Functions and Files

Joel Willoughby

COP 3503

Outline

Agenda

- Talk about some of the recurring problems from last lab
- Functions
- Files

Outline

Cohesion and Coupling

- When writing functions for a struct or class, try to keep things only *belonging* to that function in that function.
- This means if we have the following:

```
1 void addVertex(string vert){
2   //...
3 }
```

  This function should not have user input in it (it already has the string passed to it)

- Similarly, you should only write functions for a class or struct that are important for that class or struct. For instance, printing a menu is not really a part of a Graph's duty
- This holds for properties of objects as well. For instance, if you use a local variable in a function, don't make it a member, just declare it in the function

Expected Format (IO)

- Always read lab and project descriptions *carefully*
- When given a format, you should code to that format. If something isn't clear, ask before assuming.
- So, if the ouput says to print your vertices like:
  Vertices: [a] [b] [c]
  Your program should print in that exact format.
- Likewise, please follow naming conventions (ie, if the file should be called lab03.cpp, then name it lab03.cpp)
- For last lab, I wasn't strict, but I will be in future labs (and projects)

Misc

- When iterating through an array, typically, you go from i=0 until i < num, not i <= num.
- Things that are called everytime a loop executes should ideally be called just once in the loop.
- In general, reserve global variables for constants. Do not use them for temporary variables or control. If these are needed in multiple functions, pass them along.

Outline

Goal

- We will talk about functions and function prototypes in C++
- We will do some file I/O (if you haven't gotten it to work yet for your project)
- Finish early, ask questions about the project

Functions

- Last lab, I mentioned helper functions
- In general, anything that is repeated multiple times, or is a logical unit of code, you should consider wrapping in a function
- The alternative to this is putting all of your logic in main. This is harder to debug and understand for larger programs
- Have things in main that need to be accessed in multiple functions? Pass them as parameters (usually references or pointers)

Prototypes vs. Definitions

- A function prototype (also called a declaration or signature) consists of a function's return type, name, and the types of its parameters. Example of a function called foo that returns a bool and takes in an int and a string:

```
1 bool foo(int, string);
```

- The point of a prototype is to tell the compiler the structure of your function. This needs to be seen by the compiler *before* you use the function in your code.
- The function definition is the actual code of the function and can be defined anywhere after the prototype
- It can also act as a prototype if it is defined before it is used

Function Example

```
1  int max(int, int);
2
3  int main(){
4    int a = 5, b = 6;
5    cout << max(a, b) << endl;
6  }
7
8  int max(int n1, int n2){
9    if(n1 > n2)
10     return n1;
11   return n2;
12 }
```

Parameters

- No globals, multiple functions need access to certain objects. What do you do?
- Pass them as paramters.
- Assume we are using the Graph from last lab. This is not a correct way to do it:

```
1 void fun_1(Graph g){ //useful code }
```

Parameters

- No globals, multiple functions need access to certain objects. What do you do?
- Pass them as paramters.
- Assume we are using the Graph from last lab. This is not a correct way to do it:

```
1 void fun_1(Graph g){ //useful code }
```

- The parameter g is pass by value. If we want to edit it in the function, use references or pointers:

```
1 void fun_1(Graph & g){ //g is a reference }
2 void fun_2(Graph * gp){ //gp is a pointer }
3
4 //In main...
5 Graph g;
6 fun_1(g);
7 fun_2(&g);
```

Then, when you edit g or gp in the functions, the g you passed in will reflect the changes

File IO

- In C++, the two classes you would use for this are ifstream for input and ofstream for output.
- Both can be found in the fstream library. (Need to #include <fstream>)
- Before you do anything with the stream, you must open a file
- With file io, you should always check to make sure the file opened correctly
- From there, you can read (for ifstream) or write (for ofstream) as much as you want

Useful ifstream functions

- is.open(filename) - This opens a file called filename. Note that filename must be a cstring (not a string). You can get a csrting from a string by doing str.c_str(); (if str is your string)
- is.fail() - returns true if the stream failed to open the file
- is.eof() - returns true if the stream has reached the end of the file. Returns false otherwise
- is » var - works just like cin. You can use the » operator to read from a file just like cin reads from the console
- is.get() - returns a single character from the file. Useful for doing involved parsing.

Outline

Questions

???