OOPP / C++: L1 Example: DataPack

Alice E. Fischer

August 28, 2016

main.cpp

pack.hpp

pack.cpp

Insertion Sort

A main program instantiates a class, then calls its functions.
File: main.cpp, with line numbers in parentheses.

- This program reads a file of data, sorts it, and writes output to cout.
- The work is done using the DataPack class.
- (5) By including `"tools.hpp"`, we indirectly include all the C and C++ header files that we are likely to need.
- The tools header also contains the `using namespace std;` declaration, which allows us to use all the standard functions without writing `std::` in front of each function name.

The Skeleton of main()

- (5, 6) We include the header files for the classes we need.
- (26, 31) The main function begins and ends exactly as in C.
- (27, 29) In this course, please call `banner()` and `bye()` at the beginning and end of every program. This makes it easier for me to grade your work.
- (28) In this course `main()`should call a series of unit-test functions, one per module.
- (8–24) This is a unit test function for the DataPack module.

A Unit Test

A unit test calls every function in the class, and tests every line of code that it is possible to test. The output follows the code.

- Each test has three parts:
  - Explain what is being tested, and why.
  - Explain what the output should be.
  - Test it and produce output.
- Lines 9–12 interact with the user to get a file name. By using type string, we can read a name of any length safely.
- Lines 14–16 test the constructor and print a debugging comment.
- Lines 18–19 test the print function.
- Lines 21–22 test sorting.
- Line 23 tests the output operator.

When you write your own code, model your unit tests after this one.

Storage allocation and management.

- This function declares two local variables: a string named `filename` (line 11) and a DataPack named `theData` (line 15).
- Both are created by declarations, so they will be allocated on the runtime stack which does not need explicit memory management.
- We instantiate the DataPack class (line 15) by calling its constructor with arguments; the file name and the maximum number of data items to read.
- When control leaves the enclosing block (line 24), these objects will be deallocated.
- Objects can also be created by calling `new`; new objects are allocated in heap memory and must be managed. Avoid using `new` unless you need it.

The DataPack Class Declaration : pack.hpp

A header file declares the class.

- A block comment at the top identifies the file name, author, and dates of creation and improvement.
- This file contains the class declaration and other things related to the class.
- (6) A pragma is advice to the C++ preprocessor. This one says that the header file should never be included twice in the same compilation module. DO NOT PUT pragma statements in .cpp files.
- (7) We include the tools header file; it includes several other header files that are normally needed.
- Header files contain only type declarations and prototypes. Object must all be in .cpp files.

The Class Declaration, continued

- (9) This typedef defines the type of the data we will process. By changing `float` to some other type, the entire program will work to sort the other type of data.
- (10) The `#define` supplies a default length for the DataPack. This will be used as the default length of the array if no parameter is supplied when the class is instantiated. The default is supplied only in the .hpp file, not in the .cpp.
- (12, 30) A class begins by declaring its name and ends with a semicolon.
- (13, 21) Within the class are the private and public parts. Each keyword applies to all the following lines until the next occurs.
- If the protection level is NOT specified, it defaults to private.

Class Data and Methods

- (14) Always declare your data to be private.
- (15–18) There are 4 data members: a filename, a dynamically allocated array of BT's and the two integers needed to manage it.
- (15) A data array may be empty, full, or partially full. We need a variable to store the fill-level.
- (16) Unlike Java arrays, a C++ array does not "store" its own length. This must be stored separately.
- (17) We intend to allocate the data array dynamically, so the type of this member is a *pointer* to the generic base type BT.
- (20) There is one private function, `readData()`. This function was defined to shorten and clarify the code in the class constructor. It is not intended for use outside the class, and is, therefore, not public.

Constructors

- (17) The class constructor and destructor are public. Function members are usually, but not always, public.
- A constructor is called automatically every time you declare or new a class instance.
- If you do not define a constructor, a do-nothing constructor will be supplied by default.
- A constructor has the same name as the class and no return type.
- Most classes have a constructor with parameters. The parameters provide a path to bring data into new class objects. This is the proper and safe way to "populate" an object.
- A class can have several constructors with different combinations of parameters.

The DataPack Constructor

- The responsibility of a constructor is to initialize the data members of an object, leaving it in an internally consistent state and ready for use.
- The DataPack constructor is named `DataPack`. Its prototype is here (23) and the definition is in the .cpp file (lines 10. . . 17)
- If no length parameter is supplied when the constructor is called, `LENGTH` will be used.
- (.cpp 11-12) This constructor allocates a dynamic array of BT's and sets max = the length of that array. The number of items currently stored in the array is set by `readData()` (.cpp line 36).

The Constructor's Helper: readData()

The constructor delegates most of the construction work to the `readData()` function that is prototyped on line 20 and defined in the .cpp file on lines 23–38.

- (25) `readData()` instantiates the input stream and opens the input file.
- (26) It checks for a valid stream. This check is necessary. Always incorporate the name of the missing file in the error comment.
- Lines 29–30 set up a pair of pointers for use in the following loop: end points at the first slot that is NOT in the array. The cursor is a scanning pointer that starts at the beginning and moves to the end of the array.
- Processing an array with pointers is slightly more efficient than using subscripts.

readData(), continued

- Line 32 reads an item into the slot under the cursor.
- Input is generic: the kind of data read will be the kind that is supposed to be stored in the array.
- (33) EOF must be checked after (not before) attempting to read data. the stream function `good()` returns false if there is a read error or an end of file.
- (36) Pointer subtraction is used to calculate the number of items stored in the array.
- (37) Clean up after yourself. Close your files.
- Line 35 differentiates between eof and error. End of file is normal, a read error is fatal.
- When eof occurs, we calculate the number of data items in the array, store it in a class data member, and close the file.

The Destructor

- A class has exactly one destructor named `~ClassName()`.
- If you do not declare one, a do-nothing destructor will be supplied automatically.
- The destructor is called automatically every time a declared object goes out of scope.
- If an object, or part of an object, is allocated using `new`, the destructor must be called explicitly.
- The task of a destructor is to deallocate all dynamic memory attached to a class instance.
- On line 27, we deallocate the dynamic BT array.

Inline and Out-of-line Functions

- The accessor function (line 27) and the destructor (line 24) are one-line functions. They are defined fully within the class, as inline functions, because inline functions are more efficient.
- You do not need to write the "inline" keyword. Any function that is fully-defined within the class is inline. Outside the class (after the final ':') you must write `inline` explicitly.
- The remaining two functions (lines 32–34) are too long and too complex to be inline, so they are prototyped in the .hpp and defined in the matching .cpp file.

Class Functions

Lines (27–29) declare the public methods supplied by this class.

- Line 27 is an inline accessor function (a "getter"). Its purpose is to give read-only access to a private data member.
- No "getters" are provided for the other data members because they are not the "business" of any outside code. All operations
- No "setters" are provided for the data members because they breach privacy and should almost always be avoided.
- Line 29 is the prototype for a non-inline sort function, defined in the .cpp file on lines 50–66.
- Line 28 is the prototype for a print function, defined in the .cpp file on lines 42–45. It is used in conjunction with the inline operator definition on lines 33-36 of the .hpp file.

The Output Operator

The output operator, << is defined as a global function in C++, and is predefined for all the basic types.

- Being a global function (not a class function) allows us to extend the operator to work with programmer-defined types. This is highly desirable because << is very convenient to use.
- The syntax for `operator<<` must conform to the pre-defined methods for the operator.
- (Lines 33–36) To define `operator<<` for a class, add a simple inline function after the semicolon. This function is almost completely the same for every class: only the name of the class differs.
- The code simply calls the public class function, `print()`, then returns the stream parameter. This code assumes that your `print()` function also returns an ostream&.

The DataPack Class Definition : pack.cpp

The .cpp file defines functions that are too long to be inline.

- Line 7 brings in the class declaration and constants.
- The rest of this file gives definitions for the class functions that are not inline.
- A series of comments above the function definition should give:
  - A dashed visual divider line. PLEASE!
  - A brief description of the purpose of the function.
  - Preconditions and error return values, if any.
  - Anything else that is non-obvious about the parameters or the return value.

printData()

Every class you write should have a method for `print()`.

- (42) The first line of each function definition gives the return type followed by the class name and a couble colon. The class name is needed because a code file COULD have functions belonging to more than one class.
- (43) Next comes the name of the function and its parameters.
- The word "const" indicates that this function does not change any member variables.
- The main task of `print()` is to output all the items in the array, with appropriate labelling.
- The task of formatting the collection data is delegated to the method defined for the BT class.
- All of the built-in types and classes have definitions for the operator <<.

sort()

- (50) This function is part of the DataPack class.
- (50–51) It has no parameters and no return value. That means it uses and changes the member variables of the DataPack.
- The function is not a “const” function because it modifies the data in the array.

The insertion sort algorithm.

This is the fastest way to sort a short array (up to about 25 items on current hardware).

- (57–65) It makes $n - 1$ passes over the array to sort n items.
- (59) On each pass, newcomer = the next unsorted item.
- (60) Removing the newcomer from the array leaves a hole.
- (51–54) We scan backwards from the hole to the head of the array, looking for the right place to store the newcomer.
- (53) At each step of the scan, the adjacent data item is moved into the hole, and the hole moves one slot to the left.
- (55) When the insertion slot is found, the newcomer is stored in the hole.