Programming Abstractions

CS106B

Cynthia Lee

Today's Topics

Introducing C++

- C++ strings
  - C strings vs C++ string class
- Reading to/from a file

Abstract Data Types

- Stack

Next time:

- Finish stack (if we don't finish today)
- Queue ADT
- Start Map (Maps to continue Friday)

Strings in C++

STRING LITERAL VS STRING CLASS

CONCATENATION

STRING CLASS METHODS

Using cout and strings

```
int main(){
   string s = "ab";
   s = s + "cd";
   cout << s << endl;
   return 0;
}

int main(){
   string s = "ab" + "cd";
   cout << s << endl;
   return 0;
}
```

- This prints "abcd"
- The + operator concatenates strings in the way you'd expect.

- But…SURPRISE!…this one doesn't work.

C++ string objects and string literals

- In this class, we will interact with two types of strings:
  - String literals are just hard-coded string values:
    - `"hello!" "1234" "#nailedit"`
    - They have no methods that do things for us
    - Think of them like integer literals: you can't do `4.add(5); //no`
  - String objects are objects with lots of helpful methods and operators:
    - `string s;`
    - `string piece = s.substr(0,3);`
    - `s.append(t); //or, equivalently: s += t;`

C++ standard `string` object member functions (3.2)

Member function name	Description
`s.append(str)`	add text to the end of a string
`s.compare(str)`	return -1, 0, or 1 depending on relative ordering
`s.erase(index, length)`	delete text from a string starting at given index
`s.find(str)` `s.rfind(str)`	first or last index where the start of *str* appears in this string (**returns `string::npos` if not found**)
`s.insert(index, str)`	add text into a string at a given index
`s.length()` or `s.size()`	number of characters in this string
`s.replace(index, len, str)`	replaces *len* chars at given index with new text
`s.substr(start, length)` or `s.substr(start)`	the next *length* characters beginning at *start* (inclusive); if *length* omitted, grabs till end of string

```
string name = "Donald Knuth";
if (name.find("Knu") != string::npos) {
    name.erase(5, 6);
}
```

C++ standard `string` object member functions (3.2)

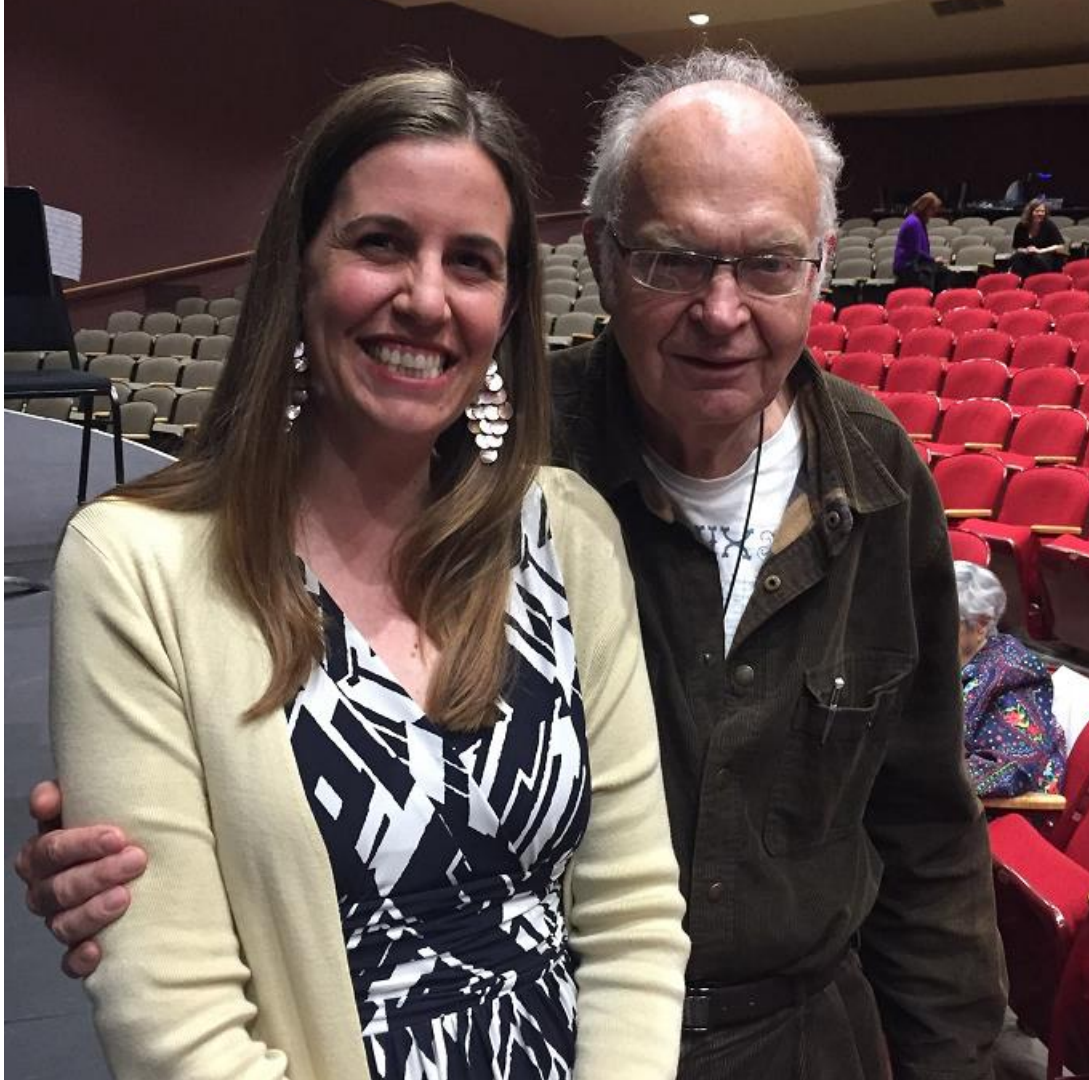
Member function name	Description
`s.append(str)`	add text to the end of a string
`s.compare(str)`	return -1, 0, or 1 depending on relative ordering
`s.erase(index, length)`	delete text from a string starting at given index
`s.find(str)` `s.rfind(str)`	first or last index where the start of *str* appears in this string (**returns `string::npos` if not found**)
`s.insert(index, str)`	add text into a string at a given index
`s.length()` or `s.size()`	number of characters in this string
`s.replace(index, len, str)`	replaces *len* chars at given index with new text
`s.substr(start, length)` or `s.substr(start)`	the next *length* characters beginning at *start* (inclusive); if *length* omitted, grabs till end of string

Exercise: Write a line of code that pulls out the number part of a string str of the form "(XXX)" where XXX is an integer with any number of digits (does not need to actually convert the number part from string to integer).

```
string numberPart = ____________________________________;
```

(Almost) true story:

Donald Knuth and I are BFF



Stanford library helpful string processing (*read* 3.7)

```
#include "strlib.h"
```

- Unlike the previous ones, these take the string as a parameter.

Function name	Description
`endsWith(`***str, suffix***`)` `startsWith(`***str, prefix***`)`	returns true if the given string begins or ends with the given prefix/suffix text

`integerToString(`***int***`)`
`realToString(`***double***`)`
`stringToInteger(`***str***`)`
`stringToReal(`***str***`)`
	returns a conversion between numbers and strings

`equalsIgnoreCase(`***s1, s2***`)`	true if s1 and s2 have same chars, ignoring casing
`toLowerCase(`***str***`)` `toUpperCase(`***str***`)`	returns an upper/lowercase version of a string
`trim(`***str***`)`	returns string with surrounding whitespace removed

```
if (startsWith(name, "Mr.")) {
    name += integerToString(age) + " years old";
}
```

Recap: C++ Stan**dar**d libraries vs. Stan**for**d libraries

- For **console input/output** we have two libraries:
  - `#include <iostream> // C++ standard library`
    - `cin, cout, endl`
  - `#include "simpio.h" // Stanford library`
    - `getInteger(), getLine() and lots more`

- For **string** handling we have two libraries:
  - `#include <string> // C++ standard library`
    - `string type, +, substr(), and more`
  - `#include "strlib.h" // Stanford library`
    - `Convert upper/lower case, convert int<->string, and more`

- For **file** handling there are similarly two libraries (more on Friday)

Stacks

New ADT: Stack

stack.h

```
template <typename ValueType> class Stack {
public:
   Stack();
   ~Stack();
   int size();
   bool isEmpty();
   void clear();
   void push(ValueType value);
   ValueType pop();
   ValueType peek();
   string toString();
private:
   -Redacted-

};
```

Source: http://www.flickr.com/photos/35237093334@N01/409465578/
Author: http://www.flickr.com/people/35237093334@N01 Peter Kazanjy]

Using a Stack to buffer file input

```
void mystery(ifstream& infile) {
    Stack<string> lines;
    string line;
    while (getline(infile,line)) {
        lines.push(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines.pop()
             << endl;
    }
}
```

What does this code do?

A. Prints all lines of a file to cout
B. Prints only the first line of a file to cout
C. Prints only the last line of a file to cout
D. Prints all lines of a file to cout in reverse
E. All/ none/ more

Using a Stack to buffer file input

```
void mystery(ifstream& infile) {
    Stack<string> lines;
    string line;
    while (getline(infile,line)) {
        li
    }
    infile
    while (!lines.isEmpty()) {
        cout << lines.pop()
             << endl;
    }
}
```

Why do I need Stack?
I could have done that with a Vector!

What does this code do?

A. Prints all lines of a file to cout
B. Prints only the first line of a file to cout
C. nly the last line of a ɔut
D. ll lines of a file to cout se
E. All/ none/ more

Stack or Vector?

```
void mystery(ifstream& infile) {
    Stack<string> lines;
    string line;
    while (getline(infile,line)) {
        lines.push(line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines.pop()
             << endl;
    }
}
```

```
Vector<string> lines;
```

```
lines.insert(lines.size(), line);
```

```
cout << lines[lines.size() - 1]
     << endl;
lines.remove(lines.size() - 1);
```

Vector version

```
void mystery(ifstream& infile) {
    Vector<string> lines;
    string line;
    while (getline(infile,line)) {
        lines.insert(lines.size(),line);
    }
    infile.close();
    while (!lines.isEmpty()) {
        cout << lines[lines.size() - 1]
             << endl;
        lines.remove(lines.size() - 1);
    }
}
```

This code isn't terrible, but it is harder to read quickly, and is probably more error prone.

For example, it would be easy to forget the "-1" when you remove "lines.size()-1".

Applications of Stacks

We've seen one (buffering input and giving it back in reverse—LIFO—order). What else are Stacks good for?

Operator Precedence and Syntax Trees

Ignoring operator precedence rules, how many distinct results are there to the following arithmetic expression?

- 3 * 3 + 3 * 3

Reverse Polish Notation

Ambiguities don't exist in RPN

Also called "postfix" because the operator goes after the operands

Postfix (RPN):

- 4 3 * 4 3 * +

Equivalent Infix:

- (4*3) + (4*3)

http://commons.wikimedia.org/wiki/File:Hewlett-Packard_48GX_Scientific_Graphing_Calculator.jpg

Reverse Polish Notation

This postfix expression:

- 4 3 * 7 2 5 * + +

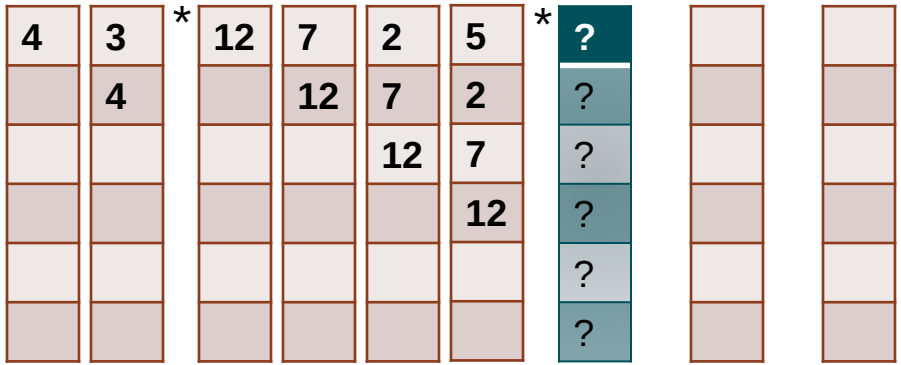
Is equivalent to this infix expression:

A. ((4*3) + (7*2)) + 5
B. (4*3) + ((7+2) + 5)
C. (4*3) + (7 + (2*5))
D. Other/none/more than one

http://commons.wikimedia.org/wiki/File:Hewlett-Packard_48GX_Scientific_Graphing_Calculator.jpg

Stacks and RPN

- Evaluate this expression with the help of a stack
  - Encounter a **number**? **PUSH** it
  - Encounter an **operator**? **POP** two numbers and **PUSH** result
- 4 3 * 7 2 5 * + +



Contents of the stack, reading from top down:

(A) 7, 12

(C) 10, 7,12

(D) 10, 5, 2, 7, 12

(E) Other

Stacks and RPN: What does that look like in code?

Evaluate this expression with the help of a stack

- Encounter a **number**: **PUSH** it
- Encounter an **operator**: **POP** two numbers and **PUSH** result

43*725*++

```
/* Note: this code assumes numbers are all 1 digit long */
bool evaluate(Stack<int>& memory, string instruction) {
    for (int i = 0; i < instruction.size(); i++) {
        if (isdigit(instruction[i])) {
            int value = instruction[i] - '0'; //convert char->int
            memory.push(value);
        } else if (isSupportedOperator(instruction[i])) {
            if (memory.size() < 2) return false;
            int second = memory.pop();
            int first = memory.pop();
            int result = compute(first, instruction[i], second);
            memory.push(result);
        } else {
            return false;
        }
    }
    return memory.size() == 1; //validity check
}
```