CS 102

Introduction to Programming Using C++

Chapter 7

Pointers

Homework

- Written homework
- R7.1, 2, 4, 7, 8, 15, 16, 19, 21 to 27

- Programs
- p. 344, choose one of P7.1, 2, or 4.
- Also, implement a linked list.

Arrays and Pointers

- Here is an idea from C itself
- These are the key points
  - A pointer contains an actual memory address,
  - An array is a block of bytes that are all physically together
    - We say the bytes are contiguous
- So, it would seem that a pointer could be used to point to (or reference) various elements of an array

A Pointer to an Array

- Suppose we have

  ```
  double costs [10];
  ```

- Then we can say

  ```
  double* cost_pointer = costs;
  ```

- Now cost_pointer points to the array costs
- You can say

  ```
  cout << *cost_pointer;
  ```

- which is the same as

  ```
  cout << costs [0];
  ```

- It’s even deeper than that!

Pointer Arithmetic

- You can use pointers in lots of clever ways
- One idea is called pointer arithmetic
- Once again, assume

```
double costs [10];
double* cost_pointer = costs;
```

- Then

  *(cost_pointer+2) is the same as costs [2];

- Change 2 to your favorite integer
  - Your favorite integer has to be less than 10 of course
- This still works!
- This equivalence is called the array/pointer duality law

Why Does This Work?

- Looking inside memory explains why this works
- Suppose an int gets 4 bytes
  - It's frequently 4 or 8 bytes, but it doesn't have to be
- Then we can see why a pointer can be used as a reference to an array
- array [n] is truly at the same memory location as array + 4n

Problems from the Textbook to Check Our Knowledge

- p. 317-318
- Problems 6, 7, 8, 9
- Do these to check on pointer arithmetic

Using a Pointer to Step through an Array

- Two ways to print an array
- Assume: int counts [10];
- Method 1

```
for (int i=0; i<10; i++)
    cout << counts [i] << endl;
```

- Method 2

```
int *p = counts;
for (int i=0; i<10; i++)
{
    cout << *p << endl;
    p++;
}
```

More Advanced Use of ++, --

- The increment, decrement operators can occur inside of statements, not just on their own
- For example, you can code

```
int n = 10;
cout << n--;
```

- You can also code

```
int n = 10;
cout << --n;
```

Following the Examples

- This code

```
int n = 10;
cout << n--;
```

- prints 10 and then decrements n.
- This code

```
int n = 10;
cout << --n;
```

- decrements n and then prints 9.

Operators Before Variables, Operators After Variables

- If an increment/decrement operator occurs *before* the variable
  - the increment/decrement is done *before* the statement is executed
- If an increment/decrement operator occurs *after* the variable
  - the increment/decrement is done *after* the statement is executed

Tracing Some Code

- What will print in the example below?

```
int x, y;
x = 1;
cout << "x = " << x << endl;
y = ++x   +   2;
cout << "x = " << x << endl;
cout << "y = " << y << endl;
y = x++   +   2;
cout << "x = " << x << endl;
cout << "y = " << y << endl;
```

Tracing Code from the Textbook

```
double sum (double* a, int size)
{
    double total = 0;
    while (size-- > 0)
        total = total + *a++;
    return total;
}
```

- Trace the code with

  ```
  double a [3] = {15, 10, 20};
  sum (a,3);
  ```

- What is in a at the end of the function call?
- What does the function return?

A Second Example from the Textbook

- Be careful when using pointers with local variables

```
double* firstlast (double a[], int size)
{
  double result [2];
  result [0] = a [0];
  result [1] = a [size-1];
  return result;
}
```

C Strings

- C++ has a kind of string that is a holdover from C itself
  - C++ programmers call it a “C string”
- This string is just a character array that has a special character at the end
  - The character is ‘\0’
    - In fact, there are several ‘\’ codes
  - That is the actual number 0
  - It is not a real, printable character
  - It’s called a null terminator

Character Arrays

- In C, the only way to find the end of a string was to search for ‘\0’
- This made it easy to mismanage strings
- If you overwrote that character accidentally, your string would lose its end
- This means that C programmers would actually manage the ‘\0’ in their programs!

C vs. C++

- In C++, we have the length() function
- C has many different string functions that work with strings
- All those functions expect the null terminator at the end of the string
- Many of those functions make string software susceptible to hacking
  - Code using those specific functions has to be rewritten

C String Functions

- Since C++ grew out of C, there are a lot of string functions from C in C++ code
- You should check out the list on p. 324
- Again, many of these are not safe, and are not used any more
  - For example, strncpy (t, s, n) vs. strcpy (t, s)

Storing Data

- We know several data types
    - Some are string, int, double, bool
- We want to make a new type, one that is more general
- We want to store more than one single data item
- For example, we want to store a person’s name and address
    - We need to store these items
        - The person’s name
        - The street address
        - The city
        - The state
        - The zip code

Parallel Arrays

- One way to store this would be using parallel arrays
- This means we will have five arrays
  - name, street, city, state, zipcode
  - They will all be strings
- Information for Person 1 will be in

  name[0], street[0], city[0], state[0], zipcode[0]
- Information for Person 2 will be in

  name[1], street[1], city[1], state[1], zipcode[1]

A Problem

- We have already seen variables used as subscripts
- Sometimes people calculate subscripts
- The subscript comes from a formula
  - It’s not just a single variable
- How are these arrays “held together”?
- What’s to stop bad programming from associating name [0] with street [1]?

A Solution

- A better idea would be to create a new array
    - Call it info[ ]
- We will store
    - *all* the information for Person 1 in info [0]
    - *all* the information for Person 2 in info [1], etc.
- How can we do this?

Creating a struct

- First, we create a struct
- A struct is a way of grouping different data items together
- An example is

```
struct PersonInfo
{
        string name;
        string street;
        string city;
        string state;
        string zip_code;
};
```

Using a struct

- A struct is a type
  - It’s just like int, bool, double, etc.
- Just like we can code

```
int number_of_items;
```

- we can code

```
PersonInfo a_person;
```

- We have created a new type!

Accessing Parts of the struct

- We can set values into the struct
  - We can store a name into the struct

```
a_person.name = “Cay Horstmann”;
```

- We can print a person’s city and state

```
cout << a_person.city << “, “ << a_person.state;
```

- We can test if a person lives in California

```
if (a_person.state == “CA”)
```

Practice

- Let’s design structs for
  - A driver’s license from the DMV
  - A car (as viewed when you want to get a license plate)
  - A student (as viewed from the school)
  - A book (as viewed by the library staff)

Back to That Array

- Now we can create the array we wanted
  - PersonInfo client_data [10];
- This overcomes the drawback of parallel arrays
- Now if we use client_data [15] in our code, it refers to *all* the information for a client
- The computer is keeping it together
  - The programmer doesn’t have to do it

Questions

- Are there any questions?