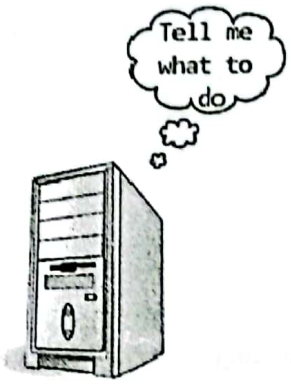
Introduction to Programming

What is Program?

A program is a set of instructions that tells the computer (Central Processing Unit) what to do. Since CPU can't perform any task at its own. Hence set of instructions are provided to CPU in the form of programs to perform specific tasks.



Source code:

Human readable format of a program is called its Source code. We can modify the source code any **number of times.**

Executable:

Binary format or machine readable format of any program is called as its Executable. They are the compiled format of source code and are generally in binary format. We cannot modify the executable of any program.

hello.c

```
#include <stdio.h>

int main()
{
  printf("Hello");
  return 0;
}
```

hello.exe

```
10101100010110101010
10001010011000000
00100000100011010
00101110100101110101
01111001010010011
11101111111100011
10101010111100001
01010101001001010
11111111000010101
```

Source code Executable

What is Programming?

Programming can be described as the process of writing an algorithm in a sequence of computer language instructions or we may simply say, the process of writing or editing any program. This may seems surprising that programming are in practice from the days when there were no concepts of computer, and the first programmer of the world was Ada Lovelace.

What is Programming Language?

Programming language is a language that provides a medium of communication between computer and human. They provide a way through which humans can interact with the computer with the help of programs. Programming language are often used for writing Programs. Some of popular programming languages are: C, C++, Java, C#, PHP, Javascript, Python, Ruby etc.

Categorization of programming language:

Low Level Language(LLL)

- ✓ Low level languages are programming languages that are directly understood by the computer or requires less interpretation. Or we may say programming language that provides little or no abstraction. Under this category falls Machine language and Assembly language.
- ✓ Low level languages are easily understood by computers but are hard to figure out by humans.
- ✓ Processing of any low level languages is much faster than High level languages.
- ✓ Examples:BCPL, ALGOL etc.

High Level Language(HLL)

- ✓ High level languages are programming languages that are easily understood by humans and are mostly written in English like statements. Or we may say programming languages with strong abstraction.
- ✓ High level languages requires translation into machine language or low level language hence requires compilers, interpreters to accomplish this translation job.
- ✓ Processing of any high level language takes more time compared to low level language because of time elapsed in converting high level instructions into low level instructions.
- ✓ Examples:C, C++, Java, C# etc.

Compiler:

Compilers are the software program that translates or converts the source code into specific machine code. Compilers also checks for syntax errors and ensures proper grammar of the source code. They may or may not convert the source code into an intermediate code called Object code. Full program is compiled at once and on any error compilation process fails.

Interpreter:

Interpreter are the software programs that directly executes instructions written in programming language line by line. They need not any prior compilation of the whole program instead they compile each line when they are about to execute. If any line is encountered with error, then the whole process is terminated and execution of program stops.

Tokens in programming

Tokens:

Smallest individual element of a program is called as Token. Everything you see inside a program is a token. There are generally five types of tokens:

- ✓ Keyword
- ✓ Identifier
- ✓ Operator
- ✓ Separator
- ✓ Literal

Keyword:

Keyword is a reserved word whose meaning is already defined by the programming language. We cannot use keyword for any other purpose inside programming. Every programming language have some set of keywords.

Examples: int, do, while, void, return etc(Note: These keywords are common to C and C influenced languages).

Identifier

Identifiers are the name given to different programming elements. Either name given to a variable or a function or any other programming element, all follow some basic naming conventions listed below:

Keywords must not be used as an identifier.

Identifier must begin with an alphabet(a-z A-Z) or an underscore(_) symbol.

Identifier can contains alphabets(a-z A-Z), digits(0-9) and underscore(_) symbol.

Identifier must not contain any special character(e.g. !@$*.'[] etc) except underscore(_).

Examples of some valid identifiers:

num, Num, _num, _Num, num1, Num1, _num1, _Num1, _1num, _1Num, _num_, number_to_add etc.

Examples of some invalid identifiers:

1num, number to add, 1_num, num-to-add, num@ etc.

Operator:

Operators are the symbol given to any arithmetical or logical operations. Various programming languages provides various sets of operators some common operators are:

Let's suppose two variables a=10, b=5

Arithmetic operator

Arithmetic operator are used to perform basic arithmetic operations.

Operator	Description	Example
+	Adds two operands.	a + b gives 15
-	Subtracts second operand from first.	a - b gives 5
*	Multiplies two operands.	a * b gives 50
/	Divides two operands.	a / b gives 2
%	Modulus operator divides the first operand from second and returns the remainder. It is generally used for checking divisibility.	a % b gives 0 (As 10/5 will have 0 remainder)

Assignment operator

Assignment operator is used to assign value to a variable. The value is assigned from right to left.

Operator	Description	Example
=	Assigns value from right operand to left operand.	a = 10 will assign 10 in a

Relational operator

Relational operator is used to check relation between any two operands. Whether any of them is greater, equal or not equal.

Operator	Description	Example
<	If value of left operand is greater than right, returns true else returns false	(a > b) will return true

>	If value of right operand is greater than left, returns true else returns false	(a < b) will return false
= =	If both operands are equal returns true else false	(a == b) will return false
!=	If both operands are not equal returns true else false.	(a != b) will return true
>=	If value of left operand is greater or equal to right operand, returns true else false	(a >= b) will return true
<=	If value of right operand is greater or equal to left operand, returns true else false	(a <= b) will return false

Logical operator

Logical operator are used to combine two boolean expression together and results in a single boolean value according to the operand and operator used.

Operator	Description	Example
&&	Used to combine two expressions. If both operands are true or Non-Zero, returns true else false	((a>=1) && (a<=10)) will return true since (a>=1) is true and also (a<=10) is true.
||	If any of the operand is true or Non-zero, returns true else false	((a>1) || (a<5)) will return true. As (a>1) is true. Since first operand is true hence there is no need to check for second operand.
!	Logical NOT operator is a unary operator. Returns the complement of the boolean value.	(!(a>1)) will return false. Since (a>1) is true hence its complement is false.

Increment/Decrement operator

Increment/Decrement operator is a unary operator used to increase an integer value by 1 or decrease it by 1. Increment/decrement operator are of two types Postfix and Prefix.

Operator	Description	Example
++	Increment operator will add 1 to an integer value.	a++ will give 11 ++a will also give 11
--	Decrement operator will subtract 1 from an integer value.	a-- will give 9 --a will also give 9

Separator

Separators are used to separate different programming elements. The various types of separators used in programming are:

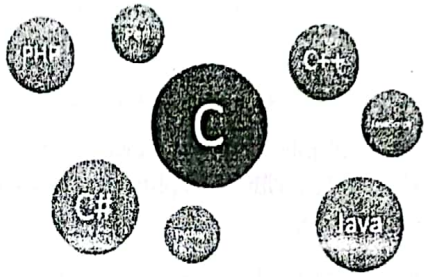
(Space) \t(Tab) \n(New line) . , ; () {} []

Introduction to C programming

C PROGRAMMING LANGUAGE

C is a procedural programming language developed by Dennis Ritchie in 1972 at AT&T Bell Labs. C is considered as the most powerful programming languages ever. It was originally developed to re-implement UNIX operating system. C is a High Level Programming language but often referred as Middle level programming language because of its ability to provide rich support to low level programming.

C is also called as mother of all modern programming languages as many of current programming languages such as C++, Java, C#, PHP, Python, Perl, JavaScript etc. are directly or indirectly influenced from C. It is the also the most popular programming language ever and is widely used for Operating Systems implementations and Embedded programming's.

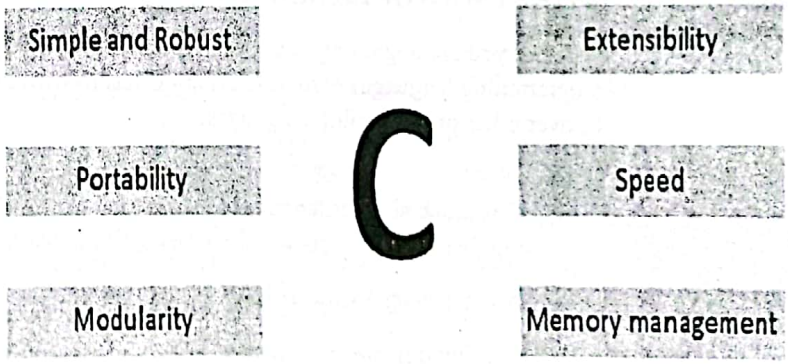


Languages originated from C

FEATURE OF C

C is powerful language and its real power comes from number of features it provides. Here are some features that C provides:

- Simple and Robust
- Portability
- Modularity
- Extensibility
- Speed
- Memory management



Features of C programming language

- **Simple and Robust**
  C is considered as simplest and easiest language ever. Its simplicity lies in the lesser number of programming constructs that are used to create any complex program. C is a robust programming language with rich set of in-built library functions and operators that can even write the most complex programs ever.

- **Portability**
  C is a machine independent programming language i.e. C program can run on any machine that has C compiler with few or no modification. Although C is doesn't provide platform independency as Java provides.

- **Modularity**
  C programs are modular in nature i.e. they are basically divided into modules by modules. **Modular programming** is a programming construct where we divide our program into different modules and combine them together in a single module to form a Program.

- **Extensibility**
  C provides a rich set of in-build library functions. But we can also write our custom library function and attach with the C programs to use them in future.

- **Speed**
  C programs compiles and executes faster than any other modern programming languages. And this is also the fact that it is used for Operating System development.

- **Memory management**
  C provides an efficient way to access and manage memory through the use of Pointers.

ADVANTAGES OF LEARNING C

Whether you are beginning your programming with C or you have learnt some other programming languages earlier. Learning C has its own importance, here are few advantages of C, over other programming languages.

- C is much simpler language as compared to any other programming language. It has less number of constructs which makes easier to learn.
- Almost every hardware have a C compiler.
- Compilation and execution of a C program is way faster than other modern programming languages.

- Being a high level programming language, its ability to support low level operations makes it different from other programming languages.
- C language is considered as a backbone of the **Operating System** development. As all popular OS are fully or partially written in C.
- C language is also widely used for developing **Compilers, Assemblers, Language Interpreters, Device drivers, Databases** etc.
- All modern programming languages are directly or indirectly inherited from C. Hence learning C will make easier to learn C influenced programming language.

WHAT ARE TOKENS

Tokens are the smallest individual elements of a program. Tokens are the basic building blocks of any program.
For example let's consider this English paragraph

"Programming in C is really very interesting! I am programming since I was 14. "

The above paragraph is made of several basic fundamentals things. Can you figure out all those fundamentals things?
Actually the paragraph contains
13 words, 2 digits, 13 blank spaces, 2 special character (Exclamation mark and full stop)
These are the basic building blocks of any English paragraph. Likewise five fundamental things are used to construct any simple or complex program that are **keywords, identifiers, operators, separators, literals.**

- **Keywords**
  Keywords are the reserved words whose meaning is pre-defined by the programming language specification. They convey some special meaning in the programming.
  Let's take an example of C keyword: **int** in C is used to declare an integer type variable.
  C has total 32 keywords.

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Identifier

In programming we need some data to work upon, since program without data is worthless. And if we have defined some data in memory then there must be some easiest way to access them later.
As the name sounds Identifier might be used for identifying something. Yes you are right, identifiers are used for identifying various programming elements (they might be some variables, or some function or any other programming element). Or in simple words Identifiers are the name given to various programming element.

There are some rules that must be followed by an identifier:

- It must not be a keyword.
- It can contain alphabets a-z A-Z, digits 0-9, and underscore _ symbol.
- It must not contain any special character other than_.
- It must begin with an alphabet or underscore _ symbol.

Examples of some valid identifiers: num, num1, _num1, first_name, _name_
Examples of some invalid identifiers: 1num, first name, email@id, name.

Operators

Operators are symbols given to any mathematical or logical operation. C language provides a rich set of operators. There are basically seven types of operators in C:

- Arithmetic operator : + - * / %
- Assignment operator: =
- Relational operator: > , < , >=, <=, ==, !=
- Logical operator : && || !
- Bitwise operator : >> << & | ^ ~
- Increment/Decrement operator : ++ --
- Conditional/Ternary operator : ? :
- Miscellaneous operator sizeof()

- **Separators**

  Separators are used for separating various programming element from one another. Separators in programming are as vital as they are in English paragraphs, as I don't think anyone could ever read an English sentence clearly without separator (Spaces) and punctuation marks. Various types of separators used in C programming are:

  . , : ; ( ) { } [ ] White spaces (Space, \t, \n)

  Amongst all semicolon ; is the most important one as you will find everywhere in all C programs. Semicolon ; is used to terminate any C statement and is also used for separating one statement from another.

- **C CHARACTER SET**

  C character set defines the valid set of alphabets, digits and special characters that are allowed in C. C character set consists of:

- Alphabets: a-z A-Z
- Digits: 0-9
- Special characters: ` ~ ! @ # $ % ^ & * ( ) _ - + = { } [ ] | \ : ; " ' < , > . ? /

Data types in C programming

Data type defines the type of the data that any variable will store. It is a system of declaring variables of different types. Data types are broadly categorized into three categories which are:

1. Primitive/Pre-defined
2. User defined
3. Derived

Primitive/Pre-defined data type:

Primitive data types are the basic data types that are needed for performing general operations.
Example: int, char, float, double

User defined data type:

User defined data types are defined by user according to their need. They are defined or created from the primitive data types.
Example: struct, union, array, pointer

Derived data types:

Derived data types are basically the primitive data type with increased range of values it can store.
Example: short, long long int, double Below is a list of C data types:

Data type	Size	Range	Format specifier
char	1 byte	-127 to +127	%c
unsigned char	1 byte	0 to 255	%c
short	2 byte	-32,767 to 32,767	%hi, %d, %i
unsigned short	2 byte	0 to 65,535	%hu, %d, %i
int	4 bytes	-2,147,483,847 to +2,147,483,847	%d, %i, %u
unsigned int	4 bytes	0 to 4,294,967,295	%u
long	4 bytes	-2,147,483,847 to +2,147,483,847	%d, %i, %u, %ld, %l, %li
unsigned long	4 bytes	0 to 4,294,967,295	%lu, %u
long long	8 bytes	-9,223,372,036,854,775,807 to +9,223,372,036,854,775,807	%lli
unsigned long long	8 bytes	0 to 18,446,744,073,709,551,615	%llu
float	4 bytes	1.2E-38 to 3.4E+38	%f
double	8 bytes	2.3E-308 to 1.7E+308	%lf
long double	10 bytes	3.4E-4932 to 1.1E+4932	%Lf

Hello World Program in C

Here we will write our first C program. A Hello World program.

Program:

```
1    #include <stdio.h>
2
3    int main()
4    {
5       printf("Hello World!");
6       return 0;
7    }
```

Know it:

- #include <stdio.h> is a pre-processor directive. When the above program is compiled the compiler replaces the first line with entire contents of stdio.h file. The stdio.h (Standard Input/Output) is called as header file and contains functions for basic input/output operations.
- int main() is a function and defines the starting point of our program. It contains two parts:
  int It is the return type of the function. It specifies that the main() function should return an integer value when its work is finished.
  main() It is an identifier or name of the function.
  Note: The starting point of any program in C is defined by main function.
- The next line contains pair of curly braces. In C we use a pair of curly braces to define a block of statements. An opening curly brace is always followed by a closing curly brace.
- printf(); is a function used to print any data on screen. Anything inside the double quotes " " will be printed as it is on screen.
- return 0; returns an integer value to the operating system or the Runtime environment of C acknowledging OS/C Runtime Environment that the program has terminated successfully.

If you have written the above program same as in the source code without making any mistake then it should print Hello World! on the screen.

Variables and constants in C

Variables:

Variables are the storage location in memory that holds some value. We define a variable to make sure a stored value in memory is accessible by user later. Variables are accessed by their identifiers (Symbolic name that refer to the value stored in the memory) to which they are linked with. In order to use any variable and the value it contains, the variable must be declared first.

Declaring variables in C:

Syntax: data-type variable-name;
Here data-type refers to valid C Data type.
And variable-name refers to valid C identifier.
Note: In C all variables must be declared on the top.

Example:

```
int num1;
```

The above code in C will declare an integer variable in memory with Garbage value. Garbage value is a random value of specific type (int in our case) stored in the variable. Hence, variables in C must be initialized prior to their use.

Initializing a variable:

Assigning a value to the variable is called as initializing a variable. Values are initialized or assigned to any variable using assignment operator =.
Example:

```
int num1 = 10;
```

Above code will now declare an integer variable num1 in memory with 10 stored in it.

Declaring multiple variables:

Multiples variables in C can be declared one after another declaration.

```
int num1;
int num2 = -30;
unsigned int num3 = 10;
float num4 = 1.2;
char ch = 'a';
```

The above code will declare an integer variable num1 with garbage value, num2 with value -30, an unsigned integer variable with value 10, float variable num4 with value 1.2 and a character variable ch with value 'a'.
Note: Values of character variable must be assigned inside a single quote. And must only contain single character.

Declaring multiple variables of same type:

Multiple variables of same type can be declared and initialized by separating them with comma,.

```
int num1, num2, num3;
int num4 = 10, num5, num6 = -1290;
```

The above code will declare 6 integer variables num1, num2, num3, num5 uninitialized containing garbage values and num4 with value 10, num6 with value -1290.

Taking input from user using scanf() function

We use scanf() function in C to take formatted input from user. It is defined in stdio.h header file. Here we will see how to take simple inputs from scanf() function.

Syntax:

```
int scanf(const char * format, ...);
```

How to Use:

```
scanf("format-specifier", &variable-name);
```

Here **format-specifier** denotes the format specifier of input. And **variable-name** specifies the name of variable in which input is to be taken. The variable name is prefixed by &(Address of operator) which returns the address of the variable. If & is somewhat confusing you leave it we will see in-depth use of & later in pointers. Lets see basic input program.

Program:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int num;
6
7     printf("Enter any number : ");
8     scanf("%d", &num);
```

```
9
10      printf("Number = %d", num);
11
12      return 0;
13  }
```

Output:

Enter any number: 120
Number = 120

Taking multiple inputs:

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int num1, num2,num3;
6      float f1, f2;
7      printf("Enter two number:\n");
8
9      //Multiple scanf() can be used to take multiple inputs.
10     scanf("%d", &num1);
11     scanf("%d", &num2);
12
13     printf("Enter a number and two float values:\n");
14
15     //All inputs can also be taken in a single scanf()
16     scanf("%d%f%f",&num3,&f1,&f2);
17
18     printf("Num1 = %d, Num2 = %d, Num3 = %d\n", num1,num2,num3);
19     printf("f1 = %f, f2 = %f\n",f1,f2);
20     return 0;
21  }
```

Output:

Enter two number:
12
10
Enter a number and two float values:
130
1.2
1.6
Num1 = 12, Num2 = 10, Num3 = 130
f1 = 1.200000, f2 = 1.600000

Exercise of Structured C Programming

A. Taking User Input

1) Write a C program to enter two numbers and find their sum.
2) Write a C program to enter length and breadth of a rectangle and find its perimeter & area.
3) Write a C program to enter radius of a circle and find its diameter, circumference and area.
4) Write a C program to enter base and height of a triangle and find its area.
5) Write a C program to enter length in centimeter and convert it into meter and kilometer.
6) Write a C program to enter temperature in Celsius and convert it into Fahrenheit.
7) Write a C program to enter temperature in Fahrenheit and convert it into Celsius
8) Write a C program to convert days into years, weeks and days.
9) Write a C program to find power of any number (x^y).
10) Write a C program to enter any number and calculate its square root.
11) Write a C program to enter two angles of a triangle and find the third angle.
12) Write a C program to calculate area of an equilateral triangle.
13) Write a C program to enter marks of five subjects and calculate total, average and percentage.
14) Write a C program to enter P, T, and R and calculate Simple and Compound Interest.
    [Hints:
    Compound Interest:
    Formula: P * ((1 + R / 100) T - 1) (where P = Principle, T = Time, R = Rate)
    Simple Interest:
    Formula: Principle * Time * Rate / 100]

B. Conditional/Ternary operator

1) Write a C program to find maximum and minimum among three numbers using conditional/ternary operator.
2) Write a C program to check whether a number is even or odd
3) Write a C program to check whether year is leap year or not using conditional/ternary operator.
4) Write a C program to check whether character is an alphabet or not using conditional/ternary operator.

C. If-Else

1) Write a C program to check whether a number is negative, positive or zero.
2) Write a C program to check whether a number is divisible by 5 and 11 or not.
3) Write a C program to count total number of notes (money) in given amount.
4) Write a C program to check whether a character is alphabet or not.
5) Write a C program to input any alphabet and check whether it is vowel or consonant.
6) Write a C program to input any character and check whether it is alphabet, digit or special character.
7) Write a C program to check whether a character is uppercase or lowercase alphabet.
8) Write a C program to input week number and print week day.
9) Write a C program to input month number and print number of days in that month.
10) Write a C program to input angles of a triangle and check whether triangle is valid or not.
11) Write a C program to input all sides of a triangle and check whether triangle is valid or not.
12) Write a C program to check whether the triangle is equilateral, isosceles or scalene triangle.
13) Write a C program to find all roots of a quadratic equation.
14) Write a C program to calculate profit or loss.
15) Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

    Percentage >= 90%: Grade A
    Percentage >= 80%: Grade B
    Percentage >= 70%: Grade C
    Percentage >= 60%: Grade D
    Percentage >= 40%: Grade E
    Percentage < 40%: Grade F

16) Write a C program to input basic salary of an employee and calculate its Gross salary according to following:

    Basic Salary >= 10000: HRA = 20%, DA = 80%
    Basic Salary >= 20000: HRA = 25%, DA = 90%
    Basic Salary >= 30000: HRA = 30%, DA = 95%

17) Write a C program to input electricity unit charges and calculate total electricity bill according to the given condition:

    For first 50 units BDT. 0.50/unit
    For next 100 units BDT. 0.75/unit
    For next 100 units BDT 1.20/unit
    For unit above 250 BDT. 1.50/unit

    An additional surcharge of 20% is added to the bill

D. Switch case:

1) Write a C program to check whether an alphabet is vowel or consonant using switch case.
2) Write a C program to print day of week name using switch case.
3) Write a C program print total number of days in a month using switch case.
4) Write a C program to find maximum between two numbers using switch case.
5) Write a C program to check whether a number is even or odd using switch case.
6) Write a C program to find roots of a quadratic equation using switch case.
7) Write a C program to create Simple Calculator using switch case.

E. Loop programming:

1) Write a C program to print all natural numbers in reverse (from n to 1). - using while loop
2) Write a C program to print all alphabets from a to z. - using while loop
3) Write a C program to print sum of all even numbers between 1 to n.
4) Write a C program to print sum of all odd numbers between 1 to n.
5) Write a C program to print prime numbers between 1 to n.
6) Write a C program to print table of any number.
7) Write a C program to enter any number and calculate sum of all natural numbers between 1 to n.
8) Write a C program to find first and last digit of any number.
9) Write a C program to count number of digits in any number.
10) Write a C program to calculate sum of digits of any number.
11) Write a C program to calculate product of digits of any number.
12) Write a C program to swap first and last digits of any number.
13) Write a C program to find sum of first and last digit of any number.
14) Write a C program to enter any number and print its reverse.
15) Write a C program to enter any number and check whether the number is palindrome or not.
16) Write a C program to find frequency of each digit in a given integer.
17) Write a C program to enter any number and print it in words.
18) Write a C program to print all ASCII character with their values.
19) Write a C program to find power of any number using for loop.
20) Write a C program to enter any number and print all factors of the number.
21) Write a C program to enter any number and calculate its factorial.
22) Write a c program to find out the sum of series $1 + 2 + ... + n$.
23) Write a c program to find out the sum of series $7 + 20 + 33 + ... +$ up to 100 term
24) Write a c program to find out the sum of series $5 - 11 + 17 - .... +$ up to 75 term
25) Write a c program to find out the sum of series $1 + (1+2) + (1+2+3) .... + (1 + 2 + 3 + ... + n)$.
26) Write a c program to find out the sum of series $1^2 + 2^2 + ... + n^2$.
27) C Program to Find Sum of the Series $1/1! + 2/2! + 3/3! + ...1/N!$
28) Write a C program to find GCD of two numbers.

29) Write a C program to find LCM of two numbers.
30) Write a C program to check whether a number is Armstrong number or not.
31) Write a C program to check whether a number is Perfect number or not.
32) Write a C program to check whether a number is Strong number or not.
33) Write a C program to print all Armstrong numbers between 1 to n.
34) Write a C program to print all Perfect numbers between 1 to n.
35) Write a C program to print all Strong numbers between 1 to n.
36) Write a C program to enter any number and print its prime factors.
37) Write a C program to print Fibonacci series up to n terms.
38) Write a C program to find one's complement of a binary number.
39) Write a C program to find two's complement of a binary number.
40) Star pattern programs - Write a C program to print the given star patterns.

a)	***** ***** ***** ***** *****	d)	* ** *** **** *****
b)	***** **** *** ** *	e)	***** **** *** ** *
c)	***** **** *** ** * * ** *** **** *****	f)	* ** *** **** ***** **** *** ** *

41) Number pattern programs - Write a C program to print the given number patterns.

a)	1 12 123 1234 12345	c)	1 12 123 1234 12345
b)	54321 4321 321 21 1	d)	54321 4321 321 21 1

F. Array and Matrix programming

1) Write a C program to read and print elements of array. - using recursion.
2) Write a C program to print all negative elements in an array.
3) Write a C program to find sum of all array elements. - using recursion.
4) Write a C program to find maximum and minimum element in an array. - using recursion.
5) Write a C program to find second largest element in an array.
6) Write a C program to count total number of even and odd elements in an array.
7) Write a C program to count total number of negative elements in an array.
8) Write a C program to copy all elements from an array to another array.
9) Write a C program to insert an element in an array.
10) Write a C program to delete an element from an array at specified position.
11) Write a C program to print all unique elements in the array.
12) Write a C program to count total number of duplicate elements in an array.
13) Write a C program to delete all duplicate elements from an array.
14) Write a C program to count frequency of each element in an array.
15) Write a C program to merge two array to third array.
16) Write a C program to find reverse of an array.
17) Write a C program to put even and odd elements of array in two separate array.
18) Write a C program to search an element in an array.
19) Write a C program to sort array elements in ascending order.
20) Write a C program to sort array elements in descending order.
21) Write a C program to sort even and odd elements of array separately.
22) Write a C program to add two matrices.
23) Write a C program to subtract two matrices.

24) Write a C program to perform Scalar matrix multiplication.
25) Write a C program to multiply two matrices.
26) Write a C program to check whether two matrices are equal or not.
27) Write a C program to find sum of main diagonal elements of a matrix.
28) Write a C program to find sum of minor diagonal elements of a matrix.
29) Write a C program to find sum of each row and column of a matrix.
30) Write a C program to interchange diagonals of a matrix.
31) Write a C program to find upper triangular matrix.
32) Write a C program to find lower triangular matrix.
33) Write a C program to find sum of upper triangular matrix.
34) Write a C program to find transpose of a matrix.
35) Write a C program to find determinant of a matrix.
36) Write a C program to check Identity matrix.
37) Write a C program to check sparse matrix.
38) Write a C program to check Symmetric matrix.

G. String Manipulation

1) Write a C program to find length of a string.
2) Write a C program to copy one string to another string.
3) Write a C program to concatenate two strings.
4) Write a C program to compare two strings.
5) Write a C program to convert lowercase string to uppercase.
6) Write a C program to convert uppercase string to lowercase.
7) Write a C program to toggle case of each character of a string.
8) Write a C program to find total number of alphabets, digits or special character in a string.
9) Write a C program to count total number of vowels and consonants in a string.
10) Write a C program to count total number of words in a string.
11) Write a C program to find reverse of a string.
12) Write a C program to check whether a string is palindrome or not.
13) Write a C program to reverse order of words in a given string.
14) Write a C program to find first occurrence of a character in a given string.
15) Write a C program to find last occurrence of a character in a given string.
16) Write a C program to search all occurrences of a character in given string.
17) Write a C program to count occurrences of a character in given string.
18) Write a C program to find highest frequency character in a string.
19) Write a C program to find lowest frequency character in a string.
20) Write a C program to count frequency of each character in a string.
21) Write a C program to remove first occurrence of a character from string.
22) Write a C program to remove last occurrence of a character from string.
23) Write a C program to remove all occurrences of a character from string.
24) Write a C program to remove all repeated characters from a given string.
25) Write a C program to replace first occurrence of a character with another in a string.
26) Write a C program to replace last occurrence of a character with another in a string.

27) Write a C program to replace all occurrences of a character with another in a string.
28) Write a C program to find first occurrence of a word in a given string.
29) Write a C program to find last occurrence of a word in a given string.
30) Write a C program to search all occurrences of a word in given string.
31) Write a C program to count occurrences of a word in a given string.
32) Write a C program to remove first occurrence of a word from string.
33) Write a C program to remove last occurrence of a word in given string.
34) Write a C program to remove all occurrence of a word in given string.
35) Write a C program to trim leading white space characters in a string.
36) Write a C program to trim trailing white space characters in a string.
37) Write a C program to trim both leading and trailing white space characters in a string.
38) Write a C program to remove all extra blank spaces from a given string.

H. User Define Functions programming

1) Write a C program to find cube of any number using function.
2) Write a C program to find diameter, circumference and area of circle using functions.
3) Write a C program to find maximum and minimum between two numbers using functions.
4) Write a C program to check whether a number is even or odd using functions.
5) Write a C program to check whether a number is prime, Armstrong or perfect number using functions.
6) Write a C program to find all prime numbers between given interval using functions.
7) Write a C program to print all strong numbers between given interval using functions.
8) Write a C program to print all Armstrong numbers between given interval using functions.
9) Write a C program to print all perfect numbers between given interval using functions.
10) Write a C program to find power of any number using recursion.
11) Write a C program to print all natural numbers between 1 to n using recursion.
12) Write a C program to print all even or odd numbers in given range using recursion.
13) Write a C program to find sum of all natural numbers between 1 to n using recursion.
14) Write a C program to find sum of all even or odd numbers in given range using recursion.
15) Write a C program to find reverse of any number using recursion.
16) Write a C program to check whether a number is palindrome or not using recursion.
17) Write a C program to find sum of digits of a given number using recursion.
18) Write a C program to find factorial of any number using recursion.
19) Write a C program to generate nth Fibonacci term using recursion.
20) Write a C program to find GCD of two numbers using recursion.
21) Write a C program to find LCM of two numbers using recursion.
22) Write a C program to display all array elements using recursion.
23) Write a C program to find sum of elements of array using recursion.
24) Write a C program to find maximum and minimum elements in array using recursion.

Lecture-2

Basic structure of a C program:

1. Including header files
2. Macro definition
3. Global declaration
4. Main declaration
   {
   local declaration
   statement sequence
   other function call
   }
5. User defined function

Statements are parts of your program that actually performs operation. Statements are contained within functions.

All C statements end with a semicolon. C does not recognize the end of the line as a terminator. This means there are no constraints on the position of statements within a line. Also you may place two or more statements on one line.
The standard library contains functions to perform I/O, string manipulation, mathematics and much more.

So, we can start our first C program that will print "This is our first C program" to the standard output device.

```
#include<stdio.h>
void main(void)
{
printf("This is our first C program");
}
```

1. include is a pre processor directive that tells the compiler to include library functions contained in stdio.h file with your program.
2. main function
3. { beginning of main function
4. library function printf, that prints string within " "
5. } end of a function.

Character that can be used in C:

1. Alphabets: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.
   A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.
2. Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
3. Special symbols: ~ ` ! @ # $ % ^ & * ( ) _ - + = | \ { } [ ] ; : " ' < > , . ? /

White Space: blank, new line, tab.

In a program the smallest individual units are known as C-tokens

5 types of tokens in C: keywords, identifiers, constants, operators & punctuators.

Keywords: Keywords are specially reserved words that have strict meaning as individuals token in C.

ANSI standard: 32 keywords.

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	far	for
goto	if	int	long	near
register	return	short	signed	static
struct	switch	typedef	union	unsigned
void	while.			

Additional keywords like asm etc. are in turbo C.

Identifiers: Identifiers are user defined name of objects in a program. An identifier is composed of a sequence of letters, digits and underscores (_). An identifier must begin with a letter or underscore. Ex: sb_int, cse_aust, s_99, _x11.

Constants: C constants can be divided into 2 major categories.

1. Primary constants.
   a. Integer constants: 123, +234, -876 etc.
   b. Real constants: 23.4, -45.6, 0.56, 4.2e7, 9.3e-4, -8.1e4, -2e-4 etc.
   c. Character constants: 'A', 'b', 'x', 'S', '&' etc.

   * Every keyboard key is a character constant.

2. Secondary constants.
   a. Array
   b. Pointer
   c. Structure
   d. Union
   e. Enum

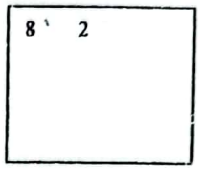
Lecture - 3

Operators:

1. Binary operator: + , - , / , * , %

Example:

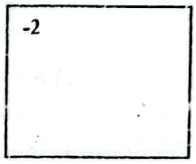
```
void main(void)
{
int a = 5, b= 3;
printf("%d\t%d",a+b,a-b);
}
```



2. Unary operator: + ,-

Example:

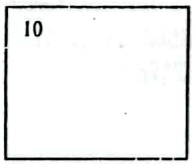
```
void main(void)
{
int a = 5, b= 3;
printf("%d",-a+b);
}
```



3. Relational operator: > , > = ,< , < = , = = , ! =
   true will return 1 and false will return 0.

Example:

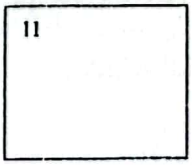
```
void main(void)
{
int a = 5, b= 3;
printf("%d%d",a>b,a= =b);
}
```



4. Logical operator: && (AND) , || (OR) , ! (NOT)
   true will return a nonzero value and false will return 0.

Example:

```
void main(void)
{
int a= 5, b= 3;
printf("%d%d",(a>b)&&(a!=b),!(b>a)||(b= =a));
}
```



5. Bitwise operator: & (AND), | (OR), ^ (XOR), <<, >>, ~

b	A AND b
0	0
1	0
0	0
1	1

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

b	a XOR b
0	0
1	1
0	1
1	0

a	NOT a
0	1
1	0

Example:

```
void main(void)
{
int a = 40, b= 15;
printf("%d", ~(((a>>2) & (b<<3)) ^ (a | b)));
}
```

-40

6. Increment and decrement operator: ++, --

Example:

```
void main(void)
{
int i = 5;
printf("%d%d",i++,i--);
printf("%d",i);
}
```

455

7. Assignment operator: =, +=, *=, -=, /=, %=, &=, |=, ^=, <<=, >>=

Example:

```
void main(void)
{
int a=5,b=3;
a+=b; b*=a;
printf("%d%d",a,b);
}
```

824

Punctuators: ` @ # $ ( ) { } [ ] ; : " ' , . ?

Lecture - 4

Instructions:

1. Type Declaration Instruction:
```
int x ;
float y ;
char z ;
```

2. Input / Output Instruction
```
Scanf(...);
printf(...);
inportb(...);
outportb(...);
```

3. Arithmetic Instruction
```
c = b + d ;
p = m * n / q ;
```

4. Control Instruction
   a. Sequence control instruction
   b. Selection or decision control instruction
   c. Repetition or Loop control instruction
   d. Case control instruction

Data Types in C:

Type	Keyword	Size	Range
Void	void	0 byte	0
Character	char	1 byte	unsigned : 0 to (2^8-1) signed : (-2^7) to $+(2^7-1)$
Integer	int	2 bytes (for DOS) 4 bytes (for Unix)	unsigned: 0 to $(2^{16}-1)$ signed : (-2^{15}) to $+(2^{15}-1)$
Floating point	float	4 bytes	unsigned : 0 to $(2^{32}-1)$ signed : (-2^{31}) to $+(2^{31}-1)$
Double	double	8 byte	unsigned : 0 to $(2^{64}-1)$ signed : (-2^{63}) to $+(2^{63}-1)$

Type modifiers: signed, unsigned; long, short.
Size of short is given into the above data types table.
Size of long = 2n bytes, where short = n bytes.
*Exception: Size of long double = 10 bytes

E 101

:onstants:

integer: 234
long integer: 1234l / 1234L unsigned
integer: 1234u / 1234U unsigned long
integer: 1234ul / 1234UL float constant:
123.4f / 123.4F
double constant: 123.4
long double: 12.34l
hex constant: 0xff / 0XFF
octal constant: 077
character constant (ASCII): 'a'

ASCII = American standard code of Information & Interchange.

Interpreting characters (scape sequence):

\a → alert (bell)
\b → backspace
\f → form feed
\n → new line
\r → carriage return
\t → horizontal tab
\v → vertical tab
\\ → back slash
\' → single quote
\" → double quote
\? → question mark
\0 → NULL

Example:

```
#include<stdio.h>
void main(void)
{
printf("\tHellow \nWorld");
}
```

```
    Hellow
World
```

Lecure 5

'ormat specifier: printf(...) converts, formats and prints its argument on the standard utput under control of the format.

Format specifier	Printed as
%d , %i	Decimal
%c	Character
%o	Unsigned octal number
%x , %X	Unsigned hexadecimal number
%u	Unsigned integer
%ld	Long integer
%hd	Short integer
%lf	Double
%s	String
%Lf	Long double
%p	Address of variable
%%	To get output of an %
%f	Floating point number

Example 1:

```
void main(void)
{
int x,y;
printf("Enter two number");
scanf("%d%d",&x,&y);
printf("%d%d",x,y);
}
```

```
Enter two number5
6
56
```

Example 2:

```
void main(void)
{
int x = 1;
printf("Enter %d \n hellow ",x);
}
```

```
Enter 1
hellow
```

Example 3:

```
void main(void)
{
char x;
scanf("%c",&x);
printf("%d",x);
}
```

```
A
65
```

E 101

xample 4:

```
void main(void)
{
int a = 5;
printf("Address = %p \nData = %d ",&a,a);
}
```

Address = FFF0
Data = 5

xample 5:

```
void main(void)
{
int x = 128;
char c ; c = x;
printf("%d",c);
}
```

-128

Example 6:

```
void main(void)
{
printf("%d ",32768);
}
```

-32768

Type conversion: (type) expression

Example 1:

```
void main(void)
{
int x ,y ;
x = 3;
y = 2;
printf("%d %d %f %d", x + y, x - y, (float) x / y, x % y);
}
```

511.5000001

Example 2:

```
char ch;
int i;
float f;
double d;
ch / i + f * d – f + i = double
```

Lecture – 6

Operator precedence and associativity:

1. ( ) , [] → Left to right
2. ++ (postfix) , -- (postfix) → Right to left
3. ! (not) , ~ (1's complement) , + (unary) , - (unary) , ++ (prefix) , -- (prefix) , & (address) , * (indirection) , sizeof → Right to left
4. * , / , % (modulus)→ Left to right
5. + (binary) , - (binary) → Left to right
6. << (shift left) , >> (shift right) → Left to right
7. < , <= , > , >= → Left to right
8. == , != → Left to right
9. & (bitwise AND) → Left to right
10. ^ (bitwise XOR) → Left to right
11. | (bitwise OR) → Left to right
12. && (logical AND) → Left to right
13. || (logical OR) → Left to right
14. ? : (a ? x : y) → Right to left
15. = , * = , / = , % = , + = , - = , & = , ^ = , | = , << = , >> = → Right to left
16. , (comma) → Left to right

Dealing with expressions:

Example 1:

```
void main(void)
{
int x=2,n=2;
x=n++;
printf("%d",x);
x=++n;
printf("%d",x);
}
```

24

Example 2:

```
void main(void)
{
int x=2, y=3;
x*=y;
printf("%d",x);
x=x*y;
printf("%d",x);
x*=y+1;
printf("%d",x);
}
```

61872