Pointers (class 2)

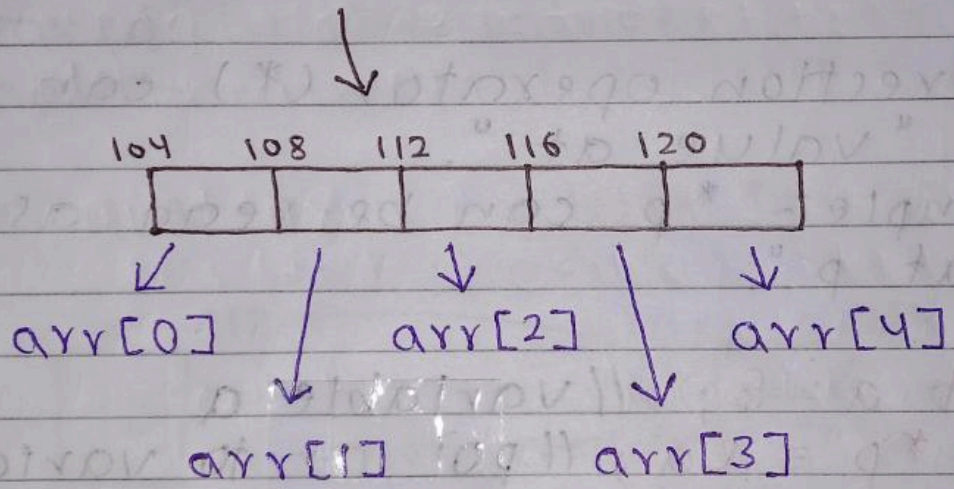
04-03-2023

Date.....

* Pointers to 1-D Array :-

The elements of an array are stored in contigous memory locations. Suppose, we have an array of type int,

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



Here, 104 is the address of first element & since each element (type int) takes 4 bytes, address of next element is 108, and so on.
The address of first element of the array is also known as the base addr-ess of the array. Thus, it is clear that the elements of an array are stored sequentially in memory one after another.

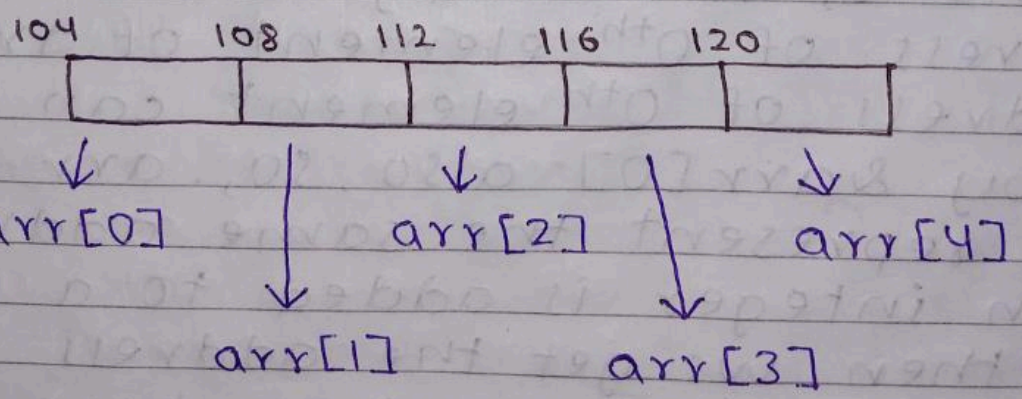
Date.....

In C++, pointers & arrays are closely related. we can access the array elements using pointer expressions. Following are the main points for understanding the relationship of pointers with arrays:-

i) Elements of an array are stored in consecutive memory locations.

ii) when the name of an array is used in any expression, the value of the name is a constant pointer that denotes the address of the first element of the array.
Example:- array name "arr" in an expression is equivalent to "&arr[0]".

iii) According to pointer arithmetic, when a pointer variable is increm-ented, it points to the next locati-on of its base type.



we can get the address of an element of array by applying '&' operator in front of subscripted variable name.

Date.....

Hence, &arr[0] gives the address of 0th element, &arr[1] gives the address of 1st element & so on.
Since array subscripts start from 0, we'll refer to the first element of array as 0th element & so on.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int arr[] = {10, 15, 20, 25, 30};
    for(int i = 0; i < 5; i++)
    {
        cout << " value : " << arr[i];
        cout << " Address : " << &arr[i];
    }
    return 0;
}
```

The name of the array arr denotes the address of 0th element of array. The address of 0th element can be given by &arr[0] also. So, arr and &arr[0] represent the same address. when an integer is added to a pointer then we get the address of next element of same base type. Hence, arr+1 will denote the address of the next element arr[1]

Date.....

arr → 0th element → &arr[0]
arr + 1 → 1st element → &arr[1]
arr + 2 → 2nd element → &arr[2]
arr + 3 → 3rd element → &arr[3]
arr + 4 → 4th element → &arr[4]

code :-

```
int main()
{
    int arr[4] = {12, 44, 66, 88};
    cout << *arr ;      // O/p: 12
    cout << arr[0];     // O/p: 12
    cout << *arr + 1;   // O/p: 13
    cout << *(arr) + 1; // O/p: 13
    cout << *(arr + 1); // O/p: 44
    cout << *(arr + 2); // O/p: 66
    cout << arr[2];     // O/p: 66
    cout << *(arr + 3); // O/p: 88
    cout << arr[3];     // O/p: 88
    return 0;
}
```

*(arr + i) *(i + arr)

Note:-

arr[i] or i[arr] } Both are exactly same, & gives same output

this means, ith element of arr

this means, arr of ith element.

Date.....

* when we directly access the eleme-nts of array then why we use pointers?

Stack memory is very limited memory. Till now, we have created all the arrays in stack memory. In future, we create all the arrays in Heap memory, & only pointers is used to create & fetch array from & to Heap memory.

```
arr = arr + 2;
```

} this will give error because we can't change the base address of array, because this is linked to Symbol Table.

But,

```
int *p = arr;
p = p + 2;
```

} when we pass the address to pointer & then update the pointer, it will not give error.

(It means, through pointer we can also show the subpart of the array.

```
int arr[10];
cout << sizeof(arr); //o/p :- 40
cout << sizeof(arr[0]); //o/p :- 4
int *p = arr;
cout << sizeof(p); //o/p :- 8
cout << sizeof(*p); //o/p :- 4
```

total space taken by array

→ total space taken by pointer.

this is compiler depended output

```
int arr[10] = {1, 5, 2, 5, 10};
cout << arr;
int *p = arr;
cout << p;
```

→ this will print the base address of the array.

then this will also print the base address of array.

→ when we assign the base address to pointer.

```
char ch[10] = "Babbar";
cout << ch;
char *c = ch;
cout << c;
```

→ the string is fully printed in the case of char array because, behaviour of cout is changed in case of char array & pointers.

Spiral

Date.....

```
char ch = 'k';
char *cptr = &ch;
cout << cptr;
```

this will print 'k' & after that garbage values is printed until it gets null character.

```
char name[10] = "Hello";
cout << name;
char *c = "how";
cout << c;
```

Both will print the string.

this will first store the string in temp storage. Then copy the string in name array storage, or in permanent storage.

this will first store the string in temp variable storage. Then the pointer c will point to the first character of the string.

this is a BAD PRACTICE

* Pass array to function :-

```
#include<bits/stdc++.h>
using namespace std;

void findSize(int arr[]){
   cout << "size in findSize() function:"
        << sizeof(arr);
}

int main(){
   int arr[] = {10,20,30,40};
   cout << "size in main() function:"
        << sizeof(arr);
   findSize(arr);
   return 0;
}
```

In main() functi-on, when we check the size of the array then it gives 24 bytes

when we pass array to a function & then check the size of array in that function, then it gives 8 bytes.

This is because, array is always passed as a reference, means pointer of Base address is passed as a reference in the function, & pointer always consume 8 bytes of space in the memory.

Date.....

```
#include<bits/stdc++.h>
using namespace std;
void solve(int arr[])
{
   cout<<"size of arr inside solve():"
      <<sizeof(arr);
   cout<<"arr address :"<<arr;
   cout<<"&arr address :"<<&arr;
   arr[0] = 5000;
}
int main()
{
   int arr[10] = {1,2,3,4,5}; int n = 10;
   cout<<"size of arr inside main():"<<
                                 sizeof(arr);
   cout<<"arr address :"<<arr;
   cout<<"&arr address:"<<&arr;
    for(int i=0; i<n; i++)
    {
      cout<<i[arr];
    }
    cout<<"calling solve() function";
    solve(arr);
    cout<<"back to main() function";
    for(int i=0; i<n; i++)
    {
      cout<<i[arr];
    }
    return 0;
}
```

Date.....

Ques why we cannot do "arr = arr + 1" in C++?

In C++, the name of an array is actually a constant pointer to the first element of the array. This means that you cannot modify the value of the array pointer itself, i.e., the address of the first element of the array, by using pointer arithmetic. So, if we try to do something like arr = arr + 1, you will get a compilation error because you are trying to modify a constant pointer.

This is because the pointer arr is pointing to the memory location of the first element of the array & we cannot change this address.