Unit 1

Introduction to programming steps

An Overview of programming languages

A computer programming language is an intermediate for communication with the system by a user.

> **A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal.**
>
> **Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing**
>
> **Programming languages are mainly classified in to three categories.**

- **High level language**

  High level language like C,C++,JAVA,PHP etc. High-level languages are designed to be used by the human operator or the programmer. High level languages also require translation to machine language before execution. The translation of high level to machine level language is done by either compiler or interpreter.

- **Assembly language**

  An assembly language is a low-level programming language. And used for microprocessors and other programmable devices. It is also known as assembly code. This translation is done by Assembler.

- **Machine language**

  Machine language is a collection of binary digits or bits. The computer reads and interprets it. This is the only language computer can understand.

What is computer program

computer program is a collection of instructions that can be executed by a **computer** to perform a specific task. A **computer program** is usually written by a **computer** programmer in a **programming** language.

Compilation in programming

Compilation is the process the computer takes to convert a high-level programming language into a machine language that the computer can understand. The software which performs this conversion is called a compiler.

What is a compiler

A compiler is a translator which converts high level computer language programs in to its equivalent machine form (binary form)

A compiler is a translator which converts source code(human understandable form) in to object code (machine understandable form).

Linking and loading in programming

The key difference between linking and loading is that the linking generates the executable file of a program whereas, the loading loads the executable file obtained from the linking into main memory for execution.

Linking

Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed. .

Loading

Loading a program involves reading the contents of the executable file containing the program instructions into memory,

and then carrying out other required preparatory tasks to prepare the executable for running.

Testing and Debugging in programming

Testing is a process of finding bugs or errors in a software product that is done manually by tester or can be automated. Debugging is a process of fixing the bugs found in testing phase. Programmer or developer is responsible for debugging and it can't be automated

Testing

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software.

Debugging

Debugging is the routine process of locating and removing computer program bugs, errors , which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications.

Documentation

Software documentation is written text or illustration that accompanies computer software or is embedded in the source code. The documentation either explains how the software operates or how to use it, and may mean different things to people in different roles.

Character set

Like every other language 'C' also has its own character set. A program is a set of instructions that when executed, generate an output. The data that is processed by a program consists of various characters and symbols. The output generated is also a combination of characters and symbols.

Character set in programming language C includes letters from a to z (A to Z) , numbers from 0 to 9 ,special symbols like @,#,&,*,”,',:,;,!,?. and white space (tab,newline,space).

Upper case letters A-Z

Lower case letters a_z

Numbers from 0-9

Special symbols like @, *,”,',:,; ,! „? +_-

White space like tab, new line, space.

C language support 256 characters

Variables and Identifiers

Variables

a **variable** is a name given to a memory location, that is used to hold a value. **Variable** is only a kind of **identifier.** A variable must be declared first before it is used somewhere inside the program. A variable name is formed using characters, digits and an underscore.

Identifier

An identifier is nothing but a name assigned to an element in a program. Example, name of a variable, function, etc.

In **C** language **identifiers** are the names given to variables, constants, functions and user-define data.The **identifier** is only used to identify an entity uniquely in a program at the time of execution. An Indetifier

can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore(_).

Keywords

Keywords are preserved words that have special meaning in C language. The meaning has already been described. These meaning cannot be changed. There are total 32 keywords in C language.

In 'C' every word can be either a keyword or an identifier.

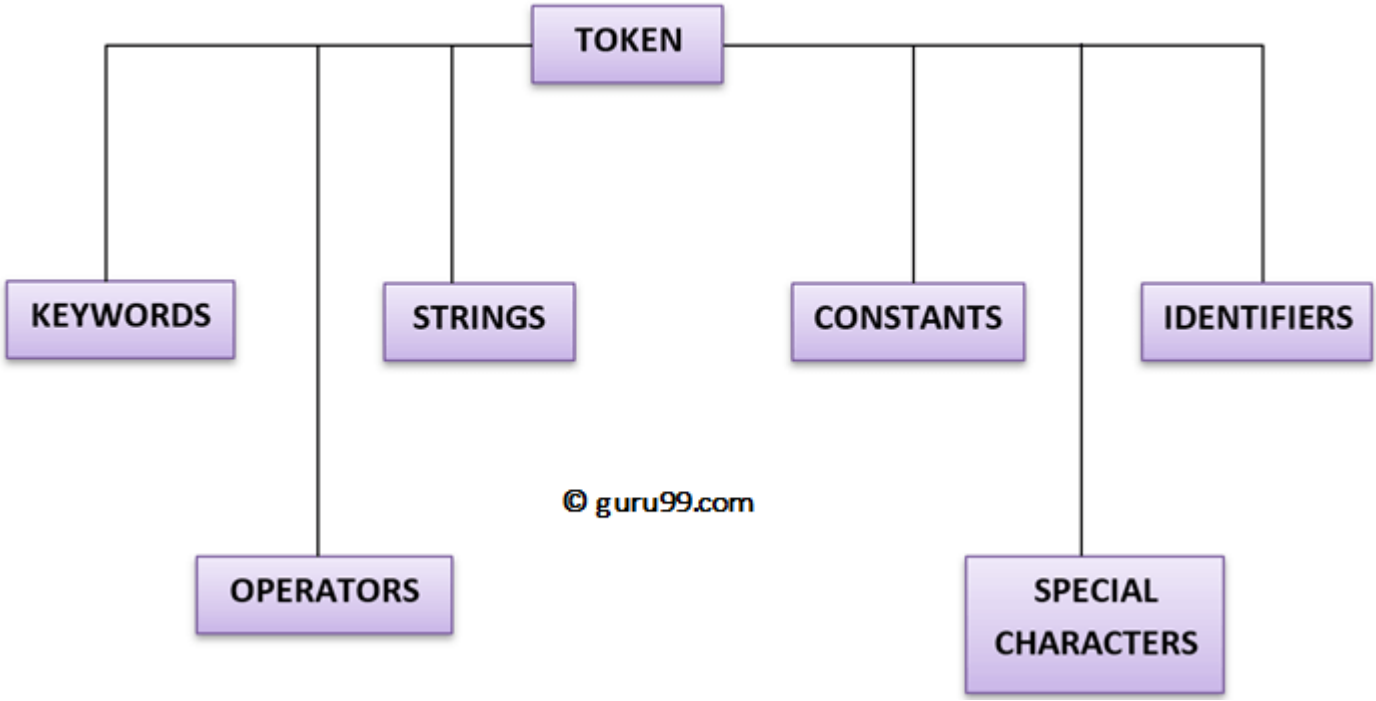
Keywords have fixed meanings, and the meaning cannot be changed. They act as a building block of a 'C' program. There are total 32 keywords in 'C'. Keywords are written in lowercase letters.

Following table represents the keywords in 'C',

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

Token in C

TOKEN is the smallest unit in a 'C' program. It is each and every word and punctuation that you come across in your C program. The compiler breaks a program into the smallest possible units (tokens) and proceeds to the various stages of the compilation. A token is divided into six different types, viz, Keywords, Operators, Strings, Constants, Special Characters, and Identifiers.



Tokens in C

Data types

'C' provides various data types to make it easy for a programmer to select a suitable data type as per the requirements of an application. Following are the three data types:

1. Primitive data types
2. Derived data types
3. User-defined data types

There are five primary fundamental data types,

1. int for integer data
2. char for character data

3. float for floating point numbers
4. double for double precision floating point numbers
5. void

Array, functions, pointers, structures are derived data types. 'C' language provides more extended versions of the above mentioned primary data types. Each data type differs from one another in size and range. Following table displays the size and range of each data type.

Data type	Size in bytes	Range
Char or signed char	1	-128 to 127
Unsigned char	1	0 to 255
int or signed int	2	-32768 to 32767
Unsigned int	2	0 to 65535
Short int or Unsigned short int	2	0 to 255
Signed	2	-128 to 127

short int		
Long int or Signed long int	4	-2147483648 to 2147483647
Unsigned long int	4	0 to 4294967295
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
Long double	10	3.4E-4932 to 1.1E+4932

Integer data type

Integer is nothing but a whole number. The range for an integer data type varies from machine to machine. The standard range for an integer data type is -32768 to 32767. An integer typically is of 2 bytes which means it consumes a total of 16 bits in memory.

Whenever we want to use an integer data type, we have place int before the identifier such as,

```
int age;
```

Here, age is a variable of an integer data type which can be used to store integer values.

Floating point data type

Like integers, in 'C' program we can also make use of floating point data types. The 'float' keyword is used to represent the floating point data type. It can hold a floating point value which means a number is having a fraction and a decimal part. A floating point value is a real number that contains a decimal point. The data type double and long double are used to store real numbers with precision up to 14 and 80 bits respectively.

While using a floating point number a keyword float/double/long double must be placed before an identifier. The valid examples are,

```
float division;
double BankBalance;
```

Character data type

Character data types are used to store a single character value enclosed in single quotes.

A character data type takes up-to 1 byte of memory space.

Example,

```
Char letter;
```

Void data type

A void data type doesn't contain or return any value. It is mostly used for defining functions in 'C'.

Example,

```
void displayData()
```

Constants

Constants are the fixed values that never change during the execution of a program. Following are the various types of constants:

Integer constants

An integer constant is nothing but a value consisting of digits or numbers. These values never change during the execution of a program. Integer constants can be octal, decimal and hexadecimal.

1. Decimal constant contains digits from 0-9 such as,

```
Example, 111, 1234
```

Above are the valid decimal constants.

2. Octal constant contains digits from 0-7, and these types of constants are always preceded by 0.

```
Example, 012, 065
```

Above are the valid decimal constants.

3. Hexadecimal constant contains a digit from 0-9 as well as characters from A-F. Hexadecimal constants are always preceded by 0X.

```
Example, 0X2, 0Xbcd
```

Above are the valid hexadecimal constants.

The octal and hexadecimal integer constants are very rarely used in programming with 'C'.

Character constants

A character constant contains only a single character enclosed within a single quote ('). We can also represent character constant by providing ASCII value of it.

Example, 'A', '9'

Above are the examples of valid character constants.

String constants

A string constant contains a sequence of characters enclosed within double quotes ("").

Example, "Hello", "Programming"

These are the examples of valid string constants.

Real Constants

Like integer constants that always contains an integer value. 'C' also provides real constants that contain a decimal point or a fraction value. The real constants are also called as floating point constants. The real constant contains a decimal point and a fractional value.

Example, 202.15, 300.00

These are the valid real constants in 'C'.

A real constant can also be written as,

Mantissa e Exponent

Summary

- A constant is a value that doesn't change throughout the execution of a program.
- A token is the smallest unit in a program.
- A keyword is reserved words by language.
- There are total 32 keywords.

- An identifier is used to identify elements of a program.
- A variable is an identifier which is used to store a value.
- There are four commonly used data types such as int, float, char and a void.
- Each data type differs in size and range from one another.

Arithmetic Operators. **C** programming language provides all basic **arithmetic operators**: +(addition), -(subtraction), *(multiplication), /(division) and % (Modulus).

The **Arithmetic operators** are some of the **C** Programming **Operator**, which are used to perform **arithmetic operations** includes **operators** like Addition, Subtraction, Multiplication, Division and Modulus.

Expression in C

An **expression** is a combination of variables constants and operators written according to the syntax of **C language**.
In **C** every **expression** evaluates to a value .

Operators, functions, constants and variables are combined together to form **expressions**. Consider the **expression**

A + B * 5.

A+B=C

There are three kinds of expressions:

- An arithmetic expression evaluates to a single arithmetic value.
- A character expression evaluates to a single value of type character.
- A logical or relational expression evaluates to a single logical value.

Simple assignment statement

C provides an **assignment** operator for this purpose, assigning the value to a variable using **assignment** operator is known as

an **assignment statement in C**. The function of this operator is to **assign** the values or values in variables on right hand side of an expression to variables on the left hand side.

= is the assignment operator

A=5

Here the value 5 is assigned to the variable A

Basic Input Output Statement

An **input/output statement** or IO **statement** is a portion of a program that instructs a computer how to read and process information. It is to gather information from an **input** device, or sending information to an **output** device. **Input**, **Output**, Programming terms.

Input means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.

scanf() and printf() functions

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.

```
printf("Please enter a value...");



    scanf("%d", &i);
```

%d inside the scanf() or printf() functions known as **format string** and this informs the scanf() function, what type of input to expect.

Format String	Meaning

%d	Scan or print an integer as signed decimal number
%f	Scan or print a floating point number
%c	To scan or print a character
%s	To scan or print a character string. The scanning ends at whitespace

Unit Three

Decision Making within a Program

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

- if statement
- switch statement
- conditional operator statement (? : operator)
- goto statement

Decision making with if statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement

2. if....else statement
3. Nested if....else statement
4. Using else if ladder statement

Simple if statement

The general form of a simple if statement is,

```
if(expression)
{
   statement inside;
}
   statement outside;
```

If the *expression* returns true, then the **statement-inside** will be executed, otherwise **statement-inside** is skipped and only the **statement-outside** is executed.

Example:

```
#include <stdio.h>


void main( )
{
   int x, y;
   x = 15;
   y = 13;
   if (x > y )
   {
```

```
        printf("x is greater than y");
    }
}
```

x is greater than y

if...else statement

The general form of a simple if...else statement is,

```
if(expression)
{
    statement block1;
}
else
{
    statement block2;
}
```

If the *expression* is true, the **statement-block1** is executed,
else **statement-block1** is skipped and **statement-block2** is executed.

Example:

```
#include <stdio.h>


void main( )
{
    int x, y;
    x = 15;
```

```
    y = 18;
    if (x > y )
    {
        printf("x is greater than y");
    }
    else
    {
        printf("y is greater than x");
    }
}
```

```
y is greater than x
```

Nested if....else statement

The general form of a nested if...else statement is,

```
if( expression )
{
    if( expression1 )
    {
        statement block1;
    }
    else
    {
```

```
        statement block2;
    }
}
else
{
    statement block3;
}
```

if *expression* is false then **statement-block3** will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if *expression 1* is true the **statement-block1** is executed otherwise **statement-block2** is executed.

Example:

```
#include <stdio.h>


void main( )
{
    int a, b, c;
    printf("Enter 3 numbers...");
    scanf("%d%d%d",&a, &b, &c);
    if(a > b)
    {
        if(a > c)
        {
            printf("a is the greatest");
```

```
        }
        else
        {
            printf("c is the greatest");
        }
    }
    else
    {
        if(b > c)
        {
            printf("b is the greatest");
        }
        else
        {
            printf("c is the greatest");
        }
    }
}
```

---

else if ladder

The general form of else-if ladder is,

```
if(expression1)
{
```

```
    statement block1;
}
else if(expression2)
{
    statement block2;
}
else if(expression3 )
{
    statement block3;
}
else
    default statement;
```

The expression is tested from the top(of the ladder) downwards. As soon as a **true** condition is found, the statement associated with it is executed.

Example :

```
#include <stdio.h>

void main( )
{
    int a;
    printf("Enter a number...");
    scanf("%d", &a);
```

```
    if(a%5 == 0 && a%8 == 0)
    {
        printf("Divisible by both 5 and 8");
    }
    else if(a%8 == 0)
    {
        printf("Divisible by 8");
    }
    else if(a%5 == 0)
    {
        printf("Divisible by 5");
    }
    else
    {
        printf("Divisible by none");
    }
}
```

Switch statement in C

When you want to solve multiple option type problems, for example: Menu like program, where one value is associated with each option and you need to choose only one at a time, then, switch statement is used.

Switch statement is a control statement that allows us to choose only one choice among the many given choices. The expression in switch evaluates to return an integral value, which is then

compared to the values present in different cases. It executes that block of code which matches the case value. If there is no match, then **default** block is executed(if present). The general form of switch statement is,

```
switch(expression)
{
   case value-1:
         block-1;
         break;
   case value-2:
         block-2;
         break;
   case value-3:
         block-3;
         break;
   case value-4:
         block-4;
            break;
   default:
            default-block;
         break;
}
```

break statements are used to **exit** the switch block.

The conditional operator (? : Operator)

which can be used to replace **if...else** statements. It has the following general form −

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this −

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

The goto statement

goto' Statement in C language

- **goto** is a jumping statement in **c language**, which transfer the **program's** control from one statement to another statement (where label is defined).
- **goto** can transfer the **program's** within the same block and there must a label, where you want to transfer **program's** control.
  - Program's control can be transfer following by the given syntax
  - **goto label_name;**

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value	A++ = 11

	by one.	
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition	(A != B) is true.

	becomes true.	
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right	(A >= B) is not true.

	operand. If yes, then the condition becomes true.	
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition	(A && B) is false.

	becomes true.	
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Loops in C

Loops are control structures used to repeat a given section of code a certain number of times or until a particular condition is met.
Looping is one of the key concepts on any programming language. It

executes a block of statements number of times until the condition becomes false. **Loops** are of 2 types: entry-controlled and exit-controlled.

C programming has three types of loops:

1. for loop
2. while loop
3. do...while loop

While Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

```
while (condition) {
        statements;
}
```

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of do- while loop is as follows:

```
do {
 statements
} while (expression);
```

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop. programming language provides the following types of loops to handle looping requirements.

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
 statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Summary

- Looping is one of the key concepts on any programming language.
- It executes a block of statements number of times until the condition becomes false.
- Loops are of 2 types: entry-controlled and exit-controlled.
- 'C' programming provides us 1) while 2) do-while and 3) for loop.
- For and while loop is entry-controlled loops.
- Do-while is an exit-controlled loop.

Sl.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop It is more like a while

	statement, except that it tests the condition at the end of the loop body.
4	nested loops You can use one or more loops inside any other while, for, or do..while loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements.

Sr.No.	Control Statement & Description
1	break statement Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

3	goto statement Transfers control to the labeled statement.

The Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#include <stdio.h>

int main () {

  for( ; ; ) {
    printf("This loop will run forever.\n");
  }

  return 0;
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

NOTE − You can terminate an infinite loop by pressing Ctrl + C keys.

Nested loops

Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as “loop inside loop.

The placing of one **loop** inside the body of another **loop** is called **nesting**. When you "nest" two **loops**, the outer **loop** takes control of the number of complete repetitions of the inner **loop**. While

all types of **loops** may be **nested**, the most commonly **nested loops** are for **loops**.

Structured Programming in C

Structured Programming. **C** is called a **structured programming** language because to solve a large problem, **C programming** language divides the problem into smaller structural blocks each of which handles a particular responsibility.

Why C is a structured programming language?

C is called **structured programming language** because a program in **c language** can be divided into small logical functional modules or structures with the help of function procedure. **C** is high level **programming language**, so easy to understand and write a program

Using structured programming languages have the following advantages.

- Programs are easier to read and understand.
- Application programs are less likely to contain logic errors.
- Errors are more easily found.
- Higher productivity during application program development.
- Application programs are more easily maintained.

UNIT FOUR

Arrays

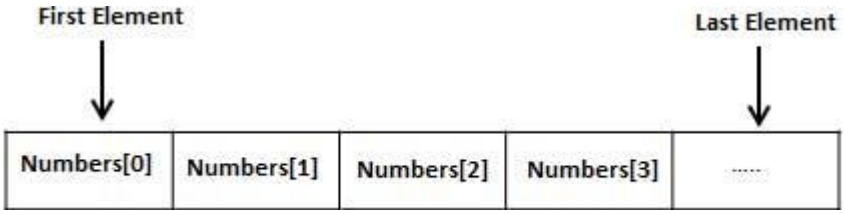
- An **array** is a collection of data items, all of the same type, accessed using a common name.
- A one-dimensional **array** is like a list; A two dimensional **array** is like a table;

An **array** can be of any **type**, For example: int , float , char etc.

Declaration of an array

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An **array** is used to store a collection of data, but it is often more useful to think of an **array** as a collection of variables of the same type. ... A specific element in an **array** is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows

data type array_name [array_size];

eg:

int number [5]; declared an integer array of name number and of size 5

Char nam[10]; declared a character array of name nam and of size 10

float total [20]; declared a floating point array of name total and of size 2

Using Arrays

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.
- Array subscripts must be of integer type. (int, long int, char, etc.)

- **VERY IMPORTANT:** Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four

Initialization of an array

Declaration and assigning values to an array is called an array initialization

Data type array_name[array_size]={elements};

Int num[5]={10,20,30,40,50};

One dimensional array

A **one-dimensional array** is a structured collection of components (often called **array** elements) that can be accessed individually by specifying the position of a component with a **single** index value.

Eg:

int number[5];

float total[10];

char name[7];

Array manipulation

Searching an element in an array

Searching is the process of finding a given value position in a list of values. It decides whether a search key is present in the data or not. It is the algorithmic process of finding a particular item in a collection of items.

to **search** an element in a given array, there are two popular **algorithms** available: Linear **Search**. Binary **Search**.

Insertion Operation in **Array**.

Insertion operation is used to insert a new element at specific position in to one dimensional **array**. In order to insert a new element into one dimensional **array** we have to create space for new element. ... We have to move last N-1 elements down in order to create space for the new element.

Or

Insertion Operation. **Insert operation** is to **insert** one or more **data** elements into an array. Based on the requirement, new element can be added at the beginning, end or any given index of array.

Deletion Operation in an array.

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

This operation is used to delete an element from specific position from one dimensional array.
In order to delete an element from one dimensional array first we have to delete element from specified position and then shift remaining elements upwards to take vacant space of the deleted element.

Finding the largest/smallest element in an array

Algorithm to find the smallest and largest numbers in an array

- Input the **array elements**.
- Initialize small = large = arr[0]
- Repeat from i = 2 to n.

- if(arr[i] > large)
- large = arr[i]
- if(arr[i] < small)
- small = arr[i]
- Print small and large.

Two dimensional arrays

Two Dimensional Array in C. The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of **rows** and columns.

2D arrays are generally known as matrix

The syntax to declare the 2D array is given below

data_type array_name[rows][columns];

eg: **int** abc[4][3];

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D array

int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Addition/multiplication of two matrices

Addition of two matrices

Add the values of the two matrixes and store it in another matrix. Display the new matrix.

Algorithm to add two matrices

- Input matrix 1 and matrix 2.

- If the number of rows and number of columns of matrix 1 and matrix 2 is equal,
- for i=1 to rows[matrix 1]
- for j=1 to columns [matrix 1]
- Input matrix 1 [i,j]
- Input matrix 2 [i,j]
- matrix 3 [i,j]= matrix 1 [i,j]+ matrix 2 [i,j];
- Display matrix 3 [i,j];

Multiplication of two Matrices

Matrix multiplication in C language to calculate the product of two matrices (two-dimensional arrays). A user inputs the orders and elements of the matrices. If the multiplication isn't possible, an error message is displayed.

Matrix Multiplication Algorithm:

- Start
- Declare variables and initialize necessary variables
- Enter the element of matrices by row wise using loops
- Check the number of rows and column of first and second matrices
- If number of rows of first matrix is equal to the number of columns of second matrix, go to step 6. Otherwise, print matrix multiplication is not possible and go to step 3.
- Multiply the matrices using nested loops.
- Print the product in matrix form as console output.
- Stop

Null terminated string as array of characters

In computer programming, a **null-terminated string** is a **character string** stored as an **array** containing
the **characters** and **terminated** with a **null character** ('\0' , called NUL in ASCII).

Null-terminated strings

So it is an array of characters, with a null character (/0) at the end.

Char greetings[6]={'H','E','L','L','O','/0’}

The above string hello is terminated with /0 (null terminating character).

Thus a **null-terminated string** contains the characters that comprise the **string** followed by a **null**. **null character** is used to mark the end of a **character** string.