CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 13
October 15, 2018

Bar Graph Demo

Bar Graph Demo

Overview of bar graph demo

These slides refer to demo 13-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 11^{th} 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.

Analysis of 13-BarGraph demo

- `main.cpp`
- `graph.hpp`
- `graph.cpp`
- `row.hpp`
- `row.cpp`
- `rowNest.hpp`
- `item.hpp`

main.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.
```

Design issues for `main.cpp`

1. Should `instructions` be a static class method or a static constant?
2. Should `fname` be a `char[]` or a `string`? If the latter, how does one prevent buffer overrun?
3. Where should the file opening code go – in `run()` (where it is now), in `Graph`, or in a new **controller** class?

graph.hpp

Points to note:

- Class `Graph` *aggregates* 11 bars `Row`.
- The `Row` array is created by the constructor and deleted by the destructor.
- `insert()` is a private function. It creates an `Item` and inserts it into one of the `Row`s.
- `instructions()` is a `static` inline function. This shows how it is defined.
- `instructions()` could also be made out-of-line in the usual way, but the word `static` must `not` be given in the definition in the `.cpp` file; only in the declaration in the `.hpp` file.

graph.cpp

Points to note:

- The `for`-loop in the constructor does not properly handle error conditions and can get into an infinite loop.
  You should test yourself to be sure you know how to fix this problem.
- The constructor has an allocation loop. The destructor has a corresponding deallocation loop.
- `bar[index]->insert( initials, score );`
  shows the use of a subscript and a pointer dereferencing in the same statement.
- Why do we need the `*` in

  ```
  out << *bar[k] <<"\n";
  ```

Design issues for `Graph` class

1. Note the use of the C preprocessor to allow preprocessor macro `NESTED` to cause compilation in two different ways.
2. Could we declare `bar` as `Row& bar[BARS]`? How might this affect the program?
3. Should `initials` be a string?
4. Why is there a potential infinite loop? What should be done about it?

Design issues in `main.cpp`, `graph.hpp`, and `graph.cpp`

- Why is it useful for `Graph` to know the file name?
- If both `infile` and `fname` are passed as parameters to `Graph()`, the precondition that stream `infile` is opened on file `fname` cannot be checked. Why is this undesirable?
- What are the consequences of moving the file-opening code from `run()` to:
  - `main.cpp`, just after the call to `banner()`?
  - To the `Graph` constructor?
  - To a new controller class?
- Why is there a potential infinite loop in the `Graph` constructor? What should be done to fix it?

row.hpp

Points to note:

- This file contains two *tightly coupled* classes, `Cell` and `Row`.
- The line `friend class Row` in `Cell` gives `Row` permission to access private data and methods of `Cell`.
- A class can give friendship. It cannot take friendship.
- The `Cell` constructor combines two operations that could be separated:
  1. It creates a new `Item` from a C-string and an integer;
  2. It creates a new fully initialized `Cell` containing as `data` a pointer to the newly-created `Item`.
- A `Row` has a `head` that points to the first `Cell` in a linked list.

row.cpp

Points to note:

- There is some clever coding in the `Row` constructor.
  Is this a good design?
- The destructor in `Row` deletes the entire linked list of `Cell`s.
  Why shouldn't this be done in the `Cell` destructor?
- `insert` creates a new `Cell` and puts it on the linked list.
  Where does it go?
- In `Row::print()`, the code reaches through `Cell` into `Item::print()`.
  This violates the rule, *"Don't talk to strangers."*
  - Is it okay in this context?
  - Why or why not?
  - What would the alternative be? [Hint: Delegation.]

rowNest.hpp

This is an alternative definition of class Row with the same public interface and behavior but different internal structure.

Points to note:

- In row.hpp, Cell is a top-level class in which everything is private. The friend declaration allows Row to use it.
- In rowNest.hpp, Cell is declared as a private class inside of Row, but everything in Row is public. Since only Row can access the class name, nobody else can access it.
- In all other respects, row.hpp and rowNest.hpp are identical.
- To determine which is used, change the #include in graph.hpp.

Discussion of `row.hpp` vs. `rowNest.hpp`

What are the questions you should be asking yourself when deciding which version you prefer?

item.hpp

This is a data class. In C, one would use a `struct`, but C++ permits tighter semantic control.

Points to note:

- The fields are private. They are initialized by the constructor and never changed after that.
- The only use made of those fields is by `print()`. Hence there is no need even for getter functions.
- `Item` could have been defined as a subclass of class `Row`. What are the pros and cons of such a decision?