07/09/2023
Thursday

* **ARRAY** - Array is a data structure or a collection of elements of similar data types stored in continuous memory blocks.

for instance, only integer values = -1, -2, 0, 1, 2
only characters = 'A', 'B', 'a', 'b'
only float type values = 1.00, 1.414, 3.14

* **Why Arrays ?**

Suppose we need to add 5000 numbers at a time. We will not create 5000 variables here. We will just create an array defining its size. The size of array we have created, the equal number of continuous memory blocks are formed in the memory where these 5000 numbers will be stored.

Continuous memory block means where the first block ends, the next memory block gets started from that place and so on.

This is how an array is created in the memory.

__/__/__

* **Syntax of Array –**

return_type array_name [size];

Example, int arr[5];
char ch [10];
float brr [15];

int arr [10];
When we create an array of size 10, 10 memory blocks are formed. Here, the data type is int so, 1 block will be of 4 bytes and 10 blocks will be of 10 × 4 = 40 Bytes.

* **Static Array Creation** – When we define the size of array during array creation.

Example – int arr [20];
This is static array creation where we have defined that the size of array will be 20.

* **ADDRESS →**

Example → int arr [10];
When we write like this, none of the block is known by arr. They all are known by their addresses which is find by the compiler using symbol table.

If we want to know the address of a memory location, we use address of operator for that.

Address of operator = &

for example,

```
int main( ) {
    int a = 5;
    cout << " Address of a : " << &a << endl;

    int arr[10];
    cout << " Base address of array: " << &arr << endl;
}
```

NOTE - If we will not use address of operator symbol, then the address will be same also as before.

like, cout << arr << endl;

This line will also print the address of array.

* Base Address → Base Address is the address of the first block of array.

It is also known as starting address.

```
int arr[5];
```

Base Address

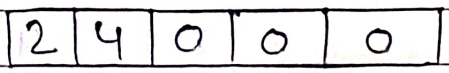
__/__/____

* **ARRAY INITIALISATION –**

Case 1 → `int arr[] = {1,2,3,4,5};`

Case 2 → `int brr[5] = {1,2,3,4,5};`

Case 3 → `int crr[5] = {2,4};`



The empty blocks will be filled by 0 as size of array is 5 and elements are only 2.

Case 4 → `int drr[2] = {1,2,3,4,5};`

Here, the size of array is less and elements are more. So, this will show error.

* **Taking size of array through user input →**

```
int n;
cin>>n;
int arr[n];
```

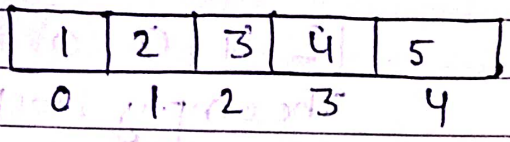
} This is a bad practice.

Our operating system (OS) allocates some specific memory to every program. Suppose, it allocates 10 bytes to a program. If we give a such input for n which requires 100 bytes, then it will throw error.
But in some compilers, it will not show error. So, remember not to take user input for size of array. To prevent this case, we use dynamic memory allocation.

* INDEXING IN ARRAYS →

If we want to access any memory block of an array, we use indexing here.
0-based indexing is done in arrays.

for example, int arr[5];



Now, we can access elements of array by their indexes as arr[0], arr[1], arr[2], arr[3], arr[4]: Whatever stored in that particular index or block will be the output.

Code –

```
int main(){

  int arr[5] = {1,2,3,4,5};
  int n=5;
  for (int i=0; i<n; i++){
        cout<<arr[i]<<" ";
  }
```

Output = 1 2 3 4 5

__/__/____

* Program to take input in an array & printing it.

```
int main(){

int arr[5];
int n=5; // Taking input in an array
for (int i=0 ; i<n ; i++){
   cout<< "enter the value of index:" << i <<endl;
   cin>> arr[i];
   cout <<endl;
   }
   // Printing an array
cout<< " printing an array"<<endl;

for (int i=0; i<n; i++){
    cout << arr[i] << " ";
   }
}
```

Output ⇒

```
Enter the value of index: 0   10
Enter the value of index: 1   20
Enter the value of index: 2   30
Enter the value of index: 3   40
Enter the value of index: 4   50

Printing the array

10  20  30  40  50
```

__/__/___

* Formula for finding value at any index in an array by calculating its address →

Value of arr[i] is at address =

$$\left(\text{Base Address} + \left(\frac{\text{data\_type}}{\text{size}} * \text{index}\right)\right)$$

Let an array,

Base Address

Let Address → 104 108 112 116 120

1	2	3	4	5

Data_type = int

Index → 0 1 2 3 4

we have to find value of arr[3] →

$$= \left(104 + \left(4 * 3\right)\right) = \left(104 + 12\right)$$

$$= 116$$

The value of arr[3] is at address = 116 which is 4.

__/__/__

* **Program to print double of an array through user input →**

```
int main(){
  int arr[10];
  int n=10;
  for(int i=0; i<n; i++){
      cin>> arr[i];
    }
  cout<< "printing an array"<<endl;
  cout<< "doubles"<<endl;
  for(int i=0; i<n; i++){
      cout<< 2* arr[i]<<endl;
    }
  }
```

Output →

```
1 2 3 4 5 6 7 8 9 10
printing an array
doubles
2 4 6 8 10 12 14 16 18 20
```

* **LINEAR SEARCH IN AN ARRAY →** Linear Search is done to check whether the given or targeted element is present in the given array or not.

For example, Input array =

2	4	6	8	10	12
0	1	2	3	4	5

Input target = 10

Output = 10 found / 10 not found

__/__/__

* Code for linear Search in an Array -

```
int main() {

 int arr[5]= {2,4,6,8,10,12};
 int target = 10;
 int n= 5;

 // 0 -> not found
 // 1 -> found
  bool flag = 0;

 for(int i=0; i<n ; i++){
       if(arr[i]==target){
            //found
             flag=1;
             break;
            }
          }
    if (flag==1){
       cout<< "target found"<< endl;
      }
    else
       cout<< "target not found"<<endl;

  return 0;
}
```

Output → Target found.

__/__/____

* **ARRAYS & FUNCTIONS →**

Pass and call an array in functions →

```
#include <iostream>
using namespace std;

void Array(int arr[], int size){

    cout << "Arrays & functions" << endl;
}

int main(){
    int arr[5] = {2,4,6,8,10};
    int size = 5;
    Array(arr, size);

    return 0;
}
```

Output → Array & functions

NOTE → Whenever passing an array in a function, always remember to send the size with it.
Size, here doesn't mean how many memory blocks present in array.
Size, here means the number of elements present in an array.

__/__/____

* Program to print an array using functions →

```
#include <iostream>
using namespace std;

void printArray(int arr[], int size){
  int n=5;
  for(int i=0; i<n; i++){
    cout<< arr[i] <<" ";
  }
}

int main(){

  int arr[5]={2,4,6,8,10};
  int size = 5;
  printArray(arr, size);

  return 0;
}
```

Output → 2 4 6 8 10

* Program for linear Search in an array using functions →

```
#include <iostream>
using namespace std;

bool linearSearch (int arr[], int size, int target){

    for (int i=0; i< size; i++){
        if (arr[i] == target){
            return true;
        }
    }
    return false;
}

int main( ){
    int arr[5]= {2,4,6,8,10};
    int size = 5;
    int target= 8;

    bool ans = linearSearch (arr, size, target);

    if (ans == true){
        cout << " target found "<< endl;
    }
    else
        cout <<"target not found "<< endl;
    }
```

Output → Target found

* Count 0's and 1's in an array →

```
#include <iostream>
using namespace std;

voit  countZeroOne (int arr[], int size){
    int zeroCount = 0;
    int oneCount = 0;

    for (int i=0 ; i<size ; i++){
        if (arr[i] == 0){
            zeroCount++;
        }
        else if (arr[i]==1){
            oneCount++;
        }
    }
    cout << "zero count: " << zeroCount << endl;
    cout << " One count: " << oneCount << endl;
}
int main(){
    int arr[] = {0,0,0,1,1,1,1,1,0,1};
    int size = 10;
    countZeroOne (arr, size);

    return 0;
}
```

Output → zero count : 4
One count : 6

__/__/__

* **Minimum and maximum number in an array →**

#include <limits.h>

This header file contains → INT_MIN
INT_MAX

These are used to find minimum and maximum values from many. So, this header file contains the meaning of it.

for eg- Range of unsigned int →

$-2^{31} \rightarrow 2^{31} - 1$

INT_MIN ← ⟶ INT_MAX

= **Best Practice →**

For Minimum number, always initialise the variable with INT_MAX.

```
int minAns = INT_MAX;
```

Because INT_MAX is the largest possible number which int data type can contain and no other number in the array will be greater than it. So, in this case comparision would be easy.

For Maximum number, always initialise the variable with INT_MIN.

```
int maxAns = INT_MIN;
```

Because INT_MIN is the smallest possible number which int data type can contain and no other number in the array will be smaller than it. So, in this case comparison would be easy.

__/__/____

Code -

```
#include<iostream>
#include<limits.h>
using namespace std;
int main(){
cout<<INT_MIN<<endl;
cout<<INT_MAX<<endl;
}
```

Output →
-2147483648 → -2^{31}
2147483647 → $2^{31}-1$

* Find minimum number in an array -

```
#include<iostream>
#include<limits.h>
using namespace std;

int minInArray(int arr[], int size){
    int minAns = INT_MAX;
    for(int i=0; i<size; i++){
        minAns = min(arr[i], minAns);
    }
    return minAns;
}
int main(){
int arr[] = {24,30,-59,68,70,-3,69,10,19,5};
int size = 10;
int minimum = minInArray(arr, size);
cout<<"minimum is: "<<minimum<<endl;
return 0;
}
```

Output → minimum is: -59

__/__/____

* Find maximum number in an Array →

```
#include<iostream>
#include<limits.h>
using namespace std;

int maxInArray (int arr[], int size){
    int maxAns = INT_MIN;
    for (int i=0; i<size ; i++){
        if (arr[i] > maxAns){
            maxAns = arr[i];
        }
    }
    return maxAns;
}
int main( ){
    int arr[] = {2,5,1,7,-3,5,67,89,4,100};
    int size = 10;
    int maximum= maxInArray (arr, size);
    cout <<"maximum is: "<< maximum <<endl;
    return 0;
}
```

Output → maximum is: 100

__/__/__

* ## WHILE LOOP →

Syntax –

```
initialisation
while (condition)
{
    // loop body
    updation ++;
}
```

Program–

```
#include <iostream>
using namespace std;

int main( ) {
  int i=1 ;
  int n;
  cin>>n;
  while (i<=n){
     cout << i<<" ";
     i++;
  }
  cout<< endl;
  return 0;
}
```

Output → 5

1 2 3 4 5

* **NESTED WHILE LOOP →**

Syntax →

```
initialisation1;
while (condition1){
    //statement
    updation1++;
initialisation2;
while (condition2){
    //statement
    updation2++;
}
}
```

Program →

```
int main( ){
 int i= 0;
while (i<2){
cout<<endl<<"outer loop"<<i<<endl<<endl;
 i++;
 int J=0;
 while (J<2){
  cout<< "inner loop"<< J<<endl;
   J++;
 }
}
}
```

Output →

```
Outer loop 0
inner loop 0
inner loop 1

Outer loop 1
inner loop 0
inner loop 1
```

* **Print the reverse of an array →**

```
#include <iostream>
using namespace std;

void returnArray(int arr[], int size){
    int left = 0;
    int right = size-1;

    while (left <= right){
        swap(arr[left], arr[right]);
        left ++;
        right --;
    }

    // Printing the reverse
    for (int i=0; i<size; i++){
        cout << arr[i] << " ";
    }
}
int main(){
    int arr[] = {10,20,30,40,50,60,70,80};
    int size = 8;
    returnArray(arr, size);

}
```

Output → 80 70 60 50 40 30 20 10

__/__/____

* Print extreme in an array →

```
#include <iostream>
using namespace std;

void extremeArray (int arr [ ], int size) {
    int left = 0;
    int right = size-1;
    while (left <= right) {
        if (left == right) {
        cout << arr [left] << " ";
        }
        else {
        cout << arr [left] << " " << arr [right] << " ";
        }
        left ++;
        right --;
    }
}
int main ( ) {
    int arr [ ] = {10, 20, 30, 40, 50, 60, 70};
    int size = 7;
    extremeArray (arr, size);

}
```

Output → 10 70 20 60 30 50 40