Strings, streams, and stringstreams

William Hendrix

Lecture 9

Outline

- String comparisons
- Dynamic arrays
- Streams
- File streams
- Stringstreams

Announcement: Puzzle Hunt

- Event where teams try to solve as many brain-twisting puzzles as possible
  - Similar to events held at Microsoft, Google, etc.

- Free food!
- Prizes!
- Spend time with other engineering students!
- Everyone welcome!
  - Programming experience not required

- 6-8pm on Thurs, Jan 30, in Wilkinson Lab
  - Sponsored by NU WiC
  - RSVP on NU WiC Facebook page

String operations

- Review
  - Declaration: `string str("initial value");`
  - Length: `str.length()`
  - Concatenation: `str1 + str2 + 42 + "text"`
  - Extraction: `str[0];  str.at(1);  str.substr(0, 5); str.substr(6);`
  - Input: `cin >> str; getline(str, cin);`
- Comparisons
  - Can use comparison operators with strings
    - E.g., `str == "apple"`, `str1 <= str2`
    - Ordered according to ASCII codes
  - **Pitfall:** Do not use comparisons between `char` arrays
    - At least one side should be a `string`
    - E.g., `"zebra" < "apple"` (unknown truth value)
      - Comparison happens between memory locations

Dynamic arrays

- Allows arrays to be sized and resized while program is running
- Somewhat more difficult to use
- Dynamic arrays have a 4-stage lifecycle
  - Declare
  - Allocate
  - Use
  - Deallocate
  - May allocate again after deallocating to change size
- To declare: `int* array;`
- To use: mostly the same as static array
  - Accessing values past the end of the array less likely to crash the program
    - **Pitfall:** be *very* careful that indices are in bounds

Allocation and deallocation

- After declaring but before using, array must be *allocated*
  - Operating system sets aside memory for array
  - Forgetting to allocate almost always causes a segmentation fault
  - Can fail if too large or not enough memory available
- C++ allocation
  - `array = new int[numElements];`
  - Can be combined with declaration
  - `numElements` can be any value or expression
- C allocation
  - `array = (int*) malloc(numElements * sizeof(int));`
  - `malloc`: takes # bytes, returns the array
  - `sizeof`: gives # of bytes for given type (1 value)
- C++ deallocation: `delete array;`
- C deallocation: `free(array);`
- Failing to deallocate arrays can fill up memory with unused arrays
  - Causes a "memory leak"

Streams

- Standard C++ interface for I/O
- Messages go into the stream
- Data flows out of the stream
- `cin`, `cout` (from `<iostream>`)
- Broadly split into input streams (`istream`) and output streams (`ostream`)
- Often buffered (I/O doesn't happen immediately)
- Main operations: `<<` (insertion) and `>>` (extraction)
  - Also support `stream.fail()`
  - Casting to a `bool` equivalent to `!stream.fail()`

File streams

- In preamble: `#include <fstream>`
- Reading a file: `ifstream`
  - To open the file: `ifstream in(“filename”);`
    - Or: `ifstream in; in.open(“filename”);`
  - `in.eof()` reports whether an error was caused by end of file
- Writing a file: `ofstream`
  - Syntax similar to `ifstream`
- When finished: `stream.close();`
- `fstream` can read and write
- Other useful functions:
  - `file.tellg():` report position in file (`long`)
  - `file.seekg(long):` set position in file
  - `in.peek():` returns next char without consuming it

Stringstreams

- Streams designed for reading and writing data to and from `string`
- In preamble: `#include <sstream>`
- To build a string: `ostringstream ostr;`
  - Can also initialize with a `string` or `char` array
    - `ostringstream ostr("initial");`
  - Build up with <<
  - Use `ostr.str()` to get the current string
- To parse a string: `istringstream istr;`
  - Combines well with `getline`: `getline(istr.str(), cin);`
  - Extract values with >>
  - Current position: `istr.tellg()`
  - Change position: `istr.seekg(pos)`

Tonight

Lab 2 is due Tuesday at noon

Recommended reading: Sections 4.1-4.7