



Created by: Akhilesh Kumar Mishra (BC/22/011)

Kiran Maurya (BC/22/043)

Bachelor of Computer Application, SMS VARANASI
(Batch 2019-2022)

TABLE OF CONTENT

01.

What is Programming Language?

02.

C- program

03.

Basic terms in C

04.

Operators

TABLE OF CONTENT

05.

Control Statement

06.

Function

07.

Recursion

08.

Array

TABLE OF CONTENT

09.

Pointer

10.

**Dynamic Memory
Allocation**

11.

String

12.

Structure

13.

Union

TABLE OF CONTENT

What is Programming Language?

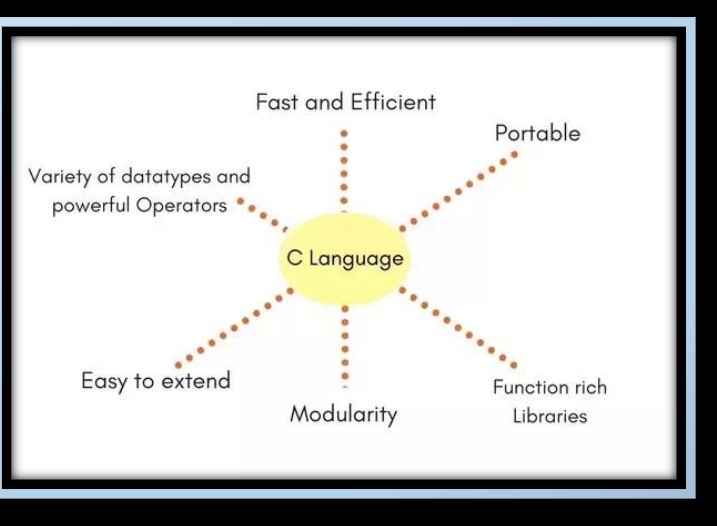
A **programming language** is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

It is a way of communication between Human being and Computer.

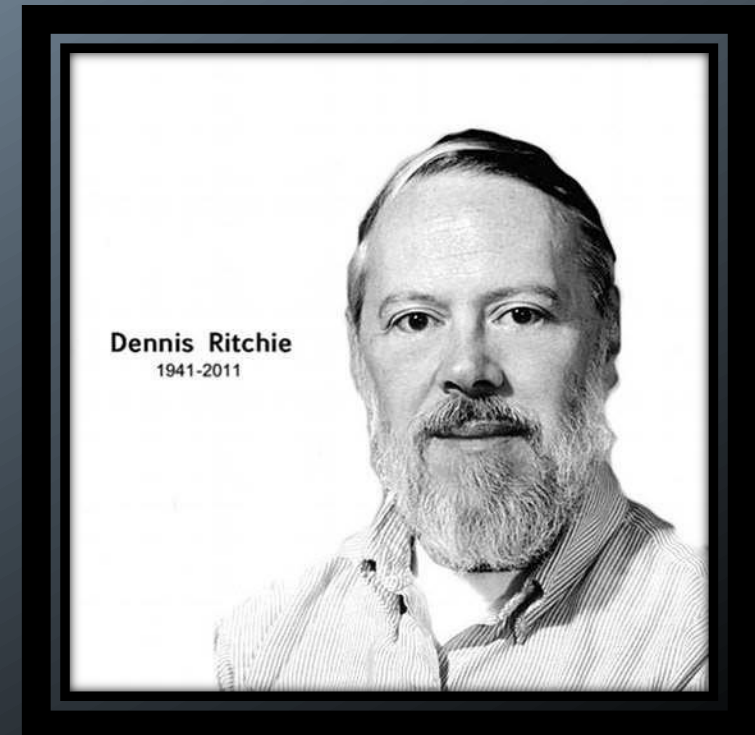
Some examples of Programming Languages are:-

C, C++, Java, Python etc.

In this Presentation we will discuss about C language.



C- Language



What is C?

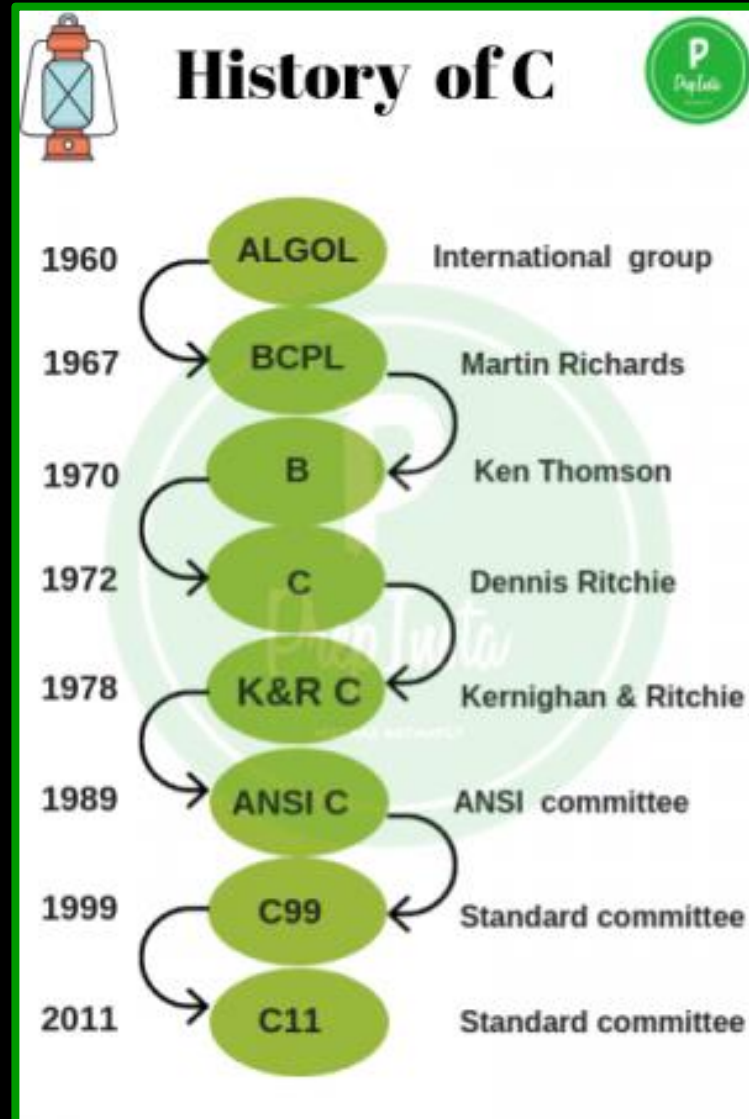
C is a programming language developed by AT & T's Laboratories f USA in 1972. It was designed and written by a men named Dennis Ritchie. It is use to design an operating system.

C has became popular because it is:-

- Simple
- Reliable
- Easy to use



Development of C/ History of C



Similarity between C language and English language

How to Learn C:

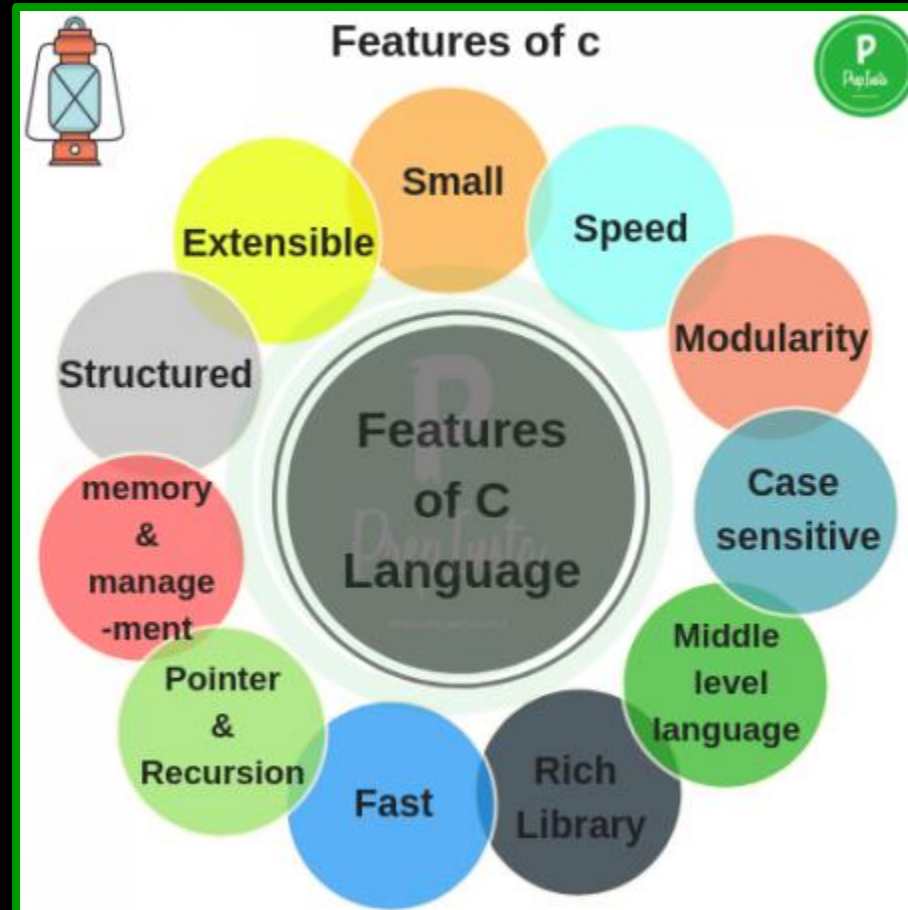
Steps of Learning English language



Steps of Learning C programming language

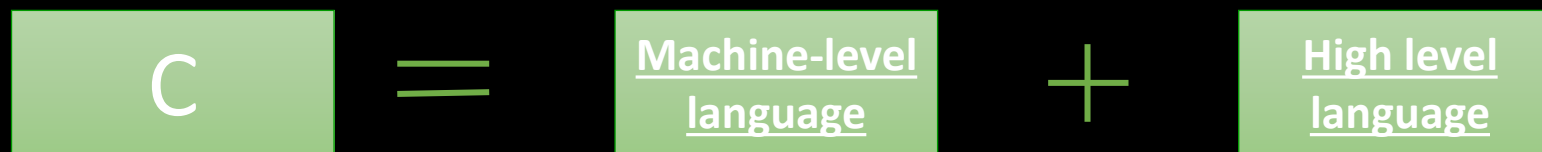


Features of C language



Why C is called as Middle Level Language?

C is called middle-level language because it actually binds the gap between a machine level language and high-level languages . A user can use c language to do System Programming (for writing operating system) as well as Application Programming (for generating menu driven customer billing system) .



Constant, Variables, Keywords, Instructions, Comments

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

Constant:- Constants in C are the fixed values that

are used in a program, and its value remains the same during the entire execution of the program. **Constants** are also called literals.

Variables:- A variable is nothing but a name given to a storage area that our programs can manipulate.

Keywords:- Keywords are predefined, reserved words in **C language** and each of which is associated with specific features. These words help us to use the functionality of **C language**. They have special meaning to the compilers. There are total 32 keywords in C.

Instructions:- C instructions are the commands in the program that instructs the compiler to do certain action.

Comments:- A comment is an explanation or description of the source code of the **program**. It helps a developer explain logic of the code and improves program readability. At run-time a comment is ignored by the compiler.

C-Program

* Format of C- program:-

#directive



Function Name

main()



Starting of program

{

.....

.....

}



Program Statement /
Instructions

First C- Program:-

```
#include<stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

Basic terms in C

- Printf() :- Use for printing output on the screen. Example:

```
printf("Hello World");
```

- Scanf() :- Use for taking input from the user. Example:-

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

'&' is an address operator.

- Data types :- A data type specifies the **type** of **data** that a variable can store. Example:

integer, floating, character, etc.

- **Identifiers** :- C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore.
- **Variables** :- A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- **Getch()** :- It is use to hold output screen after program execution until use enter any other key. To use getch() we have to use `#include<conio.h>` header file.

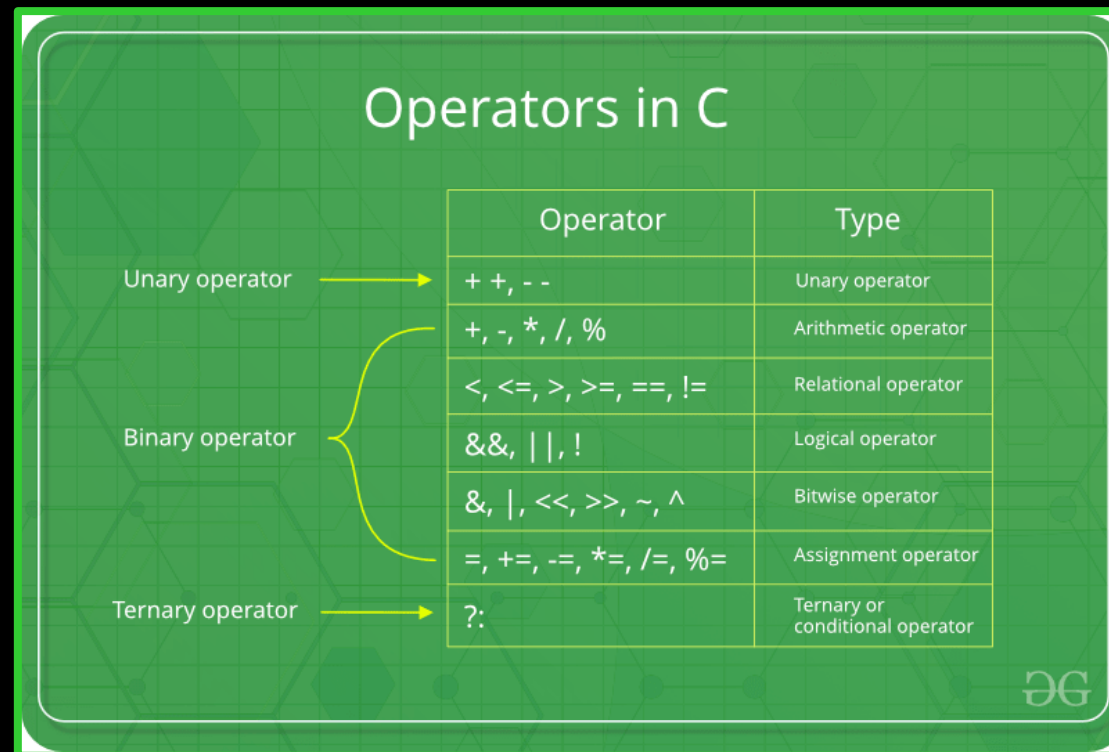
Operators

Operator:- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators.

Types of Operator:-

Operators in C

	Operator	Type
Unary operator	+ +, - -	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, , !	Logical operator
	&, , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator



GG

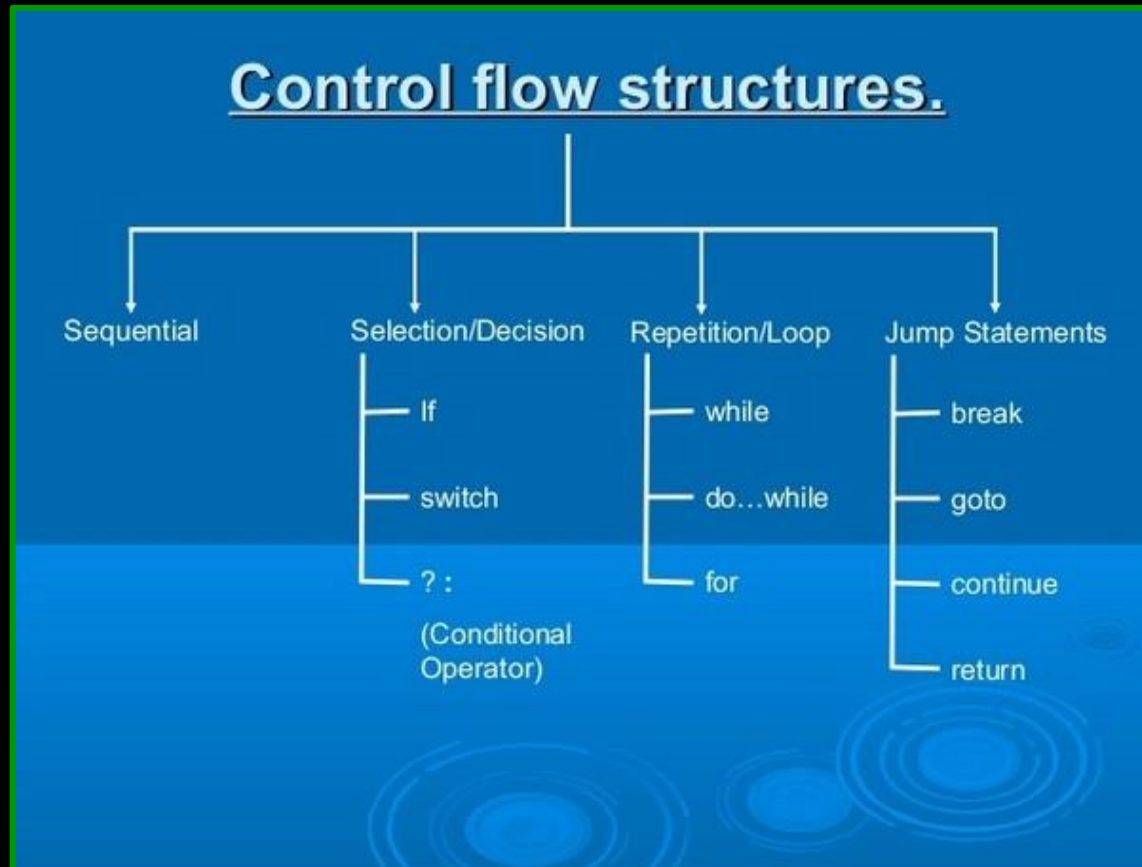
Statement :-

Command given to computer to perform a specific task.

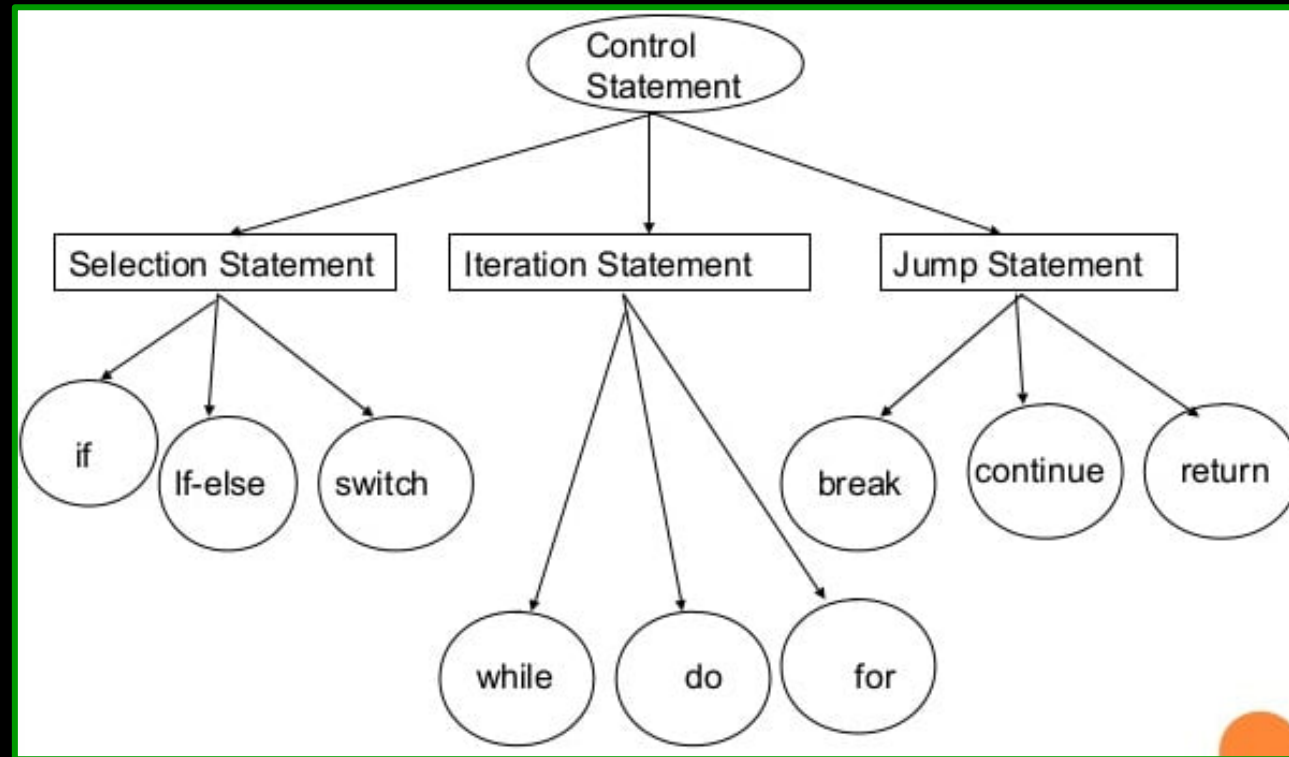
Types of statement :-

1. **Expression Statement:-** Simple statement consists of expression and semi-colon.
2. **Compound Statement:-** Combination of two or more simple statement connected by logic.
3. **Control Statement:-** Control the flow of statement and decide which statement is true or false.

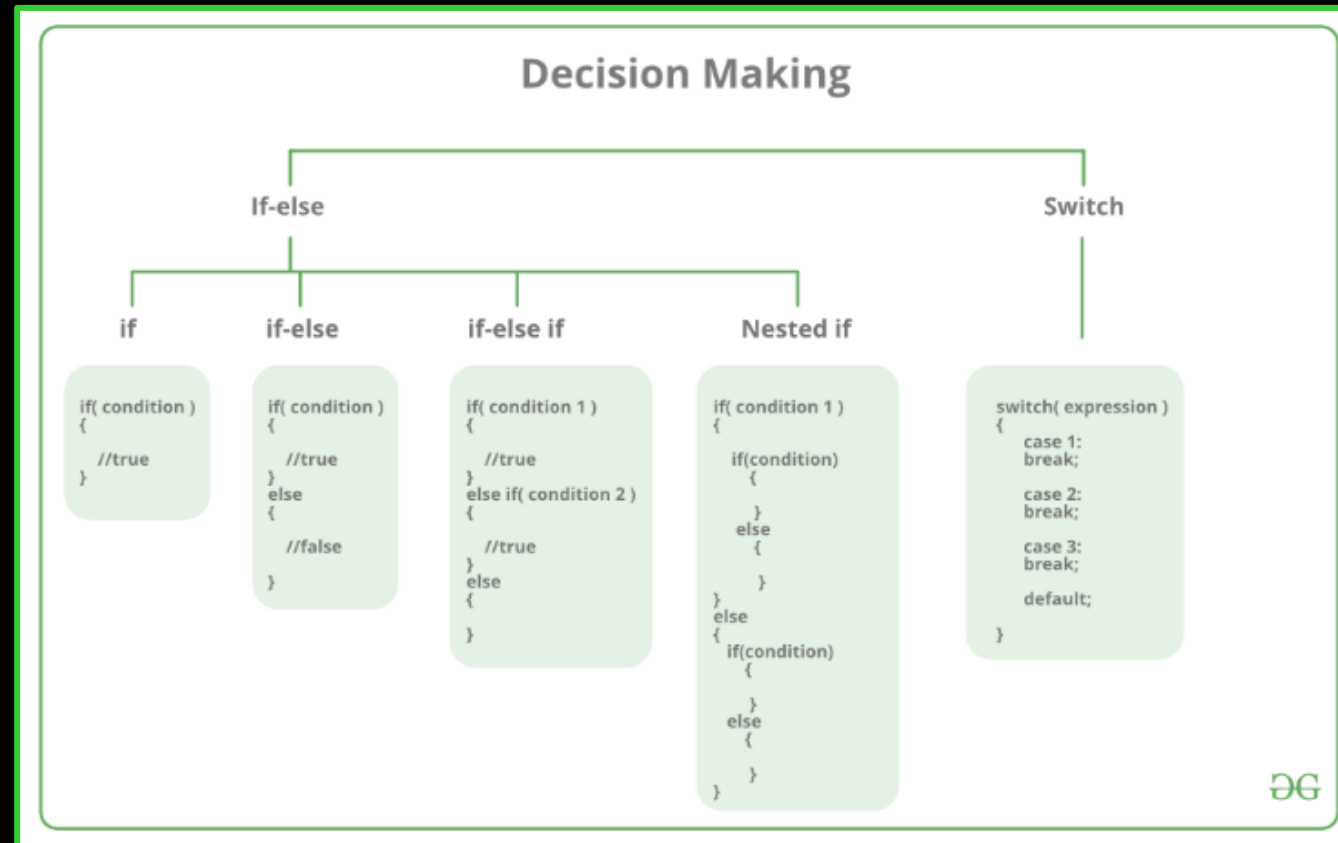
Types of control statement



Control Statement



Decision Making Statement



If statement

```
#include<stdio.h>
int main(){
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
return 0;
}
```

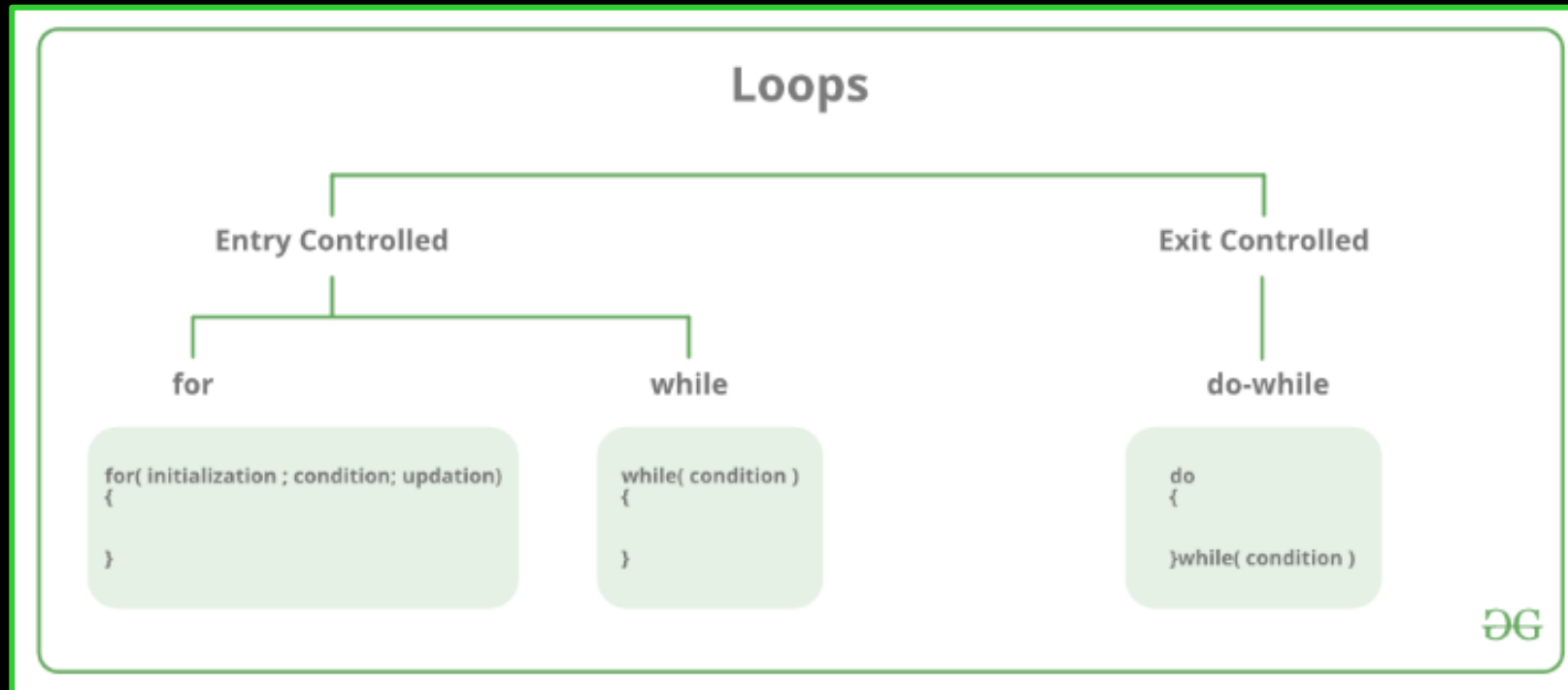
If-else Statement

```
#include <stdio.h>
int main()
{
    int age;
    printf("Enter your age?");
    scanf("%d", &age);
    if(age>=18)
    {
        printf("You are eligible to vote...");
    }
    else
    {
        printf("Sorry ... you can't vote");
    }
}
```

Switch

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d", &number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Iteration (Loop) Statement



While loop and do-while loop

While loop

```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```

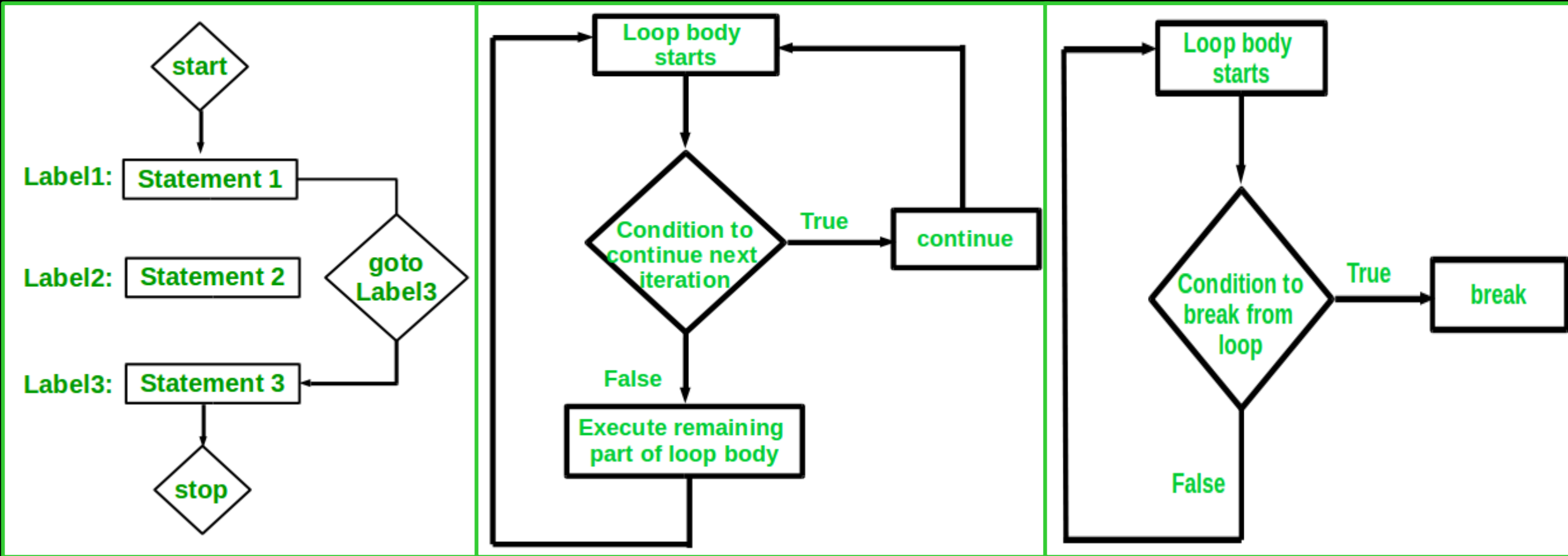
Do-while loop

```
#include<stdio.h>
int main(){
int i=1;
do{
printf("%d \n",i);
i++;
}while(i<=10);
return 0;
}
```

For loop

```
#include<stdio.h>
int main(){
int i=0;
for(i=1;i<=10;i++){
printf("%d \n",i);
}
return 0;
}
```

Jump Statement



goto statement

```
#include <stdio.h>
int main()
{
    int num , i = 1;
    printf("Enter the number whose table you want to print?");
    scanf("%d", &num);
    table:
    printf("%d x %d = %d\n", num , i, num * i);
    i++;
    if(i<=10)
        goto table;
}
```


Continue Statement

```
#include<stdio.h>
void main ()
{
    int i = 0;
    while(i!=10)
    {
        printf("%d", i);
        continue;
        i++;
    }
}
```

Break Statement

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
            break;
    }
    printf("came outside of loop i = %d",i);
}
```

Function

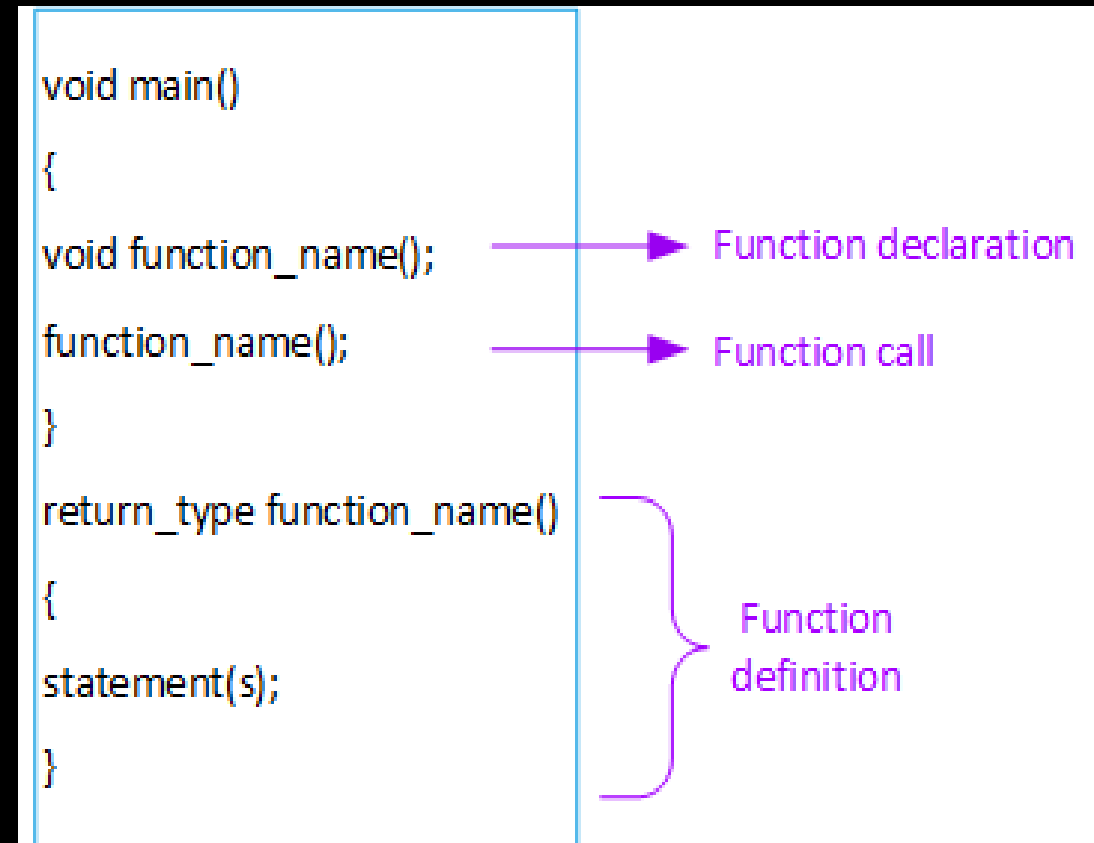
A function is a set of statements that together perform a specific task.

Syntax:-

```
return_type function_name (parameter)
{
body of function;
}
```

Steps of writing Function

1. Declaration of function
2. Calling of function
3. Define a function



Types of Function

1. **Library Function:-** Library function are those functions which are inbuilt in compiler, and user don't needs to create it. Example:-

```
sqrt();
```

2. **User define Function:-** User define functions are those functions which is created by user for according to their requirement, it is not inbuilt in compiler. Example:-

```
char akhilesh();
```

Passing Parameter in function can be done in two ways

Call by Value

```
#include <stdio.h>
void swap(int , int);
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b);
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b);
}
```

Call by Reference

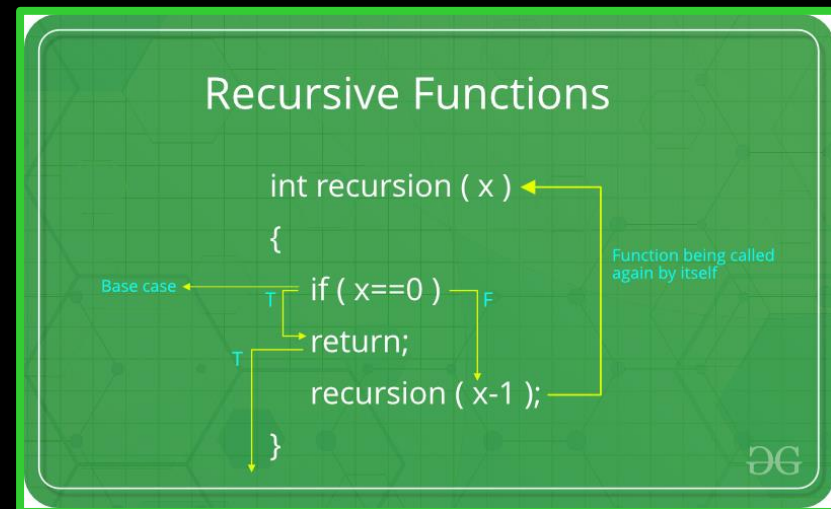
```
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);
    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(&x);
    printf("After function call x=%d \n", x);
return 0;
}
```

Difference between Call by value and Call by reference

Call By Value	Call By Reference
Here, actual value of variable are passed.	Instead of value reference of memory is passed.
Here, value of each variable is copied into corresponding another variable.	Here, address of actual parameters is copied.
Any change in formal parameter , does not reflect in the actual parameter.	If there is any change in parameter it will reflect every where.

Recursion

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function. The C programming language supports recursion, i.e., a function to call itself.

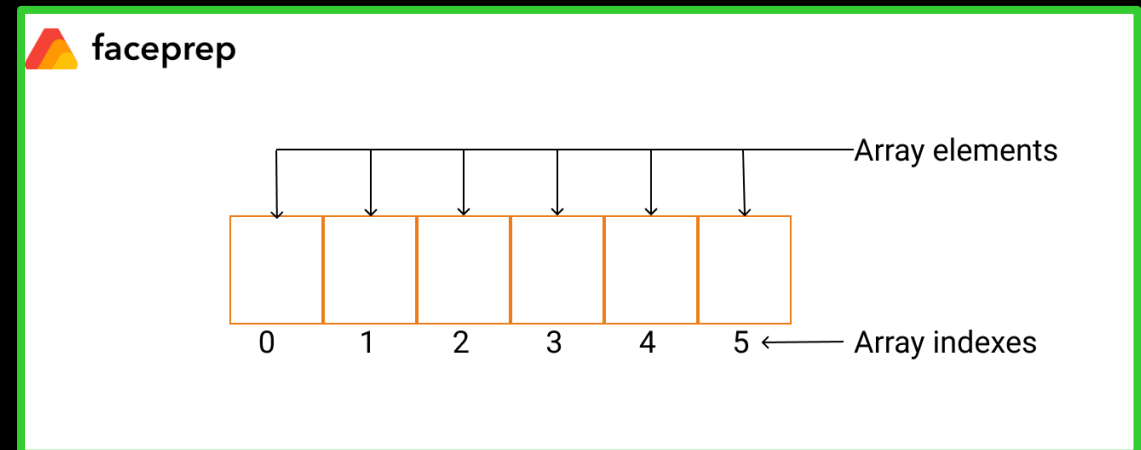


Example of Recursion Function

```
#include <stdio.h>
int fact (int);
int main() {
    int n, f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d" ,&n);
    f = fact(n);
    printf("factorial = %d", f);
}
int fact(int n) {
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1) {
        return 1;
    }
    else {
        return n*fact(n-1);
    }
}
```

Array

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.



Types of Array

1. One-Dimension Array

2. Two-Dimension Array

1. One-Dimension Array

A one-dimensional array (or single dimension array) is a type of linear **array**. Accessing its elements involves a **single** subscript which can either represent a row or column index.

- **Declaration of Array:-**

```
data_type array_name[array_size];
```

- **Initialization of Array:-**

```
data_type array_name[size] = {list of values};  
int num[5] = {1,2,3,4,5}
```

```
data_type array_name[ ] = {list of values};  
int num[ ] = {1,2,3,4,5}
```

- **Accessing Array Elements:-**

Subscript of array starts with 0.

int mydata[20]; -> total number of array.

mydata[0]; -> first element of array.

mydata[19]; -> last element of array.

Program of 1-D array

```
int main(){
    int arr[5], i;

    for(i = 0; i < 5; i++){
        printf("Enter a[%d]: ", i);
        scanf("%d", &arr[i]);
    }
    printf("\n Printing elements of the array: \n\n");
    for(i = 0; i < 5; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

2. Two-Dimension Array

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.

- **Declaration of Array:-**

```
data_type array_name[Row_Size][Column_Size];
```

- **Initialization of Array:-**

```
data_type array_name[Row_Size][Column_Size];
```

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

- **Accessing Array Elements:-**

An element in a 2-D array is access by using the subscripts, i.e., row index and column index.

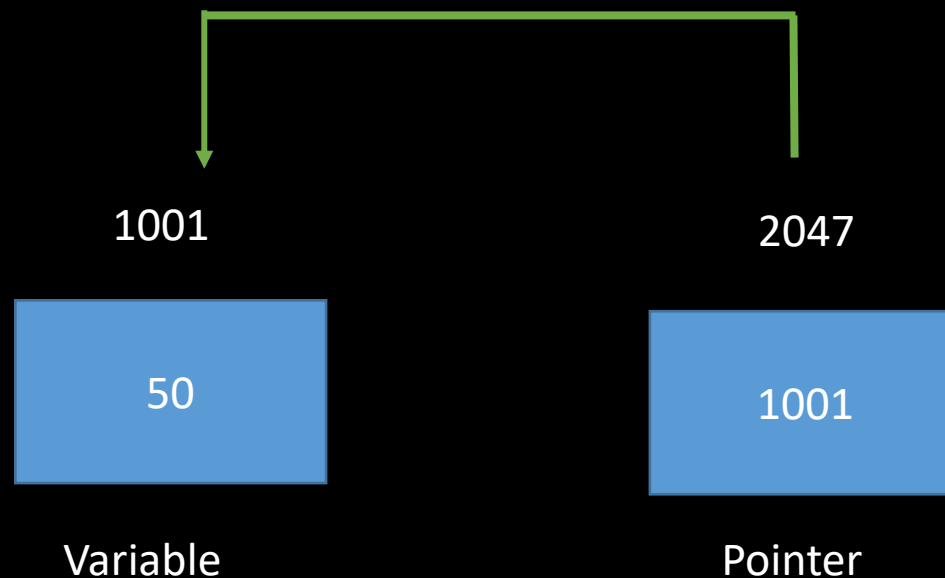
```
int value = a[2][3];
```


Program of 2-D array

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
for(i=0;i<4;i++){
for(j=0;j<3;j++){
printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
}
}
return 0;
}
```

Pointer

The Pointer in C, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.



- Declaration of Pointer Variable:-

```
data_type *pointer_variable;
```

```
int *p;
```

'*' Called asterisk sign. When use with variable name indicates that the variable is a pointer variable and it makes a differentiation between an ordinary variable and a pointer variable.

Pointer variable is a variable that holds the address of another variable.

- Initialization of Pointer:-

Address operator (&) is use to initialize a pointer variable. Example:-

```
int qty = 175;
```

```
int *p;    -> declaration
```

```
p = &qty; -> initialization
```

Variable	Value	Address
qty	175	5000
p	5000	5048

- **Indirection Operator:-**

The pointer operator, available in C is ‘*’ called ‘value at address operator’. It returns the value stored at a particular address. The ‘value at address operator’ is also called an ‘indirection operator’.

- **Indirection Operator:-**

The & operator used in this statement is C’s ‘address of operator’. The expression &i returns the address of variable i.

Dynamic Memory Allocation

The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

- malloc()
- calloc()
- realloc()
- free()

malloc() allocates single block of requested memory.

calloc() allocates multiple block of requested memory.

realloc() reallocates the memory occupied by malloc() or calloc() functions.

free() frees the dynamically allocated memory.

Malloc()

The malloc() function allocates single block of requested memory. It doesn't initialize memory at execution time, so it has garbage value initially. It returns NULL if memory is not sufficient.

Syntax :-

```
ptr = (cast-type*)malloc(byte-size)
```

Program on malloc()

```
int main(){
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));
    if(ptr==NULL) {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0; i<n; ++i){
        scanf("%d", ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d", sum);
    free(ptr);
return 0;
}
```


calloc()

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

Syntax :-

Ptr = (cast-type*)calloc(number, byte-size)

Program on calloc()

```
int main(){
int n,i,*ptr,sum=0;
printf("Enter number of elements: ");
scanf("%d",&n);
ptr=(int*)calloc(n,sizeof(int));
if(ptr==NULL){
printf("Sorry! unable to allocate memory");
exit(0);
}
printf("Enter elements of array: ");
for(i=0;i<n;++i){
scanf("%d",ptr+i);
sum+=*(ptr+i);
}
printf("Sum=%d",sum);
free(ptr);
return 0;
}
```

realloc()

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Syntax:-

```
ptr=realloc(ptr, new-size)
```

Program on realloc()

```
int main()
{
int* ptr;
int n, i;
n = 5;
printf("Enter number of elements: %d\n", n);
ptr = (int*)calloc(n, sizeof(int));
if (ptr == NULL) {
printf("Memory not allocated.\n");
exit(0);
}
else {
printf("Memory successfully allocated using calloc.\n");
for (i = 0; i < n; ++i) {
ptr[i] = i + 1;
}
}
```

```
printf("The elements of the array are: ");
for (i = 0; i < n; ++i) {
printf("%d, ", ptr[ i ]);
}
n = 10;
printf("\n\n Enter the new size of the array: %d\n", n);
ptr = realloc(ptr, n * sizeof(int));
printf("Memory successfully re-allocated using realloc.\n");
for (i = 5; i < n; ++i) {
ptr[i] = i + 1;
}
printf("The elements of the array are: ");
for (i = 0; i < n; ++i) {
printf("%d, ", ptr[i]);
}
free(ptr);
}
return 0;
```

free()

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Syntax:-

free(ptr)

Program on free()

```
int main() {
int *ptr, *ptr1;
int n, i;
n = 5;
printf("Enter number of elements: %d\n", n);
ptr = (int*)malloc(n * sizeof(int));
ptr1 = (int*)calloc(n, sizeof(int));
if (ptr == NULL || ptr1 == NULL) {
printf("Memory not allocated.\n");
exit(0);
}
else {
printf("Memory successfully allocated using malloc.\n");
free(ptr);
printf("Malloc Memory successfully freed.\n");
printf("\n Memory successfully allocated using calloc.\n");
free(ptr1);
printf("Calloc Memory successfully freed.\n");
}
return 0;
}
```

Array and Pointer

Arrays and pointers are synonymous in terms of how they use to access memory. But, the important difference between them is that, a **pointer** variable can take different addresses as value whereas, in case of array it is fixed.

Function and Pointer

In C, we can use **function pointers** to avoid code redundancy. For example a simple `qsort()` **function** can be used to sort arrays in ascending order or descending or by any other order in case of array of structures. Not only this, with function pointers and void pointers, it is possible to use `qsort` for any data type.

String

String is a character array terminated by null character (\0) or we can say, a string is a sequence of characters terminated with a null character \0. The difference between a character array and a string is the string is terminated with a special character '\0'.

- **Declaration of String:-**

```
char str_var[size];
```

```
char str_var[] = "SMSVARANASI";
```

- Initialization of strings:-

A string can be initialized in different ways. I will explain this with the help of an example.

1. `char str_var[] = "SMSVARANASI";`
2. `char str_var[40] = " SMSVARANASI ";`
3. `char str_var[] = {'S','M','S','V','A','R','A','N','A','S','I','\0'};`
4. `char str_var[12] = {'S','M','S','V','A','R','A','N','A','S','I','\0'};`

- Following is the memory presentation of the string in C language:-

	str_value[0]	str_value[1]	str_value[2]	str_value[3]	str_value[4]	str_value[5]	str_value[6]	str_value[7]	str_value[8]	str_value[9]	str_value[10]	str_value[11]
Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	S	M	S	V	A	R	A	N	A	S	I	\0
Address	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016

String Library Function

- ***strlen()***: This function returns the length of the string. Example: ***strlen(name)***; This will return the length of the string stored in the variable *name[]*.
- ***strcat()***: This function concatenates two strings. Example: ***strcat(name,name1)***; This will concatenate the strings stored in the variables *name[]* and *name1[]* in the order in which it is written.
- ***strcpy()***: This function copies the value of the second string to the first string. Example: ***strcpy(name1,name)***; This will copy the string in *name[]* to the variable *name1[]*.
- ***strcmp()***: It compares two strings. Example: ***strcmp(name,name1)***; This compares the string in *name[]* with the string in *name1[]*. It returns 0 if the strings are same. It returns a value less than 0 if *name[]<name1[]*. Otherwise, it returns a value greater than 0.

- ***strlwr()***: It changes all the characters of the string to lower case. Example: ***strlwr(name)***; It shall convert the whole string stored in the variable *name[]* to lowercase.
- ***strupr()***: It changes all the characters of the string to upper case. Example: ***strupr(name)***; It shall convert the whole string stored in the variable *name[]* to uppercase.
- ***strchr()***: It returns the location or the pointer of the first occurrence of a character in a string. Example: ***strchr(name,ch)***; It returns the location of the first occurrence of the character in *ch* in the string *name[]*. It returns null if the character is not found.
- ***strstr()***: It returns the location or the pointer of the first occurrence of one string in another. Example: ***strstr(name,name1)***; It returns the location of the first occurrence of *name1[]* in *name[]*. It returns null if the string is not found.
- These library functions are defined in the header file **<string.h>**.

Structure

In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name. Or To group variables of different types with a data type called structure. Structure is an example of heterogeneous data type.

Let's take an example to understand the need of structure: Suppose we need to store the data of students like student name, age, address, roll no etc. One way of doing this would be creating a different variable for each attribute, however when we need to store the data of multiple students then in that case, we would need to create these several variables again for each student. This is such a big headache to store data in this way. In such situations we can use structure

- **Declaration of Structure:-**

structured data type.

```
struct struct_name {  
    DataType member1_name;  
    DataType member2_name;  
    DataType member3_name;  
    ...  
};
```

Example:

```
struct StudentData{  
    char stu_name[50];  
    int stu_roll;  
    int stu_age;  
};
```

- How to declare variable of a structure?

```
struct struct_name var_name;
```

```
or struct struct_name {
```

```
    DataType member1_name;
```

```
    DataType member2_name;
```

```
    DataType member3_name;
```

```
    ...
```

```
} var_name;
```

Example:

```
struct StudentData{
```

```
    char stu_name[50];
```

```
    int stu_roll;
```

```
    int stu_age;
```

```
};
```


- How to access data members of a structure using a struct variable?

Using Dot(.) operator

```
var_name.member1_name;
```

```
var_name.member2_name;
```

```
...
```

Example:

```
printf ("%s\n", stud.stu_name);
```

```
printf ("%d\n", stud.stu_roll);
```

```
printf ("%d\n", stud.age)
```

• How to assign values to structure members?

1. Using Dot(.) operator

```
var_name.member_name = value;
```

Example:

```
stud.stu_name = "Hello";
```

```
stud.stu_roll = 20;
```

```
stud.age = 23;
```

2. All members assigned in one statement

```
struct struct_name var_name =
```

```
{value for member1, value for member2 ...so on for all the members};
```

Example:

```
struct StudentData stud = {"Hello", 20, 23};
```

Union

A union is block of memory that is used to hold data items of different types. In C, a union is similar to a structure, except that data items saved in the union are overlaid in order to share the same memory location.

Or

A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose

- Declaration of Union:-

```
union Data
```

```
{
```

```
int i;
```

```
float f;
```

```
char str[20];
```

```
};
```

Program on Union:-

```
#include union Data {
```

```
int i;
```

```
float f;
```

```
char str[20];
```

```
};
```

```
int main( ) {
```

```
union Data data ;
```

```
printf( "Memory size occupied by data : %d\n", sizeof(data));
```

```
return 0;
```

```
}
```

This is a brief introduction of C programming language...

For more content on C you can refer

<https://www.w3schools.in/c-tutorial/>

Thank You RGG Sir for your support...

