CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 9
February 16, 2016

Notes on PS2

Comments on PS2

- Each class should be in a separate file.
- Put a blank line or other visual indicator before start of each class.
- Follow structure given in class and in textbook for overloading `operator<<`.
- Compiling code should give no warnings or errors.

Storage Management (continued)

Recall from previous lecture

Object properties:

1. type (primitive or class)
2. name
3. storage area (block of consecutive memory bytes)
4. lifetime (block, dynamic, permanent)
5. storage class (stack, heap, static)

Storage area

Every object is represented by a block of storage in memory.

This memory has an internal **machine address**, which is not normally visible to the programmer.

The size of the storage area is determined by the type of the object.

Lifetime

Each object has a **lifetime**.

The lifetime begins when the object is **created** or **allocated**.

The lifetime ends when the object is **deleted** or **deallocated**.

Storage class

C++ supports three different storage classes.

1. `auto` objects are created by variable and parameter declarations. (This is the default.)
   Their visibility and lifetime is restricted to the block in which they are declared.
   The are deleted when control finally exits the block (as opposed to temporarily leaving via a function call).
2. `new` creates anonymous *dynamic* objects. They exist until explicitly destroyed by `delete` or the program terminates.
3. `static` objects are created and initialized at load time and exist until the program terminates.

Assignment and copying

The assignment operator `=` is implicitly defined for all types.

- `b=a` does a shallow copy from `a` to `b`.
- Shallow copy on objects means to copy all data members from one object to the other using the assignment operator defined for each.
- Call-by-value uses the copy constructor to copy the actual argument to the function parameter.
- If the argument object contains pointer data members, the pointers are copied but *not* the objects they point to. This results in **aliasing**—multiple pointers referring to the same object.

Static data members

A static class data member must be *declared* and *defined*.

- It is declared by preceding the member declaration by the qualifier `static`.
- It is defined by having it appear in global context *with* an initializer but *without* the keyword `static`.
- It must be defined *only once*.

Example

In `mypack.hpp` file, inside class definition for `MyPack`:

```
static int instances; // count # instantiations
```

In `mypack.cpp` file:

```
int MyPack::instances = 0;
```

Static function members

Function members can also be declared `static`.

- As with static variables, the are declared inside class by prefixing `static`.
- They may be defined either inside the class (as inline functions) or outside the class.
- If defined outside the class, the `::` prefix must be used and the word `static` omitted.

Five common kinds of failures

1. **Memory leak**—Dynamic storage that is no longer accessible but has not been deallocated.
2. **Amnesia**—Storage values that mysteriously disappear.
3. **Bus error**—Program crashes because of an attempt to access non-existent memory.
4. **Segmentation fault**—Program crashes because of an attempt to access memory not allocated to your process.
5. **Waiting for eternity**—Program is in a permanent wait state or an infinite loop.

Read the textbook for examples of how these happen and what to do about them.

Bar Graph Demo

Overview of bar graph demo

These slides refer to demo 09-BarGraph.

This demo reads a file of student exam scores, groups them by deciles, and then displays a bar graph for each decile.

The input file has one line per student containing a 3-letter student code followed by a numeric score.

```
AWF  00
MJF  98
FDR  75
...
```

Scores should be in the range $[0, 100]$

Overview (cont.)

The output consists of one line for each group listing all of the students falling in that group. An 11th line is used for students with invalid scores.

Sample output:

```
00..09:  AWF 0
10..19:
20..29:
30..39:  PLK 37
40..49:
50..59:  ABA 56
60..69:  PRD 68 RBW 69
70..79:  HST 79 PDB 71 FDR 75
80..89:  AEF 89 ABC 82 GLD 89
90..99:  GBS 92 MJF 98
Errors:  ALA 105 JBK -1
```

Method

Each student is represented by an `Item` object that consists of the initials and a score.

The program maintains 11 linked lists of `Item`, one for each bar of the graph. A bar is represented by a `Row` object.

For each line of input, an `Item` is constructed, classified, and inserted into the appropriate `Row`.

When all student records have been read in, the bars are printed.

A `Graph` object contains the bar graph as well as the logic for creating a bar graph from a file of scores as well as for printing it out.

graphM.cpp

Points to note:

- `run()` calls a static class method `Graph::instructions()` to print out usage information. It is called without an implicit parameter.
  By being static, the instructions can be printed before any `Graph` object is created.
- The file uses `cin.getline()` to safely read the file name into a `char` array `fname`.
  The simpler `cin >> fname` is unsafe. It should never be used.
  It would be okay if `fname` were a `string`.
- After the file has been opened, the work is done in two lines:

```
Graph curve( infile ); // Declare and construct a Graph object.
cout << curve;         // Print the graph.
```