CS 008
Lecture notes
2/29/24

Note:

Triangle lab has been pushed back to Tuesday

1 Error messages

Most students are presumably using Visual Studio Code to write their code. There have been a few issues with compiling and debugging that students often encounter with VSC. When you click the "play" arrow and run your code and it does not work, you'll notice that a pop up will appear telling you "... exe" does not exist. Ignore this, it's only telling you that the compilation failed. Look at the "TERMINAL" window at the bottom of the screen to figure out the issue. Even without Visual Studio Code, you will STILL encounter some error output

1.1 Tips for using error messages:

Everybody will encounter an error message at some point and it should not be intimidating. There are 3 stages to C++ compilation and each stage has it's own unique errors.

- Preprocessing (Involves #statements)
- Compiling (Syntax errors - *Look for line numbers of where the error occured.*)
- Linking (Can't find files - Missing files or maybe misnamed files)

Usually, compiling errors will be the common ones you'll find. When working with error messages, you should start looking at the file names it complains about. Don't be intimated by the error messages that you do not recognize, they usually don't matter. Usually you should look at the first or last lines of the error message. Typically, those are the most signficant.

Do not be discouraged to come across more errors while debugging, it's a part of the process.

1.2 Documentation

Variable names and comments are very important when writing code. It's more readible for other people and for yourself.

Note:

Variable names should have meaningful names and comments should be written alongside your code.

2 Operator overloading

Binary operators such as +, -, =, ... etc cannot be used by a new class unless it is explicitly defined for it. You can basically define a new 'definition' for any other binary operator for your class. Defining a new meaning for an operator is known as **overloading** an operator.

Example

```
// FILE: point.cpp
// CLASS IMPLEMENTED: point (See point.h for
↪  documentation.)
#include "point.h"
point::point(double initial_x, double initial_y)
{
        x = initial_x;   // Constructor sets the point to
↪  a given position.
        y = initial_y;
}

void point::shift(double x_amount, double y_amount)
{
        x += x_amount;
        y += y_amount;
}

void point::rotate90( )
{
        double new_x;
        double new_y;
```

```
        new_x = y;  // For a 90 degree clockwise
↪   rotation, the new x is the original y,
        new_y = -x; // and the new y is -1 times the
↪   original x.
        x = new_x;
        y = new_y;
}
```

Basically you can decide what you want your operator to do through **overloading**.

Overloading Binary Arithmetic Operators:

This is a global overload of the binary operator +.

```
point operator +(const point& p1, const point& p2)
{
        double x_sum, y_sum;
        x_sum = p1.get_x() + p2.get_x();
        y_sum = p1.get_y() + p2.get_y();
        point sum(x_sum, y_sum);
        return sum;
}
```

You could also have local overloads within classes.

Warning:

When you are overloading, you are always defining new operators for data types. In addition, the function must be a friend function since it requires direct access to the private members of a data class.

3 Friend functions

When we have a class, all attributes in a class can be either private or public. Public members can be accessed by all classes with no restrictons while private members can only be used by the class itself except for **friend functions**.

A friend function can access the private members of a class. For example, when we want to access a private variable from a class when overloading, we would need to define it as a friend function.

Anything that is private outside the class, is public for a friend function.

Note:

Friendship should be limited to functions that are written by a **programmer who implements the class**.

4 Container classes

A container class is a class that contains a collection of items. (This is a bit broad)

4.1 The Bag class

The Bag class is basically just a way to store integers.

Definition

```
class bag
{
public:
        bag();

        typedef int value_type;

        //size_t data type is an integer data type that
↪   can hold
        //only non-negative numbers.  Each C++
↪   implementation
        //guarantees that the values of the size_t type
↪   can hold the
        //size of any variable that can be declared on
↪   your machine.
        typedef std::size_t size_type;

        static const size_type CAPACITY = 30;

        size_type size() const;
        void insert(const value_type& entry);
```

```
        size_type count(const value_type& target) const;

        //provided the target is actually in the bag,
↪   function removes
        //one copy of target and returns true, otherwise
↪   returns false
        bool erase_one(const value_type& target);

        //removes all copies of the target and return
↪   value tells how
        //many copies were removed
        size_type erase(const value_type& target);

        //Union operator - the union of two bags is a new
↪   larger bag that contains
        //all the numbers in the first bag plus all the
↪   numbers in the second bag
        bag operator+(const bag& b1, const bag& b2);

        //overloading the += operator to activate the +=
↪   member function of the
        //first bag and use the second bag as the
↪   argument
        //ex first_bag += second_bag;
        void operator+=(const bag& addend);
};
```

Question:

Is `bag operator+(const bag& b1, const bag& b2);` an accessor or a mutator?

Answer: It is more likely to be an accessor than a mutator. Mostly because it returns a new instance.

Is `void operator+=(const bag& addend);` an accessor or a mutator?

Answer: It is a mutator because it returns void, therefore it most likely is a mutator.

5 Introduction to Big O notation

Let's say we have an algorithm. Sometimes this algorithm runs fast and sometimes it runs extremely slow. Every algorithm can be characterized by it's time complexity. We focus on the processing and the amount of inputs given to figure out the time complexity.

Seven Functions:

- Constant ≈ 1
- Logarithmic $\approx \log(n)$
- Linear $\approx n$
- n-log-n $\approx n\log(n)$
- Quadratic $\approx n^2$
- Cubic $\approx n^3$
- Exponential $\approx 2^n$

The use of logarithms here are not exactly the same as the ones you will encounter in your calculus class. Logarithms in computer science are in base 2 instead of the base 10 encountered in mathematics. This is the result of binary which is comprised of 2 states and is the primary language of computing systems. Outside of classical computing like in quantum computers, the base would be different due to qubits which can also be in a superposition state of these 0 and 1 states.

Constants

In algorithm analysis, a constant is catogorized as O(1) and there are certain operations that are considered to be O(1) including:

- Adding and subtracting
- Assigning a value to a number
- Comparing two numbers

Using this knowledge: How could we figure out the time complexity of accessing the first element in the array? Well, intuitively you could guess that it has a time complexity of O(1) but why?

When we create an array, we know certain properties of the array. When accessing the first element in the array, it is constant because the addresss of the array is already known and the location of the first element can be found there. But what happens when we were to figure out the time complexity of accessing the second element in the array? Or maybe the third or fourth?

Arrays

When you make an array, there are three types of information that is known:

- The address of the array (Expressed in hexadecimal)
- the data type of the array
- The amount of the elements in the array

Accessing any specific element in the array is considered to be O(1) because of these three principles. Using the address of the array, the data type of the array and the amount of elements in the array, we can use addition to find a specific element in an array.

Question:

What's the time complexity of a linear search algorithm that iterates through an array to search for a specific value?

Answer: Well, the time it takes to execute this algorithm depends on the length of the list. Searching through the list for a number is O(n) because you iterate through the list and in the worst case scenario, it will take n iterations to find.

This is unlike directly accessing a specific element in an array which is considered as O(1).

There are differences between grabbing a specific element in an array and searching through an array.