COMSATS University
Islamabad
(Lahore Campus)

CSC103- Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

RELATIONSHIP OF ARRAYS AND POINTERS

Valid operations with Pointers (Contd.)

- In general, if ptr and ptr2 are pointers to <type>, x is an integer, then:
  - ptr+x Evaluates as ptr+x*sizeof(<type>)
  - Ptr-x Evaluates as ptr-x*sizeof(<type>)
  - Ptr2-ptr Evaluates as (ptr2-ptr)/sizeof(<type>)

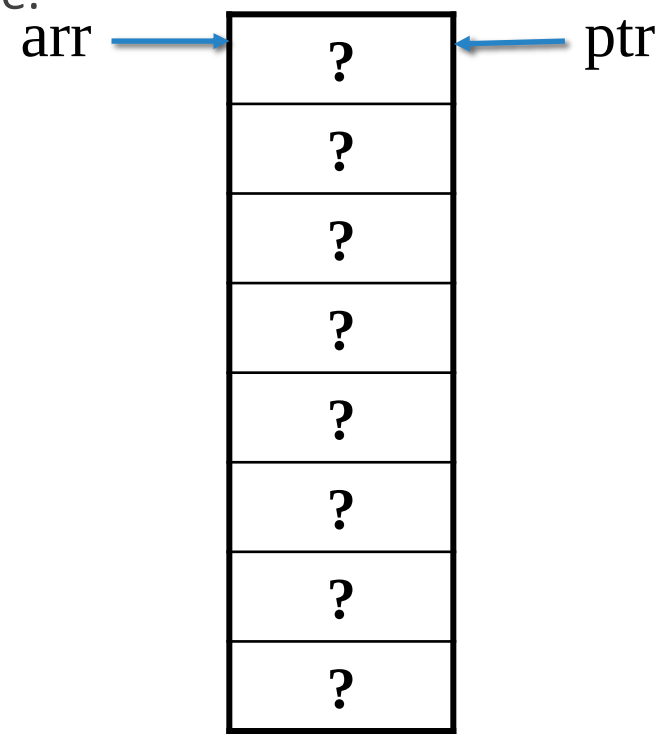
Arrays and Pointers

An array name is actually a pointer, or an address value.

- E.g. `int arr[8];`
  - `// point arr to first element address`
- Here, array name (arr) is actually a pointer to the 1st element of the list of allocated integers. The variable **`arr`** actually refers to &arr[0].

We can also initialize a pointer to first element of an array using:

- `int *ptrArr = &arr[0];`
- Or simply, `int *ptrArr = arr;`



Arrays and Pointers (Contd.)

Difference between an array pointer and an ordinary poitner

An array name is actually a *constant* pointer.

- The address where it points is fixed (does not change). For example,

```
int arr[] = {1, 2, 3};
int arr2[] = {2, 3, 4};
arr = arr2;  // WRONG: array name arr is a fixed pointer, it cannot be changed.
```

However, ptrArr in the following example can change.

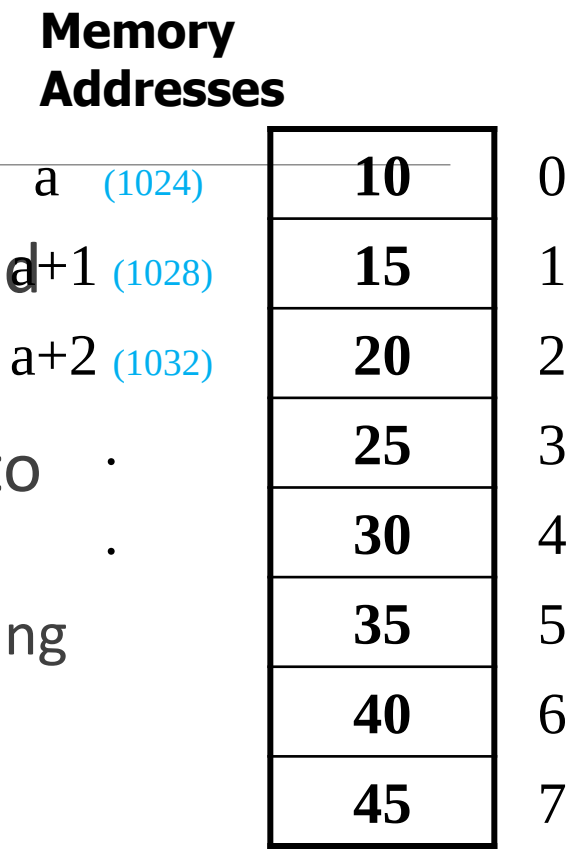
```
int *ptrArr = arr;
ptrArr = arr2;
```

Subscripting the array

We know that arrays can be subscripted to access an array element like a[i].

Considering array name ‘a’ as pointer to first element of the array:

- Referring a[i] is actually the same as referring *(a+i).
- For example:

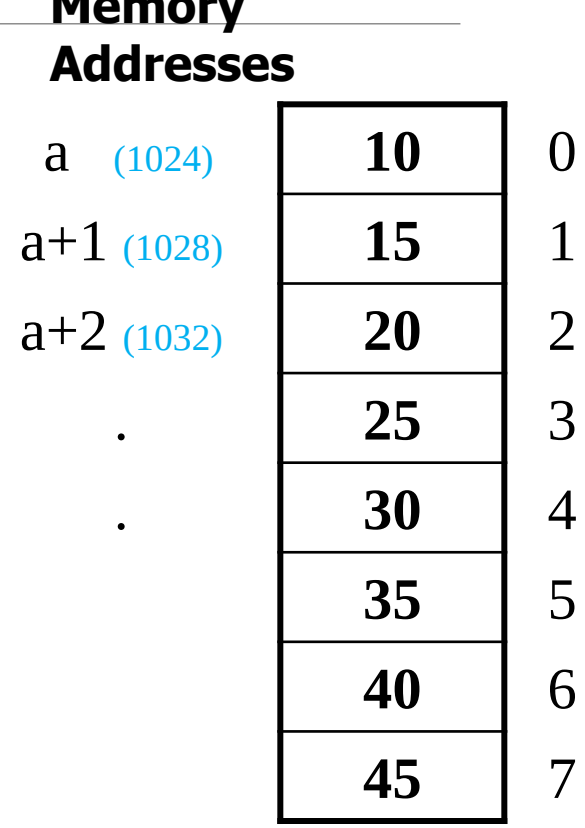



```
a[1] <==> *(a+1)<==> *(1024+1*4)<==> *(1028)→ 15
a[2] <==> *(a+2)<==> *(1024+2*4)<==> *(1032)→ 20
```

and so on …

Subscripting the pointer

- We can even subscript the poitners in the same way. (After all, array name was nothing else but a pointer to 1st element).




```
int *p = a;  // equivalent to ptr = &a[0];
```

Here, **p** points to the first element of the array.

We can get other elements by offset just like arrays, i.e. **p[i]**.

Referring p[i] is actually the same as referring *(p+i). E.g.

p[1] <==> ***(p+1)** <==> ***(1024+1*4)<==> *(1028)→15**

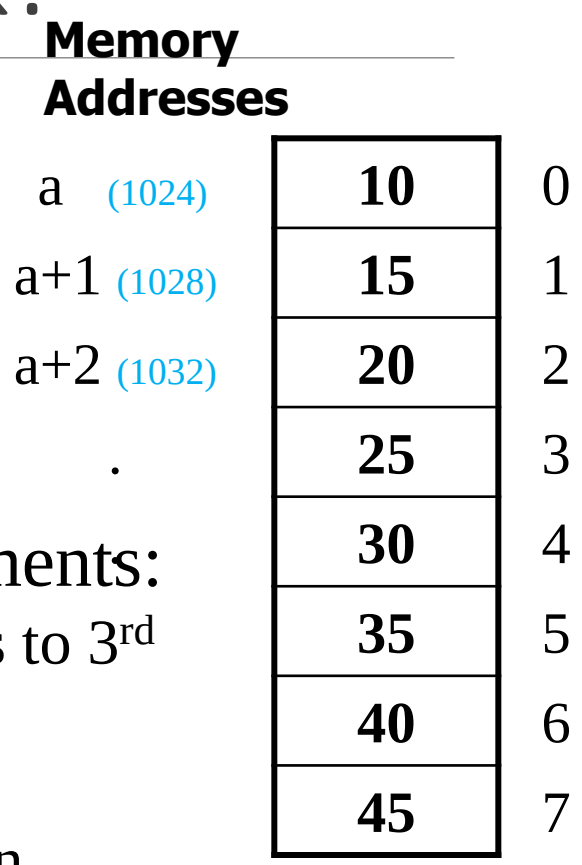
p[2] <==> ***(p+2)** <==> ***(1024+2*4)<==> *(1032)→20**

and so on …

Put your mind to work!

Now that you understand what happens in memory when a[i] is evaluated, consider the following scenario:

```
int *p = a;
int *p2 = &a[2];
```

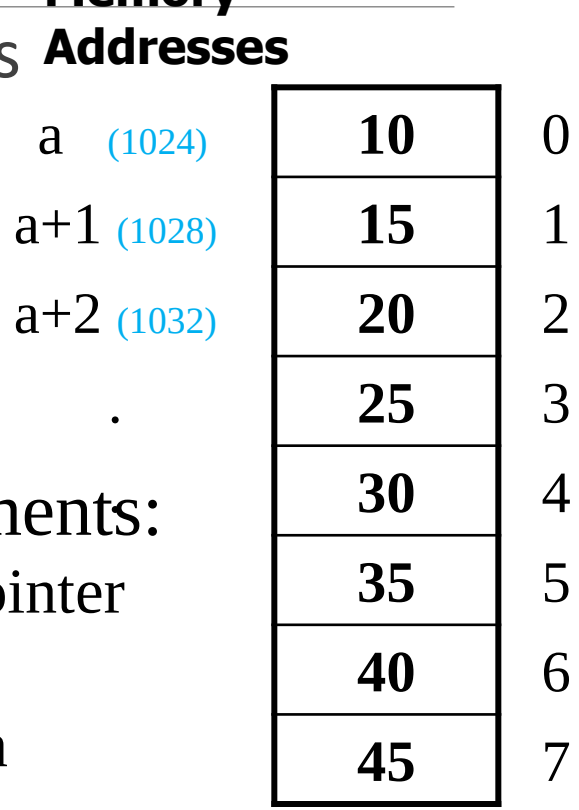


- What will be the result of these statements:
  1. `cout<< p2[0];` // be careful, p2 points to 3rd element of array.
  2. `cout<< p2[1];`
  3. `cout<< p2[-1];` // Hint: subtracting an integer from a pointer.

Put your mind to work!

Now that you understand what happens in memory when a[i] is evaluated, consider the following scenario:

```
int *p = a;
Int *p2 = &a[2];
```



- What will be the result of these statements:
  4. `cout<< p2-p;` // Hint: subtracting a pointer from another pointer.
  5. `cout<< p;` // Hint: This will display an address.
  6. `cout<< *p;`
  7. `cout<< ++p;` // Hint: `++p` is same as `p=p+1`.

Output

```
p2[0] = 20
p2[1] = 25
p2[-1] = 15
p2-p = 2
p = 0x61fdf0
*p = 10
++p = 0x61fdf4

Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

Example

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

int main()
{
   int arr[5] = {4, 2, 5, 2,1};
   cout << arr << " " << &arr[0] << endl;
   cout << arr[2] << " " << *(arr+2) << endl;
   int *p = arr; // p = &arr[0]
   cout<< arr[2] << " " << p[2] << endl;
   cout<<p[0]; // *(p+0)

   return 0;
}
```

```
1  // Fig. 8.6: fig08_06.cpp
2  // Pass-by-value used to cube a variable's value.
3  #include <iostream>
4  using namespace std;
5
6  int cubeByValue( int ); // prototype
7
8  int main()
9  {
10    int number = 5;
11
12    cout << "The original value of number is " << number;
13
14    number = cubeByValue( number ); // pass number by value to cubeByValue
15    cout << "\nThe new value of number is " << number << endl;
16 } // end main
17
18 // calculate and return cube of integer argument
19 int cubeByValue( int n )
20 {
21    return n * n * n; // cube local variable n and return result
22 } // end function cubeByValue
```

```
The original value of number is 5
The new value of number is 125
```

```
1  // Fig. 8.7: fig08_07.cpp
2  // Pass-by-reference with a pointer argument used to cube a
3  // variable's value.
4  #include <iostream>
5  using namespace std;
6
7  void cubeByReference( int * ); // prototype
8
9  int main()
10 {
11    int number = 5;
12
13    cout << "The original value of number is " << number;
14
15    cubeByReference( &number ); // pass number address to cubeByReference
16
17    cout << "\nThe new value of number is " << number << endl;
18 } // end main
19
20 // calculate cube of *nPtr; modifies variable number in main
21 void cubeByReference( int *nPtr )
22 {
23    *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
24 } // end function cubeByReference
```

```
The original value of number is 5
The new value of number is 125
```