CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 6
February 4, 2016

Notes on PS1

Improvements to already-submitted assignments

- `main()` does not require a return statement in c++11.
- Error messages from `Fatal()` are more useful if they are all different.
- Good to check age for reasonableness.
- To check validity of `cin >> fname;`, test `cin.good()` and/or `cin.fail()`, *not* `stream.good()`.
- No need for `stringstream` in this assignment.
- The variable in which the birth year is stored should be a numeric type, not a string. Let `>>` do the conversion, don't use `stoi()`. Why?

Improvements (cont.)

- To write integer `age` to output, just use `cout << age;` no need to use `to_string()`.
- Write output directly to `cout`; don't write to a string first. (This isn't Java.)
- Style: Put `main()` definition before `run()` in `main.cpp`.
- Use conditional expression to make minor output changes: `cout << ( age>1 ? "years" : "year") << " old";`
- Don't put an unnecessary `else` clause following a `throw`.
- Properly intent your code.
- Good style to make a function `current_year()` to compute the year. Why?

Improvements (cont.)

- No need to set `struct tm` pointer to `NULL`.
- In c++11, use `nullptr` instead of `NULL`.
- Remember to test your code with bad input as well as good!

Brackets Example

Code demo

The 06-BracketsCpp demo contains three interesting `class` declarations and illustrates the use of constructors, destructors, and dynamic memory management.

In the remainder of the lecture was spent going through the code line by line to see what it does and how the parts all fit together.

We briefly summarize below some of the points made during lecture.

Token class

Major points:

1. `enum` is used to encode the bracket type (round, square, etc.) and the sense of the bracket (left, right).
2. `ch` is the character representing the bracket, used for printing.
3. `classify()` is a private function.
4. The definitions of `print()` and `operator<<` follow our usual paradigms.
5. The `Token` constructor initializes all data members.

Stack class

Major points:

1. `T` is the element type of the stack. This code implements a stack of `Token`. (See `typedef` declaration.)
2. Storage for stack is dynamically allocated in the constructor using `new[]` and deleted in the destructor using `delete[]`.
3. The empty square brackets are needed for both `new` and `delete` since the stack is an array.
4. `delete[]` calls the destructor of each `Token` on the stack. Okay here because the token destructor is null.
5. `push()` grows stack by creating new stack of twice the size, copying contents of the old stack into the new, and deleting the old stack.

Brackets class

Major points:

1. Data member `stk` is dynamically allocated in the constructor and deleted in the destructor, but it is an object, not an array, and does *not* use the `[]`-forms of `new` and `delete`.
2. `in.get(ch)` reads the next character without skipping whitespace. There are other ways to do this as well.
3. If read is `!in.good()`, we `break` from the loop and do further tests to find the cause.
4. `mismatch()` uses the `eofile` argument to distinguish two different cases. Is this good design?

Main file

Major points:

1. `main()` follows our usual pattern, except that it passes `argc` and `argv` on to the function `run()`, which handles the command line arguments.
2. `run()` opens the input file and passes the stream `in` to `analyze()`.
3. The stream was not opened in a constructor, so there is no corresponding destructor for the matching `close()`. If an error is thrown, the stream will not be closed (except for the automatic cleanup that happens when a program exits).
4. Question: Which is better, to pass the file name or an open stream? Why?