CPSC 427a: Object-Oriented Programming

Michael J. Fischer

Lecture 3
September 8, 2011

Example
    Header file
    Implementation File
    Main Program

Building Your Code

Insertion Sort Example (continued)

C++ version

See code demo `02-InsertionSortCpp` and following notes.

Header file

dataPack.hpp

```
#pragma once
```

A more efficient but non-standard replacement for include guards:

```
#ifndef DATAPACK_H
#define DATAPACK_H
// rest of header
#endif
```

Header file

class DataPack

```
class DataPack {
...
};
```

defines a new class named `DataPack`.

By convention, class names are capitalized.

Note the *required* semicolon following the closing brace.

If omitted, here's the error comment:

../datapack.hpp:11: error: new types may not be defined in a return type
../datapack.hpp:11: note: (perhaps a semicolon is missing after the
definition of 'DataPack')
../datapack.cpp:12: error: two or more data types in declaration of
'readData'

Class elements

- ▶ A class contains declarations for *data members* and *function members* (or *methods*).

- ▶ `int n;` declares a data member of type `int`.

- ▶ `int getN(){ return n; }` is a complete member function definition.

- ▶ `void sortData();` declares a member function that must be defined elsewhere.

- ▶ By convention, member names begin with lower case letters and are written in camelCase.

Inline functions

- Methods defined inside a class are *inline* (e.g., getN()).
- Inline functions are recompiled for every call.
- Inline avoids function call overhead but results in larger code size.
- `inline` keyword makes following function definition inline.
- Inline functions must be defined in the header (.hpp) file. Why?

Visibility

- ▶ The visibility of declared names can be controlled.
- ▶ `public:` declares that following names are visible outside of the class.
- ▶ `private:` restricts name visibility to this class.
- ▶ Public names define the interface to the class.
- ▶ Private names are for internal use, like local names in functions.

Constructor

A *constructor* is a special kind of method.

Automatically called whenever a new class instance is allocated.

Job is to initialize the raw data storage of the instance to become a valid representation of an initial data object.

In `dataPack` example, `store` must point to storage of `max` bytes, `n` of which are currently in use.

Constructor

```
DataPack(){
    n = 0;
    max = LENGTH;
    store = new BT[max]; cout << "Store allocated.\n";
    readData();
}
```

new does the job of malloc() in C.

cout is name of standard output stream (like stdout in C).

<< is output operator.

readData() is private function to read data set from user.

Design question: Is this a good idea?

Destructor

A *destructor* is a special kind of method.

Automatically called whenever a class instance about to be deallocated.

Job is to perform any final processing of the data object and to return any previously-allocated storage to the system.

In `dataPack` example, the storage block pointed to by `store` must be deallocated.

Destructor

```
~DataPack(){
    delete[] store;
    cout << "Store deallocated.\n";
}
```

Name of the destructor is class name prefixed with ~.

delete does the job of free() in C.

Empty square brackets [] are for deleting an array.

Implementation File

dataPack.cpp

Ordinary (non-inline) functions are defined in a separate
implementation file.

Function name must be prefixed with class name followed by `::` to
identify which class's member function is being defined.

Example: `DataPack::readData()` is the member function
`readData()` declared in class `DataPack`.

Implementation File

File I/O

C++ file I/O is described in Chapter 3 of textbook. Please read it.

`ifstream infile( filename );` creates and opens an input stream `infile`.

The Boolean expression `!infile` is true if the file failed to open.

This works because of a built-in coercion from type `ifstream` to type `bool`. (More later on coercions.)

`readData()` has access to the private parts of class `dataPack` and is responsible for maintaining their consistency.

main.cpp

As usual, the header file is included in each file that needs it:
`#include "datapack.hpp"`

`banner();` should be the first line of every program you write for this course. It helps debugging and identifies your output. (Remember to modify `tools.hpp` with your name as explained in Chapter 1 of textbook.)

Similarly, `bye();` should be the last line of your program before the return statement (if any).

The real work is done by the statements `DataPack theData;` and `theData.sortData();`. Everything else is just printout.

Building Your Code

Manual compiling and linking

One-line version

```
g++ -o isort main.cpp datapack.cpp tools.cpp
```

Separate compilation

```
g++ -c -o main.o main.cpp
g++ -c -o datapack.o datapack.cpp
g++ -c -o tools.o tools.cpp
g++ -o isort main.o datapack.o tools.o
```

Makefile

make is a tool to automate the build process.
It is controlled by a Makefile or makefile.

A minimal example:

```
OBJ = main.o datapack.o tools.o
isort: $(OBJ)
        g++ -o isort $(OBJ)
main.o: main.cpp datapack.hpp tools.hpp
datapack.o: datapack.cpp datapack.hpp tools.hpp
tools.o: tools.cpp tools.hpp
```

Note: The g++ line must begin with a *tab* character.

Integrated Development Environment (e.g., Eclipse)

Advantages

- ▶ Supports notion of *project* — all files needed for an application.
- ▶ Provides graphical interface to all aspects of code development.
- ▶ Automatically creates `makefile`.
- ▶ Builds project with a mouse click or keyboard shortcut.
- ▶ Analyzes code as it is being written. Provides helpful feedback.
- ▶ Allows easy navigation among project components.
- ▶ Error comments linked back to source code.

Integrated Development Environment (e.g., Eclipse)

Disadvantages

- ▶ Complicated to learn how to use — big learning curve.
- ▶ "Simple" things can become complicated for the non-expert (e.g., providing compiler flags) or making the font larger.
- ▶ Metadata can become inconsistent and difficult to repair.

Integrated Development Environment (e.g., Eclipse)

If you use Eclipse, before submitting your assignment, you should:

1. Copy your source code, test data, and make files from Eclipse to a separate submit directory *on the Zoo*.

2. Type make in that directory to make sure your program builds and runs correctly.

3. Submit the contents of your submit directory. Do not attempt to submit the entire Eclipse project. The hidden project and metadata files are not generally portable.