**PROGRAMMING IN C**

**UNIT-1**

**Introduction: What is C**

* C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.
* C was invented to write an operating system called UNIX.
* C is a successor of B language which was introduced around 1970.
* Today C is the most widely used System Programming Language.
* C has now become a widely used professional language for various reasons.
	+ Easy to learn
	+ Structured language
	+ It produces efficient programs.
	+ It can handle low-level activities.
	+ It can be compiled on a variety of computers.

**Why Learn C?**

* It is one of the most popular programming languages in the world
* If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar
* C is very fast, compared to other programming languages, like Java and Python
* C is very versatile; it can be used in both applications and technologies

**Getting Started with C:**

Communicating with a computer involves speaking the language the computer understands, which immediately rules out English as the language of communication with computer. However, there is a close analogy between learning English language and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instructions would be combined later on to form a program.

**Steps in learning C**

Program

Instructions

Constants Variables Keywords

Alphabets

Digits

Special symbols

To start using C, you need two things:

* A text editor, like Notepad, to write C code
* A compiler, like GCC, to translate the C code into a language that the computer will understand

**Constants, Variables and Keywords**

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords. Let us see what are ‘constants’ and ‘variables’ in C. A constant is an entity that doesn’t change whereas a variable is an entity that may change.

**Types of C Constants:**

C constants can be divided into two major categories:

**(a) Primary Constants-(**Integer Constant, Real Constant, Character Constant)

**(b) Secondary Constants-(**Array, Pointer, Structure, Union, Enum,etc**)**

**Rules for Constructing Integer Constants**

* An integer constant must have at least one digit.
* It must not have a decimal point.
* It can be either positive or negative.
* If no sign precedes an integer constant it is assumed to be positive.
* No commas or blanks are allowed within an integer constant.
* The allowable range for integer constants is -32768 to 32767.

**Rules for Constructing Real Constants**

Real constants are often called **Floating Point constants.** The real constants could be written in two forms—**Fractional form and Exponential form.**

Following rules must be observed while constructing real constants expressed in fractional form:

* A real constant must have at least one digit.
* It must have a decimal point.
* It could be either positive or negative.
* Default sign is positive.
* No commas or blanks are allowed within a real constant.

**Ex.: +325.34 426.0 -32.76 -48.5792**

The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large. It however doesn’t restrict us in any way from using exponential form of representation for other real constants.

In exponential form of representation, the real constant is represented in two parts. The part following **‘e’ is called exponent.**

**Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.**

**Rules for Constructing Character Constants**

* A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example, ’A’ is a valid character constant whereas ‘A’ is not.
* The maximum length of a character constant can be 1 character.

**Ex.: 'A' 'I'**

**The First C Program:**

we begin with our first C program do remember the following rules that are applicable to all C programs:

* Each instruction in a C program is written as a separate statement. Therefore a complete C program would comprise of a series of statements.
* The statements in a program must appear in the same order in which we wish them to be executed; unless of course the logic of the problem demands a deliberate ‘jump’ or transfer of control to a statement, which is out of sequence.
* Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.
* All statements are entered in small case letters.
* C has no specific rules for the position at which a statement is to be written. That’s why it is often called a free-form language.
* Every C statement must end with a **;**. Thus **;** acts as a statement terminator.

**#include <stdio.h>**

**int main() {
  printf("Hello World!");
  return 0;
}**

**Line 1:** #include <stdio.h> is a header file library that lets us work with input and output functions, such as printf() (used in line 4). Header files add functionality to C programs.

**Line 2:** A blank line. C ignores white space. But we use it to make the code more readable.

**Line 3:** Another thing that always appear in a C program, is main(). This is called a **function**. Any code inside its curly brackets {} will be executed.

**Line 4:** printf() is a **function** used to output/print text to the screen. In our example it will output "Hello World".

**Line 5:** return 0 ends the main() function.

**Line 6:** Do not forget to add the closing curly bracket } to actually end the main function.

**Compilation and Execution:**

## **What is a compilation?**

* The compilation is a process of converting the source code into object code. It is done with the help of the compiler.
* The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



* The c compilation process converts the source code taken as input into the object code or machine code.
* The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.



Once you have written the program you need to type it and instruct the machine to execute it. To type your C program you need another program called Editor. Once the program has been typed it needs to be converted to machine language (0s and 1s) before the machine can execute it. To carry out this conversion we need another program called Compiler. Compiler vendors provide an Integrated Development Environment (IDE) which consists of an Editor as well as the Compiler.

There are several such IDEs available in the market targeted towards different operating systems. For example, Turbo C, Turbo C++ and Microsoft C are some of the popular compilers that work under MS-DOS; Visual C++ and Borland C++ are the compilers that work under Windows, whereas gcc compiler works under Linux. Note that Turbo C++, Microsoft C++ and Borland C++ software also contain a C compiler bundled with them. If you are a beginner you would be better off using a simple compiler like Turbo C or Turbo C++. Once you have mastered the language elements you can then switch over to more sophisticated compilers like Visual C++ under Windows or gcc under Linux. Most of the programs in this book would work with all the compilers. Wherever there is a deviation I would point it out that time.

Assuming that you are using a Turbo C or Turbo C++ compiler here are the steps that you need to follow to compile and execute your first C program…

* Start the compiler at **C>** prompt. The compiler (TC.EXE is usually present in **C:\TC\BIN** directory).
* Select **New** from the **File** menu.
* Type the program.
* Save the program using **F2** under a proper name (say Program1.c).
* Use **Ctrl + F9** to compile and execute the program.
* Use **Alt + F5** to view the output.

**Receiving Input:**

* **Get Integer Input**
* **Get Character Input**
* **Get**[**String**](https://codescracker.com/c/c-strings.htm)**Input**

## **1. Get Integer Input**

* To receive or get an integer input from the user in C programming, use the [function](https://codescracker.com/c/c-functions.htm) **scanf()**.
* This function takes two argument. First one is the format specifier of **input type.** And second parameter is the address of  [variable](https://codescracker.com/c/c-variables.htm) related to input data.

**Let's take a look at the program given below:**

#include<stdio.h>

#include<conio.h>

int main()

{

 int num;

 printf("Enter the Number: ");

 scanf("%d", &num);

 printf("\nYou've entered: %d", num);

 getch();

 return 0;

}

This program was build and run under **Code::Blocks** IDE. Here is its sample run:



Now supply any integer value say **10** and press ENTER key to see the following output:



**Note -**The **%d** format specifier is used to scan integer type value.

As you can see from the above program, the function **scanf()** has two arguments. The first argument specifies the format specifier of input data. And the second argument is the variable to which the value is going to be saved.

**Note** - Don't forgot to put **address of (&)** operator before the variable say **num** inside the function **scanf()** to scan any integer value.

## **2. Get Character Input**

Now let's create another program that receives a character type input data from user at run-time:

#include<stdio.h>

#include<conio.h>

int main()

{

 char ch;

 printf("Enter the Character: ");

 scanf("%c", &ch);

 printf("\nYou've entered: %c", ch);

 getch();

 return 0;

}

**Here is its sample run:**



The **%c** format specifier is used for character input/output.

### What if User enters more than One Character

If user enters two or more characters as input, then above program scans and initialized the first character only to **ch** variable. Rest of the characters gets skipped. Therefore, if user enters *programming* as input, then above program prints only *c* as output.

## **3. Get String Input**

## This program scans the string input from user. That is more than one character input from user.

#include<stdio.h>

#include<conio.h>

int main()

{

 char str[100];

 printf("Enter any String: ");

 gets(str);

 printf("\nYou've entered: %s", str);

 getch();

 return 0;

}

The **%s** format specifier is used for string input/output.

You can also use the **scanf()** function with **%s** format specifier to scan the string from user. But it is always recommended to scan any string using **gets()** function. Because, if you'll use *scanf()* function to scan the string, then it skips all the things after first space. That is, *scanf()* reads the string until a spaces gets occurred. But *gets()* reads complete string along with spaces.

**C Instructions:**

There are basically three types of C instructions.

**1. Type Declaration Instruction
2. Arithmetic Instruction
3. Control Instruction**

## **1. Type Declaration Instruction**

As its name suggests it is used to declare type of variables in C language. It is must that you have to declare the type of variable before using it. And it is also must that these type declaration instruction should be provided in the beginning of the program (just after the main()).

Now lets learn about some variations in them.

a) We can initialize the variable at the time of its declaration.

**Example**
int a=3;
char a=’d’;
int a=7\*3\*2;

b) Remember the order of variables while declaring them.

**Example**
int i=3,j=5; is same as int j=5,i=3;

However,
float a=5.5,b=a+7.1; is alright, but
float b=a+7.1,a=5.5; is not valid.

It is because in the above declaration we are trying to use variable a even before defining it.

c) Another interesting point while declaring variables is

Below instruction will work
int a,b,c,d;
a=b=c=10;

However, the following statement would not work.
int a=b=c=d=10 ;

Now again we are trying to use variables even before defining them.

I hope now you must have understand the importance of type declaration instruction. So we can only use variables after defining them. And these definition should be written at the starting of main() body. We cannot define variables anywhere else in the program. If you do so, it will result in an error.

## **2. Arithmetic Instruction**

General form of an arithmetic instruction follow rules given below.

i) It should contain one assignment operator i.e. =.
ii) On the left side of =, there should be one variable.
iii) On the right side of =, there should be variables and constants.
iv) And those variables and constant will be connected by some arithmetic operators like +,-,\*,/,%.

**Example**
int a=12;
float b,c=2.2;
b=c+a/6.1\*8;

**What are operands?**

These variables and constants together are called operands. These operands are connected by arithmetic operators.

**How the execution of arithmetic instructions takes place?**

Firstly all the variables and constants on the right hand side of assignment operator (=) is calculated. After that the result will be stored in the variable on the left hand side of assignment operator.

Now let’s checkout certain nuances of arithmetic operators

1. It is compulsory to use only one variable on the left hand side of =.

**Example**
c=a\*b is correct,
whereas a\*b=c is incorrect.

2. A operator named modular operator (%) is also used in C. This modular operator will return the remainder to left hand side variable of assignment operator. It cannot be used with floats. It will return the same sign as the numerator has.

**Example**
-5%2=-1
5 %-2=1

## **3. Control Instruction**

This instruction is used to shift the control of program as per the user or programmer wants. This instruction contains decision making, looping and more.

**The Decision Control Structure:**

In any programming language, there is a need to perform different tasks based on the condition. Using decision control statements, we can control the flow of program in such a way so that it executes certain statements based on the outcome of a condition (i.e. true or false). In C Programming language we have following decision control statements.

* + **if statement**
	+ **if-else Statement**
	+ **Nested if statement or if-else statement**
	+ **switch-case statements**

**1. The *if* Statement:**

* It is one of the simplest forms of decision control statements which is frequently used in decision making.
* An if statement consists of a Boolean expression followed by one or more statements.
* The statements inside if body executes only when the condition defined by if statement is true. If the condition is false then compiler skips the statement enclosed in if’s body. We can have any number of if statements in a C program.

**Syntax:**

**if (test expression)
{
    statement 1;
    …...............
    statement n;
}
statement x;**

## **Flow Diagram:**



* The if structure may have one statement or n number of statements which are enclosed within the curly braces ({ }).
* Initially the test expression is evaluated.
* If this expression is true the statement of the if block is executed or all the statements will be skipped and statement x will be executed.

#### **Example:**

**Write a program to print the highest number.**#include <stdio.h>
void main()
{
    int n;
    printf("Enter a number:");
    scanf("%d", &n);
    printf ("The entered number %d", n);
    if (n>100)
        printf ("You entered a higher number");
}

In the above program, a number is accepted as an input from the user and the output is given. The expression (n>100) is given. If the user gives a number higher than 100 then only will the message be printed else the statement will be skipped.

**2. if-else statement**

## If the Boolean expression evaluates to **true**, then the **if block** will be executed, otherwise, the **else block** will be executed.

**Syntax:**

**if (expression)
    Statement 1;
else
    Statement 2;**

## **Flow Diagram**



#### **Example:**

#include <stdio.h>

int main()

{

 int age;

 printf("Enter your age:");

 scanf("%d",&age);

 if(age >=18)

 {

 printf("You are eligible for voting");

 }

 else

 {

 printf("You are not eligible for voting");

 }

 return 0;

}

In this program user is asked to enter the age and based on the input, the if..else statement checks whether the entered age is greater than or equal to 18. If this condition meet then display message “You are eligible for voting”, however if the condition doesn’t meet then display a different message “You are not eligible for voting”.

If there is **only one statement** is present in the “if” or “else” body then you do not need to use the braces (parenthesis). the above program can be rewritten like this:

**#include <stdio.h>**

**int main()**

**{**

 **int age;**

 **printf("Enter your age:");**

 **scanf("%d",&age);**

 **if(age >=18)**

 **printf("You are eligible for voting");**

 **else**

 **printf("You are not eligible for voting");**

 **return 0;**

**}**

**Uses of Logical Operators:**

* Logical operators perform logical operations on a given expression by joining two or more expressions or conditions.
* It can be used in various relational and conditional expressions.
* This operator is based on Boolean values to logically check the condition, and if the conditions are true, it returns 1. Otherwise, it returns 0 (False).
* In C programming, logical operators are classified into three types such as the logical AND (&&) operator, the logical OR operator (||), and the logical NOT (!) operator.



**There are three types of logical operators:**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| && | AND operator |
| || | OR operator |
| ! | NOT operator |

* The AND ( && ) and OR ( || ) are **binary operator** while NOT ( ! ) is a **unary operator**.

**1. Logical AND Operator:**

* The logical AND operator is represented as the '&&' double ampersand symbol.
* It checks the condition of two or more operands by combining in an expression, and if all the conditions are true, the logical AND operator returns the Boolean value true or 1. Else it returns false or 0.

**Syntax**

**(condition1 && condition2)**

There are two conditions in the above syntax, condition1 and condition2, and in between the double (&&) ampersand symbol. If both the conditions are true, the logical AND operator returns Boolean value 1 or true. Otherwise, it returns false.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A&&B** |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

**2. Logical OR operator:**

* The **‘||’** operator returns true even if one (or both) of the conditions under consideration is satisfied. Otherwise, it returns false.
* If one of the operands or expressions is true, it will return 1.
* If all of them are false, it will return 0.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A||B** |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**3. Logical NOT operator:**

* The ‘!’ operator returns true the condition in consideration is not satisfied. Otherwise, it returns false.
* The ! (not) operator negates the value of the condition. If the condition is false then it becomes true, if it is true then it becomes false.
* Unlike && (AND) and || (OR) operator, the ! (NOT) operator is unary.

|  |  |
| --- | --- |
| **Operands** | **Result** |
| 1 | 0 |
| 0 | 1 |

**Let's take an example:**

 int a = 10, b = 9;

**Suppose we have the following logical expression.**

 **!(a > 5)**

As we can see, the condition a > 5 is true. And !(NOT) operator negates the value of the condition, so the result of the overall expression is false i.e 0.

**Program:**

|  |
| --- |
| #include <stdio.h> int main(){    int a = 10, b = 4, c = 10, d = 20; **// logical operators**     **// logical AND example****if (a > b && c == d)**        printf("a is greater than b AND c is equal to d\n");    else        printf("AND condition not satisfied\n");  **// logical OR example****if (a > b || c == d)**        printf("a is greater than b OR c is equal to d\n");    else        printf("Neither a is greater than b nor c is equal "               " to d\n");   **// logical NOT example****if (!a)**        printf("a is zero\n");    else        printf("a is not zero");     return 0;} |

**Output:**

AND condition not satisfied

a is greater than b OR c is equal to d

a is not zero

**The Conditional Operators:**

* The conditional operator is also known as a **ternary operator**.
* The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., **'?' and ':'.**
* As conditional operator works on three operands, so it is also known as the ternary operator.
* The behavior of the conditional operator is similar to the 'if-else’ statement as 'if-else' statement is also a decision-making statement.

### Syntax of a conditional operator

**Expression1? expression2: expression3;**

**The pictorial representation of the above syntax is shown below:**



**Meaning of the above syntax.**

* In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
* If the expression1 results into a true value, then the expression2 will execute.
* The expression2 is said to be true only when it returns a non-zero value.
* If the expression1 returns false value then the expression3 will execute.
* The expression3 is said to be false only when it returns zero value.



**Example:**

**#include <stdio.h>**

**int main()**

{

 **int age;** // variable declaration

 **printf("Enter your age");**

 scanf("%d",&age); // taking user input for age variable

 **(age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));** // conditional operator

 return 0;

}

In the above code, we are taking input as the 'age' of the user. After taking input, we have applied the condition by using a conditional operator. In this condition, we are checking the age of the user. If the age of the user is greater than or equal to 18, then the statement1 will execute, i.e., (printf("eligible for voting")) otherwise, statement2 will execute, i.e., (printf("not eligible for voting")).