CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 8
February 11, 2016

Notes on PS2

Storage Management

Notes on PS2

vector

`vector<T>` is one kind of **container** in the standard C++ library.

- It is used to store a collection of objects of type `T`.
- It is like the C array `T myvec[]` except that it grows as needed, manages its own storage, and never overflows.
- `T` can be any copyable type or class.
- You will be using `vector<string>` and `vector<Score>`.
- By default, a `vector` starts empty and grows as needed.
- Use `myvec.push_back(item)` to append `item` to the end of vector `myvec`.
- `myvec.size()` returns the number of items in `myvec`.
- Vectors support many other operations, including subscript.
- You must `#include <vector>` to use it.

Algorithms

The standard C++ library contains many useful algorithms.

To sort an object `vsc` of type `vector<Score>`, write

```
sort(vsc.begin(), vsc.end());
```
[1]

You must somehow tell `sort` how to compare two scores. Do this by defining

```
bool operator<(const Score& rhs) const;
```

in class `Score`.

It should return `true` if `*this` should come before `rhs` in the sorted order and `false` otherwise.

You must `#include <algorithm>` to use the sort functions.

[1] `stable_sort()` can be used instead. It's guaranteed not to reorder equal elements.

Delegation

Delegation is when a function passes control to a delegate function in another class to carry out a task for which the other class is the expert.

In the assignment, both `SixOh` and `Score` should have `print()` functions (and corresponding `operator<<()` extensions).

When `SixOh::print()` wants to print a `Score` object `sc`, it should delegate that task to `Score::print()` by call `sc.print()` (or do `out << sc;`) rather than attempting to reach into the `Score` object to grab the individual data members.

Principle: Don't talk to strangers.

Manipulators

Manipulators are a class of objects that, when sent to a stream, change the behavior of the stream.

`setw()` sets the output field width to its argument, useful for getting numbers to line up in a column.

Example:

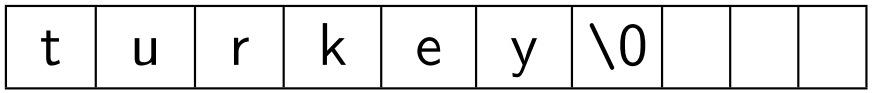
```
int x = 23;
cout << setw(5) << x;
```

prints 23 in a field of width 5, producing the printed output ␣␣␣23. (Here, ␣ represents a space.)

To use manipulators, you must `#include <iomanip>`.

C and C++ strings

A C string is a null-terminated sequence of characters, stored in a `char` array, e.g.,



A C++ string is an object of type `string`, defined by invoking `#include <string>`. Internally, it has a buffer containing a C-string along with its current length and the length of the buffer.

C++ strings are not length limited but grow as needed. The implementation takes care of managing storage and preventing the **buffer overruns** so common with C strings.

The C++ programmer needs to be able to deal with both kinds of strings. Many system calls require C-strings as paramenters or produce C-strings as results.

Converting between C and C++ strings

Let `str` be a C++ string. Then `str.data()` returns a pointer of type `const char*` to the internal buffer that contains the C-string. (The older form `str.c_str()` also works.)

Given a C string `s`, one can construct a new C++ string initialized to `s` by using the `string(s)` constructor, e.g., `string str(s)` or `new string(s)`.

For an existing C++ string `str`, one can use assignment to store a C string `s` into it, e.g., `str = s;`.

Range-based loops

C++ now has a kind of "for each" loop called a **range-based loop**.

For example, if `vec` is a vector of element-type `T`, then one can iterate through `vect` by writing

`for (const T& t : vec) { ... }`.

For each element in `vec`, the name `t` is bound to the element and the loop body is executed.

For example, `for (const T& t : vec) { cout << t; }` prints each element of `vec`.

String streams

A string stream is a kind of stream that reads from or writes to a string rather than from/to a device or file.

To the programmer, a string stream can be used just like any other stream. For example, to read an `int x` from an input string stream `iss`, one simply writes `iss >> x`.

Flag `eof` is set when the stream attempts to read beyond the end of the string. Flag `fail` is set if a requested data conversion fails, e.g., requesting to read an `int` when the next non-whitespace character in the stream is not a digit or '+' or '-'.

To use a string stream, you need to `#include <sstream>`. The input string stream class is called `istringstream`.

Reading a file a line at a time

Reading and processing a file line by line makes it easier to identify the location of bad data in a file and to skip a bad data record and go on (when that is deemed to be desirable).

String streams can be used for line-by-line file reading as follows:

1. Read a line from the input file into a `string textline` and check for success.
2. Create an `istringstream iss` initialized with `textline`.
3. Read and parse the data from `iss`. Use `iss.good()`, `iss.fail()`, and `iss.eof()` as usual in order to detect bad data and know when to stop.

For PS2, you should read and process the `.csv` input file a line at a time.

Storage Management

Objects and storage

Objects have several properties:

- A **type**. This determines the size and encoding of the allowable **data values**.
- A **name**. This is one way to access the object.
- A **storage area**. This is a block of memory big enough to hold any legal value of the specified type.
- A **lifetime**. This is the time span between an object's creation and its demise. Data left behind in an object's storage area after it has died is unpredictable and shouldn't be used.
- A **storage class**. This determines the lifetime of the object, where the storage area is located in memory, and how it is managed.

Example of an object

Declaration: `int n = 123;`

This declares an object of type `int`, name `n`, and an `int`-sized storage area, which will be initialized to 123. It's lifetime begins when the declaration is executed and ends on exit from the enclosing block. The storage class is `auto` (stack).

The unary operator `sizeof` returns the storage size (in bytes).

`sizeof` can take either an expression or a parentheses-enclosed type name, e.g., `sizeof n` or `sizeof(int)`.

In case of an expression, the size of the result type is returned, e.g., `sizeof (n+2.5)` returns 8, which is the size of a `double` on my machine.

Name

An object may have one or more names, or none at all!

Not all names are created equal. A name may exist but not be visible in all contexts.

- It is not visible from outside of the block in which it is defined.
- For a class data member, the name's visibility may be restricted, e.g., by the `private` keyword.
- An object may have more than one name. This is called **aliasing**.
- An object may have no name at all. Such an object is called **anonymous**. It can only be accessed via a pointer or subscript.

References

A name can be given to an anonymous object at a later time.

```
#include <iostream>
using namespace std;
int main() {
  int* p;
  p = new int;  // Creates an anonymous int object
  *p = 3;       // Store 3 into the anonymous object
  cout << *p << endl;
  int& x = *p;  // Give the object the name x
  x = 4;
  cout << *p << " " << x << endl;
}
/* Output
3
4 4
*/
```