

An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 16 Part - 1
Array Part – 2
Introduction

Hello and welcome to the NPTEL course An Introduction to Programming through C++, I am Abhiram Ranade and this lecture is about arrays. The reading for this is chapter 14.


(Refer Slide Time: 0:43)

Arrays: the view so far

Defining an array:

```
elementype aname[asize];
```

- Creates variables `aname[0]`, ..., `aname[asize-1]`
- Each is of type `elementype`
- `aname` : name of array
- Informally `aname` denotes the entire collection of variables `aname[0]`, ... `aname[asize-1]`




So in the last lecture as well we talked about arrays and so far we have said that you can define an array by writing something like this, give that name, give the type of the array, then give the name and give a size and this creates so many variables, each of type `elementype` that you have mentioned in the definition and `aname` is the name of the array and informally, `aname` denotes the entire collection of variables, `aname[0]` through `aname[size-1]`.

(Refer Slide Time: 1:11)

Outline

- The computer's view of arrays
 - Where the elements are stored in memory
 - How the computer indexes into an array
 - What happens when an index out of range is used
- Function calls using arrays
- A function for sorting an array.
 - Sort: rearrange elements so that they are in non-decreasing or non-increasing order



Now in this lecture, lecture sequence we are going to look at how the computer or how C++ views arrays, what exactly happens when we do array operations. So for example, where are the elements stored in memory? How does a computer index into an array? And what happens if you give an index out of range? Then we are going to talk about how function calls can be made on arrays and we will talk about a function for sorting an array. By sort I mean rearrange the elements so that they are in non-decreasing or non-increasing order. The order that we are going to use is going to be non-decreasing.


(Refer Slide Time: 2:06)

Computer's view of array definition

```
int q[4] = {11,12,13,14};
```

Assumption: a single `int` uses one four byte word

- 4 consecutive words of memory are allocated to `q`.
- Allocated memory used to store the variables `q[0]`, `q[1]`, `q[2]`, `q[3]` in order.
- Initial values stored in the allocated region.



So what is the computer's view of an array definition? So let us say we have this array q with integer elements and 4 integer elements and these being the initial values. So in, what I am going to discuss next, I am going to make the assumption that a single int uses one four byte word. So, if I want to store this entire array I need four, four byte words or four and they have to be consecutive, and so four consecutive words of memory are allotted to this array q. And in that allocated memory we are going to store the variables q0, q1, q2, q3 in order and also when this statement executes the initial values will be stored in the allocated regions.

(Refer Slide Time: 3:06)

Computer's interpretation of array name

Address	Used for	Content	Array name
1004	q[0]	11	= address of allocated region
1008	q[1]	12	= address of 0 th array element.
1012	q[2]	13	• For our array: q = 1004
1016	q[3]	14	• Type of q : int *

• Array name is a pointer, but its value cannot be changed. "q = 1008" is illegal.

"Address": address of first byte.
 Address 1004: bytes 1004, 1005, 1006, 1007

So here is a possible outcome, so I have shown here the address so 1004 at address 1004, I am going to start storing q0, q0 uses four words, so byte 1004, byte 1005, byte 1006, byte 1007 is going to be used for q0. And in that we are going to store the content 11. So the data stored in these bytes 1004 through 1007 is going to be 11 because we said that q0 should contain the value 11.

Then q1 will be stored likewise in bytes 1008, 9, 10 and 11 and these 4 bytes will contain the binary equivalent of 12. Then similarly, 12, 13, 14, 15 the addresses the corresponding bytes will be used for storing q2 and 16, 17, 18, 19 will be used for storing q3. And again they will also be initialized with the appropriate bit strings representing in binary, or actually in binary two's complement the numbers 13 and 14.

Now, this is fine, this is what, these are details, but now let us come to how does a computer, how or how does C++ think about the array name, so how does C++ think about q? So the array name is thought of as not as the collection of all these variables, but simply the starting point of this region. So this region is allocated, so what is the region that is given to us for storing the array q? It starts at 1004 and the last byte for this is 16, 17, 18, 19 so 1 0 1 9, so this area from 1004 through 1019 is what is given to this array q. So, the array 'name' C++ has decided is going to be, is going to mean or is going to have the value rather equal to the address of the allocated region. So in our example or it is also the address of the 0th element and in our example q will then mean 1004. So this is natural in a way and not natural in a way because it is, so what would be sort of the right thing to have done over here? So we could have said that q really means the entire area from 1004 through 1019, but instead of that C++ sort of makes it simple it just wants to associate one value with the symbol q. And so it says that it is going to be the starting address. So the starting address of the entire area or the address of the zeroth element, so the address of the zeroth element is really the same as the address of the allocated region. So q is going to be 1004. Now the value of q has not mattered until so far until so far, until so long, until now, it has not mattered so far. But we will see that it will, it will matter soon enough, it will be used soon enough.


And we also have a type associated with this name, so the type associated with this name is going to be int*. Well that is consistent with its value, its value is 1004 which is an address. And therefore, the type of an address to an int variable is int* and so the type is defined in consistency with our first decision. So that is how C++ uses q, its value and its type. And the array name is a pointer, so it is an address and or a pointer, but its value cannot change. So when we talked about pointers, when we talked about functions, our pointers could be changed, but this pointer cannot be changed. So I cannot say in my program q equals 1008, because, in fact you can think of q as being a 'const'. So you remember that we can declare variables as being const. So q is actually a const pointer, so if you write something like this C++ will flag it as an error. The idea is that look q really is referring to this array, and while it is not telling you till how long the array extends, it is telling you where the array begins and so please do not change it that is what, that is what this whole, the rationale of this whole discussion is.

(Refer Slide Time: 8:38)

In general

```
elemtype aname[alength];
```

- Block of memory of length $S * \text{alength}$ is allocated,
S = size in bytes of a single elemtype variable.
- aname = starting address of zeroth element = address of allocated block.
- Value of aname cannot be changed.
- Type of aname: elemtype *
- Type of aname[i] : elemtype




In general what happens, so if you have elemtype aname[alength], a block of memory of S times alength is allocated. And S has to be the size of a single elemtype variable. And so aname then is the starting address of the zeroth element or the address of the allocated block just like our q example. And the value of aname cannot be changed, and the type of aname is elemtype* or pointer to elemtype. So this is just a generalization about of what we said for our original definition int q[4].

And of course the type of aname[i] is elemtype, the type of just this is elemtype* and the type of aname[i] is elemtype and that applies to q as well. The type of q[0] is integer and the type of q was int*.

(Refer Slide Time: 9:54)


Exercise



What is printed when the following executes?

```
int q[4]={11,12,13,14}, r = 2;
cout << q << r << &r << endl?
cout << q[0] << &q[0] << endl;
```

- Make reasonable assumptions if you wish, or say if some value cannot be predicted.
- Reasonable assumption: the compiler allocates memory in increasing order of addresses.




So an exercise for you, figure out what is printed when the following executes? And in this you are expected to make reasonable assumptions if you wish or if you do not want to make any assumptions say that, that value cannot be predicted. And let me tell you what reason it assumption might be that you may say that, so as the compiler is making is allocating memory it sort of allocates memory in increasing order of addresses, so that might be (reasoning) reasonable assumption.

(Refer Slide Time: 10:31)

Exercise

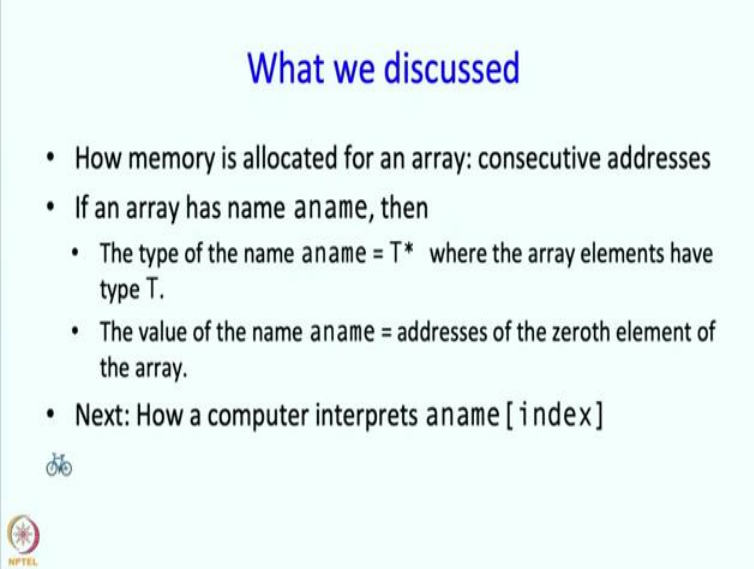
Point out the mistakes in the following program fragment.

```
int A[5];
A[0] = 10;
cout << A[0]<<' '<<&A[0]<<' '<<&A<<endl;
```





Here is one more, so again you are given a fragment and you are supposed to point out mistakes in the program fragment.

(Refer Slide Time: 10:38)



What we discussed

- How memory is allocated for an array: consecutive addresses
- If an array has name `aname`, then
 - The type of the name `aname = T*` where the array elements have type `T`.
 - The value of the name `aname = addresses of the zeroth element of the array.`
- Next: How a computer interprets `aname [index]`

Alright, so what have we discussed in this segment? We said how memory is allocated for an array? And basically consecutive addresses assigned, contiguous region of memory is given to you and we also said that if an array has name `aname`, then the type of the name is `T*`, where the array elements have type `T`. And the value of the name `aname` is the address of the zeroth element of the array or the address of the starting or the starting address of the allocated region for this array. In the next segment we are going to see how a computer interprets the expression `aname of index`, but before that we will take a small break.