(ARRAYS)

- Single unit of multiple values.
- A fixed size collection of items with same datatype.
- Homogenous collection.

Storage :

Array is stored sequentially in memory.

→ Contiguous Memory :

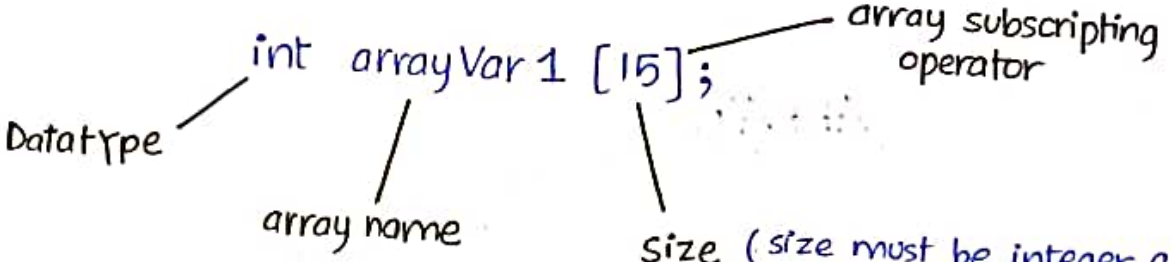
- a chunk of memory allocated without any gaps in the addresses it occupies.
- One single unbroken "block" of memory.

What if memory is non-contiguous :-

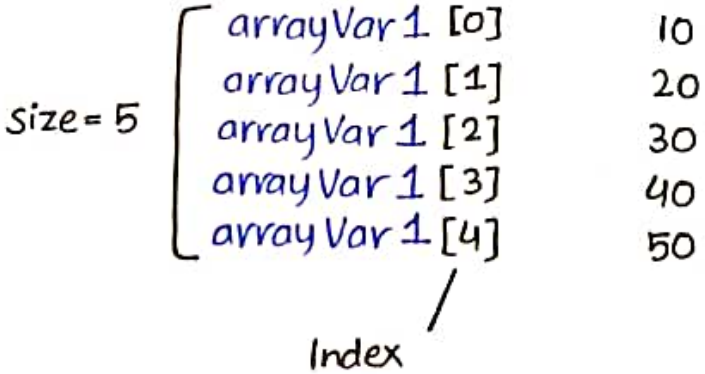
An array is ~~non-~~contiguous as it is an indexed based data structure.
In absence of contiguous space available in memory, program will fail to create an array.
An error will be thrown.

Declaration :

Datatype Arrayname [int - expression]



Memory View :



- More than one array can be declared on a line.

```
int student[10], Faculty[30], Player[20];
```

- Arrays can be declared alongwith the declaration of variables.

```
int var1, var2, marks[8];
```

Initializers :

Elements of array.

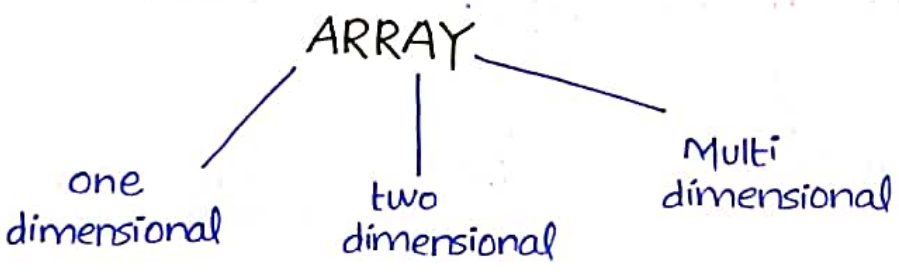
```
int n[5] = {1,2,3,4,5};
           initializers
```

if size is not provided during declaration, initializers will determine the size.

Limitations of arrays :

- We must know how many elements are to be stored in array
- Array is static structure with fixed size, so the memory allocated to array cannot be changed.
- As the memory is contiguous, insertion and deletion is difficult and time consuming.

Types of arrays :



(2D-ARRAYS)

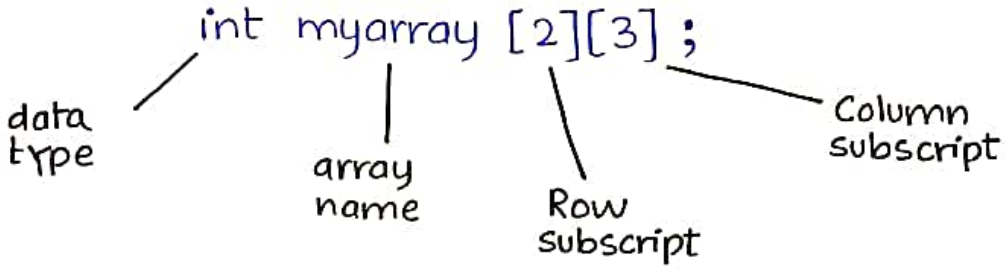
- Arrays with 2 or more dimensions.
- Correspond to matrices.

1D arrays → List
2D arrays → Table

Definition:

- A fixed size collection of items of same data type arranged in two dimensions.
- 2D arrays can be considered as a table with rows and columns.
- Each element of 2D array is referred to with help of 2 indexes. 1 index indicates row and 2 index indicates column.

Declaration:



→ int myArray [3][2] = {{12,15},{22,25},{32,35}}

	column 0	column 1
Row 0	12	15
Row 1	22	25
Row 2	32	35

(FUNCTIONS)

- Problem solving approach for complex / large problems.
- Basic Rule → Divide and Conquer
  ↓
  division of large problem into smaller parts and then solving them

Definition :-

A function groups a number of program statement into a unit and gives it a name.

- Dividing a program into functions is one of the major principle of structured programming.
- Every C++ program has atleast one function, that is main()

Advantages :-

Using functions in a program is very beneficial and is considered a good approach.
Following are listed some advantages of functions :

- Avoid repetition of same code in program.
- Improve readibility of program.
- Make program modification easier.
- Debugging becomes easier.
- Multiple persons can work on the program (on seprate functions).

Types :

2 types of functions in C++.

pre-defined function :-

Functions that C++ is alreddy providing to users through different libraries.
pre-defined functions are organized into seprate libraries.

- For input and output functions, iostream is available.
- For all math functions, cmath is available.

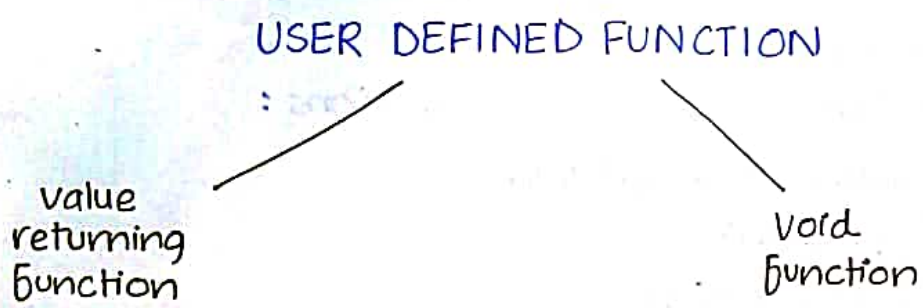
examples are :

- clrscr ( ) – clear screen
- pow ( ) – power
- sqrt (25) – square root

user-defined function :-

Functions that are created by users according to the requirements.

e.g. • To find sum of 2 numbers
• Calculate income tax.



- value returning function :

  These functions have a return type. They must return some value of a specific data type.

- void function :

  These functions don't have any return type. They dont use return statement to return a value.

USER DEFINED FUNCTIONS

Functions created by user according to requirements.

components :

- Function declaration (prototype)
- Function definition
- Function call.

syntax of components :

Declaration

```
return_type function_name (parameters);
```

Call

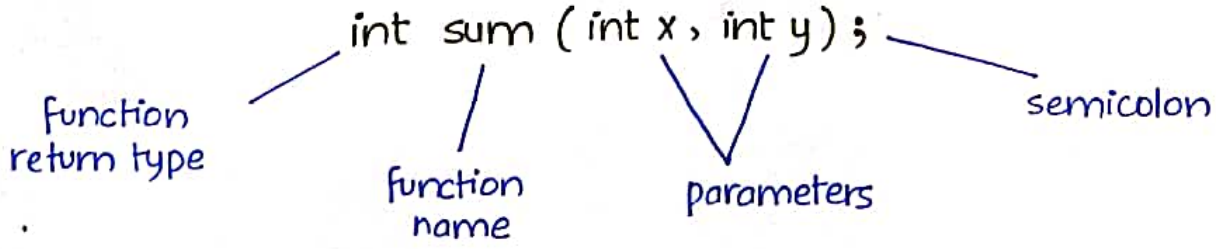
```
function_name (parameters);
```

Definition

```
return_type function_name (parameters){
    statements;
}
```

Declaration :-

- Function declaration provides information to compiler by telling its return type, parameters/datatypes and function name.
- Declaration is followed by a semicolon (;)
- Declaration is required when a function is defined below or after its call. In this case, function should be declared before call.



Definition :-

- Function body is called its definition.
- It tells the actual functionality and logic of program.
- It actually informs what the function will do in program.

⇓ imp

If function is defined before its call, then its definition also serves as its declaration, so a prototype (seprate) is unnecessary.
But if function is defined after its call, then prototype is necessary before call.

```
int sum (int x , int y) → also acts as declarator
{
statements ; → body of function.
}
```

Return type → Return type is the data type of value that function returns.

Parameters → It is just like a placeholder. We pass actual values to these parameters during function call.
Those actual values are called as "arguments"

Call :-

In C++, you give a definition of what a function has to do, and then to use that function, you will have to call that function
In function call, you have to pass the actual values (arguments) instead of the parameters used in definition.

```
sum (arguments) ;
```

- sum → function name
- (arguments) → arguments to be passed
- ; → semicolon.

(POINTERS)

pointer is a variable that holds the memory address of another variable

Normal Variable => use to store values of their type. (direct reference)
pointers => use to store address of variable. (indirect reference)

Memory address :-

location of another variable where it has been stored in the memory.

- pointer holds the address of a variable, we can say that it points to that variable.

Declaration :

Must be declared before their usage.

datatype* variable_name;

- int*x (pointer to int)
- float *ptr (pointer to float)
- char*z (pointer to char)

Address operator (&)

- Address operator (&) is used to produce an address of variable by operating on variable name.
- If we want to refer to address of a particular variable, we can use the address operator.

var1
&var1 → This will give the address

Dereferencing operator (*)

- unary operator that operates function on pointer variable.
- We can access the value stored in the variable through dereferencing operator.
- It returns the contents of variable located at specific address.

float* ptr;

const pointer / non constont data :

pointer itself is constant, but the value is pointing to is non-constcer

const. pointer :- pointer is pointing towards a specific variable. After becoming constant, the variable to which it is pointing cont' be changed.

```
int x = 5;
int *const ptr = &x; ✓

ptr = &y;   X
```

Non-constant data :- As data is non-constant, so the value of the variable pointer is pointing to can be changed.

```
int x = 5;
int*const ptr = &x;
*ptr= 7;
```
✓

const pointer / const data :

pointer and the variable it is pointing to, both are constant

```
int value = 5;
const int *const ptr = & value;
```

Noither the address of pointer can get change nor the value it is pointing to.

non-constant pointer / constant data :

pointer itself is not constant but the value it is pointing to is constant and fixed.

Non-constant pointer : As the pointer is non-constant, it can point to multiple addresses. We can change the address.

```
const int x = 5;
const int* ptr = &x;

const int y = 6;
const ptr = &y;
```
✓

const · data : data is constant, its value cannot be changed

```
int x = 5;
const int * ptr = &x;
*ptr = 6; ] ×
x = 6; ] ✓
```

we cant' change the value through pointer.

(RECAP)

```
int x = 5;
```

1 : `int * ptr1 = &x;`
(non-constant pointer and non-constant data)

2 : `const int * ptr2 = &x;`
(non-constant pointer and constant data)

3 : `int * constant ptr3 = &x;`
(constant pointer and non-constant data)

4 : `const int * const ptr4 = &x;`
(constant pointer and constant data)

POINTERS TO VOID

General purpose pointers that can point to any datatype

```
void* ptr ;
```

POINTERS IN FUNCTIONS

3 methods for passing parameters in functions

- Pass by value
- Pass by reference
- Pointers

Definition :-

```
void function_name (datatype* ptrvar) ;
```

Call :-

```
function_name (& var_name) ;
```

Static Memory Allocation

- Also called as Compile time allocation
- size and location where variable will be ~~fixed~~ stored is fixed during compile time.
- Both allocation and dellocation of memory is done by compiler itself.
- slightly faster
- memory allocation on stack.

Dynamic Memory Allocation.

- Also called as Run time allocation
- Memory requirement should be ~~req~~ defined during the execution of program.
- Programmer needs to write a proper code for allocation and deallocation.
- slightly slower.
- memory allocation on heap.

DMA

New operator

- used to allocate memory to a variable.
- pointer is used to allocate memory dynamically.
- pointervariable = new datatype;

Delete operator

used to deallocate the memory occupied by variable.

pointer is used to deallocate the memory.

delete pointervariable;

LOCAL VARIABLE

Variable defined inside a function body.

Its scope is limited to the function where it is defined.

Its lifetime ends when the function exits.

GLOBAL VARIABLE

Variable defined globally, outside all functions.

Its scope is the whole program.

Its life ends only when the program ends.

STATIC LOCAL

Also defined inside a function.

Its scope is in whole program from where it is initialized.

Its life ends with the ending of program.