An Introduction to Programming through C++

Abhiram Ranade

Lecture 7.2

Ch 14: Arrays, part 2.

Arrays: the view so far

Defining an array:

elemtype aname[asize];

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

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

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.

Possible outcome

Address	Used for	Content
1004	q[0]	11
1008	q[1]	12
1012	q[2]	13
1016	q[3]	14

“Address”: address of first byte.

Address 1004: bytes 1004, 1005, 1006, 1007

Computer's interpretation of array name

Address	Used for	Content
1004	q[0]	11
1008	q[1]	12
1012	q[2]	13
1016	q[3]	14

"Address": address of first byte.

Address 1004: bytes 1004, 1005, 1006, 1007

Array name

= address of allocated region

= address of 0th array element.

- For our array: $\boldsymbol{q}$ **=** 1004
- Type of q : int *
- Array name is a pointer, but its value cannot be changed. "q = 1008" is illegal.

In general

elemtype aname[alength];

- Block of memory of length ***S*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***

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.

Exercise

Point out the mistakes in the following program fragment.

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

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]***

How does the computer interpret ***aname[index]***

- [] is a binary operator!
- ***aname, index*** are the operands.
- ***aname[index]*** means
  - The variable stored at ***aname + S * index***, where S = size of a single element of the type ***aname*** points to.
  - Example: Next
  - Yes, the computer does a multiplication and addition to find the position of the element in memory.
  - Note that only a single multiplication and addition is done, however large the array is.

Example

Our old array q

int q[4];

Address	Used for
1004-7	***q[0]***
1008-11	***q[1]***
1012-15	***q[2]***
1016-19	***q[3]***

q = 1004

type of ***q*** = ***int****

Computer's view of q[3]

q[3] :

- variable of the type that q points to,
- q has type int*, so q[3] has type int
- stored at address ***q + s*3*** where ***s*** is size of a single variable of the type that ***q*** points to.
- variable of type ***int***, stored at 1004 + 4*3 = 1016.
- Same as what we call ***q[3]***

Summary: How a computer gets to aname[index]

- The index is multiplied by the element size and added to the starting address to get the position in memory where the variable is stored.

- That variable is used.

Index out of range

Our old array q

int q[4];

Address	Used for
1004-7	***q[0]***
1008-11	***q[1]***
1012-15	***q[2]***
1016-19	***q[3]***

q = 1004

type of ***q = int****

Suppose we execute:

q[10] = 34;

Mechanical interpretation as per our rule:

- variable of the type that q points to, stored at address q + 10*S where S is size of a single variable of the type that q points to.
- variable of type int, stored at 1004 + 10*4 = 1044.
- 34 will get stored in address 1044 which is not part of q!
- Possibly some other variable will be written into!

x = q[10] : x will get some strange value.

Summary

If you read or write from an improper address such as 1044:

- You may store data into some wrong place.
- You may get data from a wrong place.
- Occasionally, the addressed may have been deemed “protected” – then your program may abort.

So make sure index is in correct range!

More remarks

Some programming languages prevent index out of range by explicitly checking.

- First the value of the index is checked to see if it lies in the range 0..size-1.
- If it does not, an error message is printed and the program stops.

Index checking is not done in C++

- because it takes extra work, and
- because C++ designers believe that it is the programmer’s job to ensure that the index is in range.

Example

- What does the following code do?

```
int q[4];
int *r;
r = q;
r[3] = 5;
cout << q[3] << endl;
cout << r[3] << endl;
```

- The array ***q*** is allocated in memory.
- Variable ***r*** is created.
- ***q*** = address of the zeroth element of the array is placed in ***r.***
- Because ***r, q*** have the same value, ***r[3], q[3]*** also denote the same variable.

What we discussed

- ***aname[index]*** is an expression with [] as operator.
- When the index is in range, the expression when evaluated, tells what variable is meant.
- If index is out of range, then the expression does not denote a valid variable.
- Calculation happens fast, in time independent of the array length.
- ***aname[index]*** is a valid expression if ***aname*** is a pointer.
- Next: Arrays and function calls.