optimize object code at a high level, but it needs greater effort from programmers to get accurate results. Because of the impact on existing software, the C standard was unable to remedy many of these flaws.

Features of 'C' Language

C is an imperative programming language. Dennis Ritchie was the first to create it in 1972. It was created primarily as an operational programming language for creating an operating system.

C language's core characteristics are low-level memory access, a minimal set of keywords, and an elegant style; these characteristics make C language excellent for system programming such as computer operating systems or compiler development.

What are the Most Significant C Language Characteristics?

Here are a few of the most essential C language features:

- **Procedural Language:** Predetermined instructions are carried out step by step in a procedural language like C. A C program may have several functions to execute a specific job. New programmers may believe that this is the only way a specific programming language works. Other programming paradigms exist in the programming world. An object-oriented programming language is the most often utilized paradigm.
- **Fast and Efficient:** Modern languages, such as Java and Python, have more functionality than C, but their performance rate suffers as a result of more processing in these languages. The C programming language, being a middle-level language, allows programmers to directly manipulate computer hardware, but higher-level languages do not. That is one of the reasons why the C programming language is regarded the best place to begin learning programming languages. Because statically encoded languages are quicker than dynamically typed languages, it is fast.
- **Modularity:** Modularity refers to the notion of saving C programming language code in the form of libraries for future usage. This programming language can only achieve so much on its own; the majority of its power is contained by its libraries. C has its own library for solving common issues.
- **Statically Type:** The C programming language is statically typed. That is, the type of variable is verified at compilation rather than at run time. This implies that every time a programmer writes a program, they must specify the kind of variables utilized.
- **General-Purpose Language:** The C programming language is utilized in a wide range of applications, from system programming to picture editing software. Some of the most prevalent uses are as follows:
  Operating systems: Windows, Linux, iOS, Android, OXS
  Databases: PostgreSQL, Oracle, MySQL, MS SQL Server, etc.
- **Rich set of built-in Operators:** It is a diverse language with a large range of built-in operators that may be used to write complicated or simple C applications.
- **Libraries with Rich Functions:** Robust C libraries and functions make it possible for even inexperienced programmers to develop with ease.
- **Middle-Level Language:** Because it is a middle-level language, it combines both assembly language capabilities and high-level language characteristics.
- **Portability:** C language applications are extremely portable because they can run and compile on any system with little or minor modifications.
- **Easy to Extend:** Programs built in C may be extended, which implies that after a program is written in it, additional features and actions can be added to it.

Uses of 'C' program

a) Justification for usage in system programming:

C is a programming language that is commonly used in the implementation of operating systems and embedded system applications. This is due to numerous factors:

- The code generated after compilation processes does not need many system features and can be invoked in a straightforward manner from some boot code - it is easy to execute.
- The C language statements and operators typically map properly on a sequence of commands for the target processor, resulting in a low run-time interest on system resources - it is fast to execute.
- The C language, with its extensive collection of operators, can take use of many of the capabilities of target CPUs. Where a certain CPU contains more exotic instructions, a language version may be built with perhaps inherent functions to take advantage of those instructions - it can leverage almost all of the target CPU's characteristics.
- The language allows for the simple overlay of structures over blocks of data that are binary, enabling the data to be understood, traversed, and updated - it can create data structures and even file systems.
- The language has a rich collection of operators for integer computation and logic, as well as perhaps varying sizes of floating-point values - it can process suitably organized data effectively.
- C is a little language, with only a few statements and few features that create a lot of target code - it's understandable.
- C offers immediate control over the allocation of memory and deallocation, which provides fair efficiency and predictable time to memory-handling tasks while eliminating the need to worry about periodic stop-the-world garbage disposal events - it has steady performance.
- C allows for the usage and implementation of several memory allocation strategies, such as the standard malloc and free; a complicated mechanism for various domains; or a version for an OS kernel which could suit DMA, be used in interrupt handlers, or incorporate with the virtual memory system.
- Because platform hardware can be accessible via pointers and type punning, system-specific functions (e.g. Control/Status Registers, I/O registers) may be set and utilized with C code - it interacts well with the platform on which it runs.
- Based on the linker and environment, C code may also call assembly language libraries and be called from assembly language - it collaborates well with other lower-level programs.
- C, as well as its calling conventions and linker structures, are frequently used in combination with other high-level languages, with calls to and from C supported - it interoperates well with other high-level programs.
- C has a developed and diverse ecosystem that includes frameworks, libraries, and open source compilers, debuggers, and tools, and it is the de facto standard. It is likely that the drivers already exist in C, or that a similar CPU architecture exists as a back-end of a C compiler, therefore there is less motivation to use another language.

b) Previously used for web development:

C was formerly used for web development, with the Common Gateway Interface (CGI) acting as a "gateway" to transfer data between the web application, the server, and the browser. With its speed, security, and near-universal availability, C may have been selected over interpreted languages. It is no longer usual practice to design websites with C, and there are several alternative web development tools available.

c) Some additional languages are written in C as well:

Because of C's widespread availability and efficiency, compilers, libraries, and interpreters for other programming languages are frequently written in C. Python, Perl, Ruby, and PHP, for example, have reference implementations written in C.

d) Allows use with computationally demanding libraries:

Although the layer of abstractions from hardware is thin and the overhead is minimal, C allows programmers to develop efficient implementation of algorithms and data structures, which is a key criteria for computationally intensive systems. The GNU Multi Precision Arithmetic Library, the GNU Scientific Library, mathematically, and MATLAB, for example, are written entirely or partially in C. Many languages enable invoking C library functions; for example, the Python-based architecture NumPy makes use of C for high-performance and hardware interaction.

e) C is used as an intermediate language:

C sometimes can be utilized as an intermediate language by other language implementations. This method can be employed for scalability or convenience; utilizing C as an intermediary language eliminates the need for machine-specific code generators. C has certain characteristics that aid with the compilation of produced code, such as line-number preprocessor directives and optional unnecessary commas at the end of initializer lists. However, some of C's inadequacies have encouraged the development of additional C-based languages, such as C--, that are expressly designed for usage as intermediate languages. Furthermore, the modern main compilers GCC and LLVM both provide an intermediate format that is not C, and both allow front ends for numerous languages, including C.

f) End-user programs:

C is also commonly used to create end-user apps. Such programs, however, can also be created in more recent higher-level languages.

2.3 Programming Format of 'C'

In C, the format specifier informs the compiler about the kind of data to be written or scanned during input and output operations. They usually begin with a % symbol and are utilized in formatted strings in functions such as printf(), scanf(), sprintf(), and so on. The C programming language has a number of format specifiers associated with various data types, like %d for int, %c for char, and so on. This article will go over some of the most often used formatting specifiers and how to utilize them.

The table below lists the most widely used format specifiers in C:

Format Specifier	Description
%c	For character type
%d	For signed integer type
%e	For scientific notation of floats
%f	For floats type
%g	For float type with the current precision
%i	Unsigned integer
%Id or %li	Long
%If	Double

%Lf	Long double
%lu	Unsigned int or Unsigned long
%lli	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x	Hexadecimal representation
%n	Print nothing
%%	Print % character

Basic format and functions

1. **Character Format Specifier – %c in C:**

   In C, the format specifier for the char data type is %c. It may be used in C for both formatted input and formatted output.

   Syntax:

   scanf("%d...", ...);

   printf("%d...", ...);

   Example:

```
// C Program to illustrate the %c format specifier.
#include <stdio.h>
int main()
{
        char c;
        // using %c for character input
        scanf("Enter some character: %c", &c);
// using %c for character output
        printf("The entered character: %c", &c);
        return 0;
}
```

   Input:

   Enter some character: A

   Output:

   The entered character: A

2. **Integer Format Specifier (signed) – %d in C:**

The signed integer format specifier%d can be used in the scanf() and print() methods, as well as other functions that employ formatted text for int data type input and output.

Syntax:

```
scanf("%d...", ...);
printf("%i...", ...);
```

Example:

```
// C Program to demonstrate the use of %d and %i
#include <stdio.h>
// Driver code
int main()
{
        int x;
        // taking integer input
        scanf("Enter the two integers: %d", &x);
        // printing integer output
        printf("Printed using %%d: %d\n", x);
        printf("Printed using %%i: %3i\n", x);
        return 0;
}
```

Input:

Enter the integer: 45

Output:

Printed using %d: 45

Printed using %i: 45

3. **Unsigned Integer Format Specifier – %u in C:**

The format specifier for the unsigned integer data type is %u. When we provide the %u a negative integer value, it transforms it to its first complement.

Syntax:

```
printf("%u...", ...);
scanf("%u...", ...);
```

Example: C program below shows how to use %u in C.

```
// C Program to illustrate the how to use %u
#include <stdio.h>
// driver code
int main()
```

```
{
        unsigned int var;
        scanf("Enter an integer: %u", &var);
        printf("Entered Unsigned Integer: %u", var);
// trying to print negative value using %u
        printf("Printing -10 using %%u: %u\n", -10);
        return 0;
}
```

Input:

Enter an integer: 25

Output:

Entered unsigned integer: 25

Printing -10 using %u: 2494692768

4. **Floating-point format specifier – %f in C:**

The %f is a floating point format specifier in C that can be used inside a formatted string for float data input and output. In addition to %f, we may use the format specifiers %e or %E to display the floating point value in exponential form.

Syntax:

```
printf("%f...", ...);
scanf("%e...", ...);
printf("%E...", ...);
```

Example:

```
// C program to demonstrate the use of %f, %e and %E
 #include <stdio.h>
// driver code
 int main()
{
        float a = 12.67;
        printf("Using %%f: %f\n", a);
        printf("Using %%e: %e\n", a);
        printf("Using %%E, %E", a);
        return 0;
}
```

Output:

```
Using %f: 12.670000
```

Using %e: 1.267000e+01

Using %E, 1.267000E+01

5. **Unsigned Octal number for integer – %o in C:**

In the C program, we may use the%o format specifier to print or accept input for the unsigned octal integer number.

Syntax:

```
printf("%o...", ...);
scanf("%o...", ...);
```

Example:

```
#include <stdio.h>
int main()
{
int a = 67;
printf("%o\n", a);
return 0;
}
```

Output

103

6. **Unsigned Hexadecimal for integer – %x in C:**

For hexadecimal integers, the format specifier %x is used in the prepared text. The alphabets in the hexadecimal numerals will be in lowercase in this situation. Instead of %X, we use %X for uppercase alphabet digits.

Syntax:

```
printf("%x...", ...);
scanf("%X...", ...);
```

Example:

```
// C Program to demonstrate the use of %x and %X
#include <stdio.h>
int main()
{
    int a = 15454;
    printf("%x\n", a);
    printf("%X", a);
    return 0;
}
```

Output

3c5e

3C5E

7. **String Format Specifier – %s in C:**

In C, the %s symbol is used to print strings or to accept strings as input.

Syntax:

printf("%s...", ...);

scanf("%s...", ...);

Example:

```
// C program to illustrate the use of %s in C
#include <stdio.h>
int main()
{
char a[] = "Hi Raghav";
printf("%s\n", a);
return 0;
}
```

Example:

The operation of %s with scanf() differs slightly from that of printf(). Let's look at this with the aid of the C program below.

```
// C Program to illustrate the working of %s with scanf()
#include <stdio.h>
int main()
{
char str[50];
// taking string as input
scanf("Enter the String: %s", str);
printf("Entered String: %s", str);
return 0;
 }
```

Example:

Input

Enter the string: Hi Raghav

Output

Hi

As we can see, the string is only searched until it encounters whitespace. In C, we may avoid this by using scan sets.

8. Address Format Specifier – %p in C:

The C programming language also has a format specifier for printing addresses/pointers. In C, we may use %p to display addresses and pointers.

Syntax

printf("%p...", ...);

Example:

```
#include <stdio.h>
int main()
{
 int a = 10;
 printf("The Memory Address of a: %p\n",(void*)&a);
 return 0;
 }
```

Output

The Memory Address of a: 0x7ffe9645b3fc

Input and Output Formatting:

The C programming language has certain facilities for formatting input and output. They are often placed between the % sign and the format specifier symbol. Here are a few examples:

A negative (-) symbol indicates left alignment.

A number after % defines the minimum field width to be printed; if the characters are fewer than the width, the leftover space is filled with space; if it is more, it is displayed as is without truncation.

A period (.) sign denotes the separation between field width and accuracy.

Precision specifies the number of digits in an integer, the number of characters in a string, and the number of digits after the decimal point in a floating value.

Example of I/O Formatting:

```
// C Program to demonstrate the formatting methods.
#include <stdio.h>
int main()
        {
char str[] = "beginner";
printf("%20s\n", str);
printf("%-20s\n", str);
```

```
printf("%20.5s\n", str);
printf("%-20.5s\n", str);
return 0;
}
Output:
beginner
beginner
begin
begin
```

A simple C programming

C Program To Print Your Own Name:

In this case, we have two approaches for printing the name:

- Using printf()
- Using scanf()

Input:

Enter Name: Hello

Output:

Name = Hello

Example 1:

In this example, we use the printf() function to print the user name.

```
// C program to demonstrate printing of
// our own name using printf()
#include <stdio.h>
int main()
{
// print name
   printf("Name : Raghav");
   return 0;
 }
Output:
Name = Raghav
```

Example 2:

In this example, we use scanf() to accept the user's name and then print it.

```
// C program to demonstrate printing of
```

```
// our own name using scanf()
#include <stdio.h>
int main()
{
char name[20];
printf("Enter name: ");
// user input will be taken here
scanf("%s", name);
printf("Your name is %s.", name);
return 0;
}
```

Output:

Enter Name: Raghav

Name = Raghav

2.4 Creating a C Program

A C program can be executed on platforms such as DOS, UNIX etc. DOS stores C program with a file **extension .c**. Program text can be entered using any text editor such as EDIT or any other. To edit a file called **testprog.c** using edit editor, gives:

C:> edit testprog.c

If you are using Turbo C, then Turbo C provides its own editor which can be used for writing the program. Just give the full pathname of the executable file of Turbo C and you will get the editor in front of you. For example:

C:> turboc\bin\tc

Here, tc.exe is stored in bin subdirectory of turboc directory. After you get the menu just type the program and store it in a file using the menu provided. The file automatically gets the extension of. c.

UNIX also stores C program in a file with extension is. c. This identifies it as a C program. The easiest way to enter your text is using a text editor like vi, emacs or xedit. To edit a file called testprog.c using vi type

$ vi testprog.c

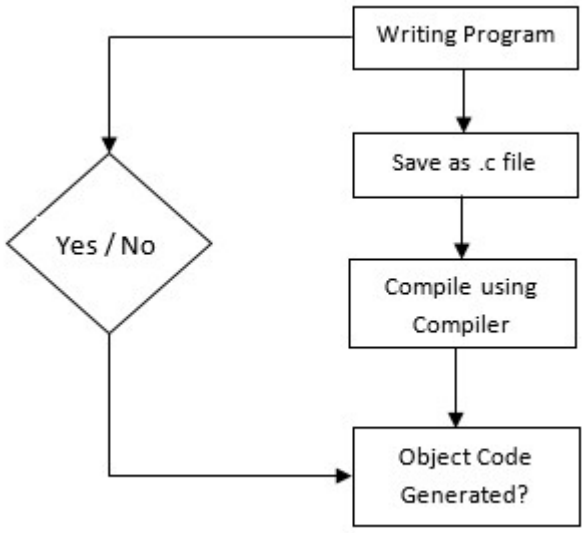
The editor is also used to make subsequent changes to the program.

2.5 Compilation process in C Program

After you've finished writing the program, save it in a file with the extension.c. This software is written in a high-level language. However, the computer does not understand this language. The following step is to translate the high-level language program (source code) to machine language (object code). A compiler is a piece of software or a program that handles this duty. Every programming language has its own compiler, which turns source code to object code. If the program is syntactically accurate, the compiler

will successfully compile it; otherwise, the object code will not be created. Figure shows an illustration of this.

This flowchart illustrating the process of writing a program, saving it, and then compiling it using a compiler. Here's a basic representation:



This flowchart represents the sequential steps of writing a program, saving it with the ".c" extension, using a compiler to compile the code into object code, and then checking whether the object code was successfully generated or not.

The Compiler

If you are working on UNIX platform, then if the name of the program file is

testprog.c, to compile it, the simplest method is to type

cc testprog.c

This will compile testprog.c, and, if successful, will produce a executable file called

a.out. If you want to give the executable file any other, you can type

cc testprog.c -o testprog

This will compile testprog.c, creating an executable file testprog.

If you are working with TurboC on DOS platform then the option for compilation is provided on the menu. If the program is syntactically correct then this will produce a file named as testprog.obj. If not, then the syntax errors will be displayed on the screen and the object file will not be produced. The errors need to be removed before compiling the program again. This process of removing the errors from the program is called as the debugging.

Semantic and Syntax Errors

Every language has an associated grammar, and the program written in that language has to follow the rules of that grammar. For example in English a sentence such a “Raghav, is playing, with a ball”. This sentence is syntactically incorrect because commas should not come the way they are in the sentence.

Likewise, C also follows certain syntax rules. When a C program is compiled, the compiler will check that the program is syntactically correct. If there are any syntax errors in the program, those will be displayed on the screen with the corresponding line numbers.

Let us consider the following program.

Example: Write a program to print a message on the screen.

```
/* Program to print a message on the screen*/ #include <stdio.h

main( )

{

printf("Hello, how are you\n")
```

Let the name of the program be test.c .If we compile the above program as it is we will get the following errors:

Error test.c 1:No file name ending Error test.c 5: Statement missing ;

Error test.c 6: Compound statement missing }

Edit the program again, correct the errors mentioned and the corrected version appears as follows:

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

{

printf ("Hello, how are you\n");
```

Apart from syntax mistakes, semantic errors are another form of error that appears during compilation. These mistakes are shown as warnings. These mistakes happen when a statement has no meaning. Although the program compiles with these mistakes, it is usually recommended that they be corrected as well, as they may cause difficulties during operation. As an example, suppose you declared a variable but did not utilize it, and you receive the warning "code has no effect." These variables are taking up unnecessary memory space.

2.6 Link and Running C Program

After compilation, the next step is linking the program. Compilation produces a file with an extension .obj. Now this .obj file cannot be executed since it contains calls to functions defined in the standard library (header files) of C language. These functions have to be linked with the code you wrote. C comes with a standard library that provides functions that perform most commonly needed tasks. When you call a function that is not the part of the program you wrote, C remembers its name. Later the linker combines the code you wrote with the object code already found in the standard library. This process is called linking. In other words, Linker is a program that links separately compiled functions together into one program. It combines the functions in the standard C library with the code that you wrote. The output of the linker in an executable program i.e., a file with an extension .exe.

Running Program through Menu

When we are working with TurboC in DOS environment, the menu in the GUI that pops up when we execute the executable file of TurboC contains several options for executing the program:

i. Link, after compiling
ii. Make, compiles as well as links
iii. Run

All these options create an executable file and when these options are used we also get the output on user screen. To see the output we have to shift to user screen window.

Running Executable File

An .exe file produced by can be directly executed.

UNIX also includes a very useful program called make. Make allows very complicated programs to be compiled quickly, by reference to a configuration file (usually called **makefile**). If your C program is a single file, you can usually use make by simply typing –

make testprog

This will compile **testprog.c** as well as link your program with the standard library so that you can use the standard library functions such as **printf** and put the executable code in **testprog**.

In case of DOS environment, the options provided above produce an executable file and this file can be directly executed from the DOS prompt just by typing its name without the extension. That is if the name of the program is **test.c**, after compiling and linking the new file produced is test.exe only if compilation and linking is successful.

This can be executed as: **c>test**

Linker Errors

If a program contains syntax errors then the program does not compile, but it may happen that the program compiles successfully but we are unable to get the executable file, this happens when there are certain linker errors in the program. For example, the object code of certain standard library function is not present in the standard C library; the definition for this function is present in the header file that is why we do not get a compiler error. Such kinds of errors are called linker errors. The executable file would be created successfully only if these linker errors are corrected.

Logical and Runtime Errors

After the program is compiled and linked successfully we execute the program. Now there are three possibilities:

1. The program executes and we get correct results,
2. The program executes and we get wrong results, and
3. The program does not execute completely and aborts in between.

The first case simply means that the program is correct. In the second case, we get wrong results; it means that there is some logical mistake in our program. This kind of error is known as logical error. This error is the most difficult to correct. This error is corrected by debugging. Debugging is the process of removing the errors from the program. This means manually checking the program step by step and verifying the results at each step. Debugging can be made easier by a tracer provided in Turbo C environment. Suppose we have to find the average of three numbers and we write the following code:

Example: Create a C program to compute the average of three integers.

```
/* Program to compute average of three numbers
* #include<stdio.h>
main( )
{
int a,b,c,sum,avg;
```

```
a=10; b=5; c=20;
sum = a+b+c; avg = sum / 3;
printf("The average is %d\n", avg);
}
```

OUTPUT:

The average is 8.

The exact value of average is 8.33 and the output we got is 8. So we are not getting the actual result, but a rounded off result. This is due to the logical error. We have declared variable avg as an integer but the average calculated is a real number, therefore only the integer part is stored in avg. Such kinds of errors which are not detected by the compiler or the linker are known as logical errors.

The third kind of error is only detected during execution. Such errors are known as run time errors. These errors do not produce the result at all, the program execution stops in between and the run time error message is flashed on the screen. Let us look at the following example:

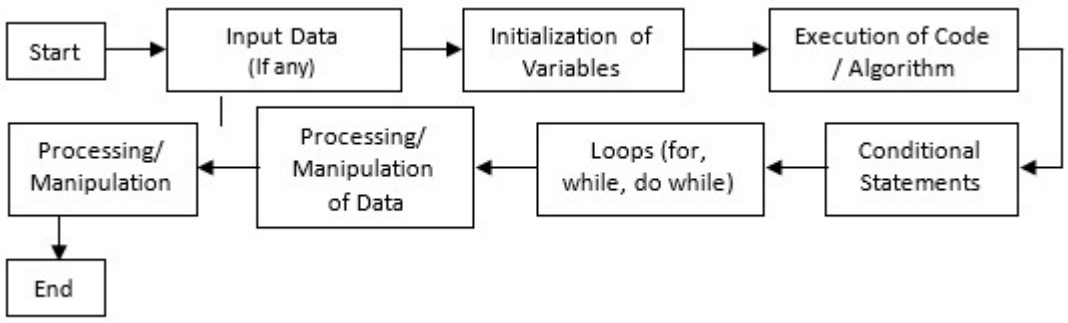
Example: Write a program to divide a sum of two numbers by their difference

```
/* Program to divide a sum of two numbers by their difference*/
#include <stdio.h>
main( )
{
int a,b; float c;
a=10; b=10;
c = (a+b) / (a-b);
printf("The value of the result is %f\n",c);
}
```

The above program will compile and link successfully, it will execute till the first printf statement and we will get the message in this statement, as soon as the next statement is executed we get a runtime error of "Divide by zero" and the program halts. Such kinds of errors are runtime errors.

2.7 Diagrammatic Illustration

The diagrammatic depiction of the program execution process is shown in the image below.



A flowchart representing the execution process of a C program is generally composed of a series of stages. Here's a simple illustration:

In end this flowchart demonstrates the general flow of a C program. It starts with data input (if necessary), followed by initializing variables, executing the main code or algorithm, applying conditional statements or loops as needed, calling functions (if used), processing data, displaying results, and finally, ending the program. This flow can vary depending on the specific logic and structure of the C program being executed.

2.8 Conclusion

You learnt about a program and a programming language in this unit. Languages can now be classified as high-level or low-level. You may now define C and its characteristics. You have investigated C's emergence. You've seen how C differs from other High-Level languages by being a middle-level language. The benefits of using high-level language over low-level language are highlighted.

You've seen how to turn an algorithm and a flowchart into a C program. In both the UNIX and DOS environments, we addressed the process of creating and saving a C program in a file.

You've learned how to compile and run a C program in both UNIX and DOS environments. We also spoke about the many sorts of problems that occur along the process, including as syntax errors, semantic errors, logical errors, linker errors, and runtime issues. You've also learned how to fix these mistakes. Simple C programs employing simple arithmetic operators and the printf() function are now possible.

Now that we've covered the fundamentals, we're ready to dive into the C language in depth in the next units.

2.9 Questions

1. **Write down the features of C Language?**

Ans.1 Features of C Language:

- **Procedural Language:** C follows a procedural programming paradigm, where programs are structured as sequences of instructions to be executed step by step.
- **Middle-level Language:** C combines low-level features (such as direct memory manipulation) with high-level constructs (such as functions and control structures), making it suitable for system programming as well as application development.
- **Portable:** C programs written for one platform can be easily ported to another with minimal changes, thanks to its standardized syntax and libraries.
- **Efficient:** C provides direct access to memory and hardware, allowing for efficient system programming and optimization of performance-critical tasks.
- **Structured Programming:** C supports structured programming constructs like functions, loops, and conditional statements, enabling modular and organized code development.
- **Rich Library Support:** C comes with a rich set of standard libraries (such as stdio.h, string.h, math.h), providing functions for common tasks like I/O operations, string manipulation, and mathematical calculations.
- **Pointer Support:** C allows manipulation of memory addresses through pointers, enabling advanced data structures and efficient memory management.

- **Extensible:** C supports modular programming through functions and libraries, facilitating code reuse and maintainability.
- **Wide Adoption:** C has been widely used for decades in various domains, including operating systems, embedded systems, game development, and scientific computing.

2. List and explain the different types of format specifies?

Ans.2 Format specifiers in C are used in functions like printf() and scanf() to specify the type and format of data to be input or output. Here are some common format specifiers:

- **%d:** Integers (int)
- **%f:** Floating-point numbers (float, double)
- **%c:** Characters (char)
- **%s:** Strings (char array or pointer)
- **%p:** Pointers
- **%ld:** Long integers (long)
- **%lu:** Unsigned long integers (unsigned long)
- **%lf:** Double-precision floating-point numbers (double)
- **%x:** Hexadecimal integers

3. Write down the structure of the C programme?

Ans.3
```
// Preprocessor Directives
#include <stdio.h> // Include standard input-output library
// Function Prototype (optional)
// int main();
// Global Variables Declaration (optional)
// Function Definition
int main() {
   // Local Variable Declaration
   // Statements
   printf("Hello, World!\n"); // Output statement
   // Return Statement
   return 0;
}
```

4. Why global declaration is different from local declaration?

Ans.4 **Global Declaration:** Variables declared outside of any function are global variables. They are accessible throughout the entire program and have a global scope. Global variables are initialized only once, and their memory is allocated when the program starts. They persist throughout the program's execution.

Local Declaration: Variables declared within a function are local variables. They have local scope and are only accessible within the function where they are declared. Local variables are initialized every time the function is called, and their memory is allocated on the stack. They cease to exist once the function execution is complete.

5. Write a basic C programme to add two numbers?

Ans.5
```
#include <stdio.h>
int main() {
```

```
    int num1, num2, sum;
    // Input
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    // Calculation
    sum = num1 + num2;
    // Output
    printf("Sum: %d\n", sum);
    return 0;
}
```

2.10 References

- Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
- Griffiths, D., & Griffiths, D. (2012). Head First C: A Brain-Friendly Guide. O'Reilly Media.
- King, K. N. (2008). C Programming: A Modern Approach (2nd ed.). W. W. Norton & Company.
- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd ed.). Addison-Wesley.

***