Comp 11 Lectures

Mike Shah

Tufts University

June 12, 2017

Arrays and Strings

Comp 11 - Pre-Class warm up

In most programming languages, we start counting from 0 (as opposed to 1). Why do you think this is?

0000 0000 - Binary for zero
0000 0001 - Binary for one
We do not want to waste any space.

Lecture

Anita Borg

Figure 1: Anita Borg is one of the great systems programmers who made many contributions to Unix (What your mac and linux systems are based off of). Googling her name will also show her numerous contributions to the Anita Borg Women in Technology Institute

Abstraction

- In the previous lecture, we learned about functions.
- Before that, we learned about loops.
- Both are abstractions that save us time and allow us to consolidate and build larger programs.

In this lecture, we are going to learn about another abstraction.

Variable Initialization

- Say we want to store a database of information, we want to store the age of everyone at our university.
- Simple, here is one way to do that with a program (next slide)

Variable Initialization

```
#include<iostream>

int main(){

int age0 = 24; // Remember, we like to count from zero
int age1 = 23;
int age2 = 24;
int age3 = 32;
int age4 = 64;
int age5 = 77; // I plan to be a life longer learned
int age6 = 32;
int age7 = 54;
// ..
  return 0;
}
```

Listing 1: We can use as many variable as we want as long as they are uniquely named.

Large Number of Variable Initializations

- Now if I restate the problem and say we need to do this for 10,000 students.
- All of a sudden we have to do a large number of copying and pasting.
- This is prone to mistake, and does not scale well if we want to make changes

Array

Our first data structure - the array

An array is here to solve our problem.

- Arrays are homogeneous, meaning they store the same data type (either all integers, all floats, all doubles, etc).
- Arrays allow us to index into the data to change values.
- We can modify the contents of an array very efficiently when we know the index of the data we want to change.
- Arrays are a data structure that are stored contiguously in memory.

Array with ages

```
1 #include<iostream>
2
3 int main(){
4
5 // This is it, we now have 8 int's stored.
6     int age[8];
7
8     return 0;
9 }
```

Listing 2: A much more concise piece of code

Array with ages, this time storage for 1000 entries

```
1  #include<iostream>
2
3  int main(){
4
5  // This is it, we now have 800 int's stored.
6  // Only 1 change to the code, the initialization is to 800
7      int age[800];
8
9      return 0;
10 }
```

Listing 3: A much more concise piece of code

Setting up our age array

```
#include<iostream>

int main(){

int age[8];
// Remember, we index from zero
age[0] = 24;
age[1] = 23;
age[2] = 24;
age[3] = 32;
age[4] = 64;
age[5] = 77;
age[6] = 32;
age[7] = 54;
//  ..
  return 0;
}
```

Listing 4: Use a loop

Wait a second

- Wait a second, that looks like just as much typing.
- But the variable name for the array is the same, so that is convenient
- The indexing of each item also seems to give some sort of ordering, so that looks better.

Solution: Combine abstraction with loop(1/2)

```
1  #include<iostream>
2
3  int main(){
4
5  int age[8];
6
7  age[0] = 24; age[1] = 23; age[2] = 24;
8  age[3] = 32; age[4] = 64; age[5] = 77;
9  age[6] = 32; age[7] = 54;
10
11 // Can make use of a collection of data very easily.
12 for(int item: age){
13   std::cout << item << "\n";
14 }
15
16   return 0;
17 }
```

Listing 5: Range based for loop using c++11

Solution: Combine abstraction with loop (2/2)

```
#include<iostream>

int main(){

int age[8];

age[0] = 24; age[1] = 23; age[2] = 24;
age[3] = 32; age[4] = 64; age[5] = 77;
age[6] = 32; age[7] = 54;

// Iterate through half of our list
for(int i =0; i < 4; ++i){
  std::cout << age[i] << "\n";
}

  return 0;
}
```

Listing 6: for loop for iterating through part of the collection

Updating an array

- So remember, arrays are a collection of data.
- They are like variables in that we can update them.
- We just have to use the array name, and the index.

Updating an array Example

```
#include<iostream>
int main(){
int age[8];

age[0] = 24; age[1] = 23; age[2] = 24; age[3] = 32;
age[4] = 64; age[5] = 77; age[6] = 32; age[7] = 54;
for(int i =0; i < 4; ++i){
  std::cout << age[i] << "\n"; }

// 100 years later
age[0] = 124; age[1] = 123; age[2] = 124; age[3] = 132;
age[4] = 164; age[5] = 177; age[6] = 132; age[7] = 154;

  return 0;
}
```

Listing 7: Updated values from an array

Array Visualization - int age

int age[8];

index	0	1	2	3	4	5	6	7
value	??	??	??	??	??	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[0] = 24;

index	0	1	2	3	4	5	6	7
value	24	??	??	??	??	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[1] = 23;

index	0	1	2	3	4	5	6	7
value	24	23	??	??	??	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[2] = 24;

index	0	1	2	3	4	5	6	7
value	24	23	24	??	??	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[3] = 32;

index	0	1	2	3	4	5	6	7
value	24	23	24	32	??	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[4] = 64;

index	0	1	2	3	4	5	6	7
value	24	23	24	32	64	??	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[5] = 77;

index	0	1	2	3	4	5	6	7
value	24	23	24	32	64	77	??	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[6] = 32;

index	0	1	2	3	4	5	6	7
value	24	23	24	32	64	77	32	??

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Array Visualization - Set an age

int age[7] = 54;

index	0	1	2	3	4	5	6	7
value	24	32	24	32	64	77	32	54

- Note that all of the elements are integers of the same type
- The size of the array is fixed when we start the program.
- Default values not guaranteed

Another way to initialize array

```
#include <iostream>

int main(){
    // Can use this syntax to set values.
    int age [] = { 24, 23, 24, 32, 64, 77, 32, 54 };

    // Iterate through our list
    for(int item: age){
        std::cout << item << "\n";
    }

    return 0;
}
```

Listing 8: Abbreviated syntax for intialization

std::string

std::string our familiar friend.

- std::string is actually an array of chars.
- How exactly is it implemented?
- Lets take a look

std::string implementation Visual

std::string name = "Mike";

index	0	1	2	3
value	'M'	'i'	'k'	'e'

- Note the single quotes, remember a std::string is made up of individual char's.
- How exactly is it implemented?
- Lets take a look

String as array of char

```
#include <iostream>
// Note, no need to include <string>

int main(){
  // First way
     char name[] = {'M','i','k','e', '\0'};
  // Second look way without initializer
     char name2[5];
     name2[0] = 'M';
     name2[1] = 'i';
     name2[2] = 'k';
     name2[3] = 'e';
     name2[4] = '\0';

     std::cout <<  name << "\n";
     std::cout <<  name2 << "\n";

     return 0;
}
```

Listing 9: String representation

null character - '\0'

- One new idea in previous sample the null character.
- The slash '\0' (zero) at the end of the string.
- This signals to C++ when we *cout* (amongst other things) that we have reached the end of our array.
- We say this terminates the string.
- It can be dangerous to forget this, that is why we prefer to use *string*.

Passing strings into functions

```
1  #include <iostream>
2  #include <string>
3
4  void hello(std::string name){
5    std::cout << "hello " << name << "\n";
6  }
7
8  int main(){
9    hello("Mike");
10
11   return 0;
12 }
```

Listing 10: Strings with functions

A few note on arrays

- Remember a string is really a character array
- When we talk about types, an array if char's is a type. An array of int's is a type. That is different than a plain old int.
- We can pass arrays into functions as well (we just did it with the string).
  - An array is just a 'type' of information after all. The rules are the same as we learned before, just make sure the types of data match.
  - Useful to pass a size parameter so we know how large array is

Passing Arrays into functions

```
#include <iostream>
#include <string>
void printArray(int array[], int size){
  for(int i =0; i < size; ++i){
    std::cout << array[i] << " ";
  }
  std::cout << "\n";
}
int main(){
  int myArray[10];
  for(int i =0; i < 10; ++i){
    myArray[i] = i;
  }

  printArray(myArray,10);
  return 0;
}
```

Listing 11: Passing arrays into functions

In-Class Activity

http://www.mshah.io/comp/11/activities/activity5.pdf

Activity Discussion

Review of what we learned

- (At least) Two students
- Tell me each 1 thing you learned or found interesting in lecture.

5-10 minute break

To the lab!

Lab: `http://www.mshah.io/comp/11/labs/lab4.pdf`
[1]

[1] You should have gotten an e-mail and hopefully setup an account at `https://www.eecs.tufts.edu/~accounts` prior to today. If not–no worries, we'll take care of it during lab!