CSC 533 REVIEW SHEET: C++ Classes & Libraries

- `string` class `#include <string>`

  construct with no arguments, e.g., `string str1, str2;`

  member functions include:

```
int length();           // returns number of chars in string
char at(int index);     // returns character at specified index
                        // note: indexes start at 0
int find(string substring, int startpos = 0);
                        // returns position of first occurrence of
                        // substring, starting at startpos
                        // returns string::npos if not found
int find(char ch, int startpos = 0);
                        // similarly finds a character
string substr(int startpos, int size);
                        // returns substring starting at startpos,
                        // of length size
void insert(int startpos, string substring);
                        // inserts substring at startpos position
void delete(int startpos, int size);
                        // removes substring starting at startpos,
                        // of length size
char[] c_str();         // returns the underlying C-style string,
                        // needed for library routines/classes that
                        // require C-style strings (e.g., ifstream)
```

  operators include:

```
+                       // concatenation
=                       // assignment
== != < <= > >=         // relational operators (dictionary ordering)
```

EXAMPLE CODE:

```
string str1 = "foobar";                         // DECLARES & ASSIGNS str1
str1 = str1 + "oo";                             // SETS str1 = "foobaroo"
cout << str1.length() << endl;                  // OUTPUTS 8

for (int i = 0; i < str1.length(); i++) {       // DISPLAYS "ooraboof"
    cout << str1.at(str1.length()-i-1);         // (THE STRING REVERSED)
}
cout << endl;

cout << str1.find("oo") << endl;                // OUTPUTS 1
cout << str1.find("oo", 3) << endl;             // OUTPUTS 6
cout << str1.find("ooo") << endl;               // OUTPUTS string::npos
                                                //   (which is usually MAX_INT)
string str2;
cin >> str2;
if (str1.find(str2) < str1.length()) {          // DETERMINES IF str2 IS A
    cout << str1 << " contains " << str2 << endl; // SUBSTRING OF str1
}

str1 = str1.substr(0, 6);                       // SETS str1 = "foobar"

str1.insert(3, "lish");                         // SETS str1 = "foolishbar"

str1.erase(3, 4);                               // SETS str1 = "foobar"
```

- `vector` class (templated) `#include <vector>`

  construct with one argument, the vector size e.g., `vector<int> nums(100);`
  if no size is specified, 0 items will be allocated `vector<int> empty;`

  member functions include:

```
int size();       // returns size of vector
void resize(int newsize); // resizes the vector to newsize,
                          // copying any old entries

void push_back(const <TYPE> & item);
                        // adds item to end of vector
void pop_back();        // removes item at end of vector
<TYPE> & back();        // returns item at end of vector
```

  operators include:

```
[]                      // indexing (note: first item at index 0)
=                       // assignment
```

Note: `push_back`, `pop_back`, and `back` provide the three standard operations of a stack (push, pop, and top, respectively). Thus, you can use a vector when you want the behavior of a stack.

Also: `push_back` is useful when you want to read in a sequence of values and store them as you go. You can start with an empty vector, then repeatedly add to the end using `push_back`. Once values have been read in, you can access the elements using `[]`.

EXAMPLE CODE:

```
vector<int> nums(5);                            // DECLARES VECTOR OF 5 INTS
for (int i = 0; i < nums.size(); i++) {         // STORES SQUARES IN VECTOR
    nums[i] = i*i;                              // (0, 1, 4, 9, 16)
}

nums.resize(10);                                // RESIZES TO STORE 10 INTS
for (int j = 5, j < nums.size(); j++) {         // STORES SQUARES IN NEW SPOTS
    nums[j] = j*j;                              // (25, 36, 49, 64, 81)
}

int sum = 0;
for (int k = 0; k < nums.size(); k++) {         // SUMS UP ALL 10 SQUARES
    sum += nums[k];
}


vector<string> words;                           // CREATES EMPTY VECTOR
string input;
cin >> input;
while (input != "DONE") {                       // REPEATEDLY READS IN INPUT
    words.push_back(input);                     // AND ADDS TO END OF VECTOR
    cin >> input;                               // UNTIL "DONE" IS READ
}

for (int n = 0; n < words.size(); n++) {        // DISPLAYS ALL WORDS IN VECTOR
    cout << words[n] << endl;
}
```

```
// Demonstration program to show the use of strings and vectors.
//
// The program reads in strings and displays statistics, including the
// total number of words, total number of unique words, and longest and
// shortest word lengths.  It terminates on the END-OF-INPUT character
// (^Z for Windows, ^D for UNIX).
//
// Author:  Dave Reed
////////////////////////////////////////////////////////////////////

#include <iostream>                                 // NEEDED for cin, cout
#include <string>                                   // NEEDED for string
#include <vector>                                   // NEEDED for vector
using namespace std;

int main()
{
   int totalWords = 0;
   int longest = 0, shortest = 10000;               // ASSUMES NO STRING > 10000 CHARS
   vector<string> unique;                           // CREATES DEFAULT VECTOR, SIZE 0

    string str;                                     // READ STRINGS (DELIMITED BY
    while (cin >> str) {                            //   WHITESPACE) UNTIL ^Z
          totalWords++;                             // UPDATE TOTAL # OF WORDS

        if (str.length() > longest) {               // UPDATE LONGEST WORD IF NEC.
              longest = str.length();
          }
          if (str.length() < shortest) {            // UPDATE SHORTEST WORD IF NEC.
              shortest = str.length();
          }

        bool found = false;                         // SEARCH FOR WORD IN
          for (int i = 0; i < unique.size(); i++) {//   VECTOR OF UNIQUE WORDS
              if (unique[i] == str) {               //   SO FAR.
                    found = true;                   // SET FLAG AND EXIT LOOP
                    break;                          //   IF FOUND.
              }
          }
          if (!found) {                             // IF NOT FOUND,
              unique.push_back(str);                //   ADD TO END OF VECTOR
          }
   }

   cout << "Total number of words  = " << totalWords << endl;
   cout << "Number of unique words = " << unique.size() << endl;
   if (totalWords > 0) {
          cout << "The shortest word was length " << shortest << endl;
          cout << "The longest word was length  " << longest << endl;
   }

    return 0;
}
```

- `ifstream` and `ofstream` classes `#include <fstream>`

  `ifstream` is an input file stream (for reading data from a file)
  `ofstream` is an output file stream (for writing data to a file)

  construct with one argument, a C-style string, e.g.,

```
ifstream instr("foo.in");          ofstream ostr("foo.out");

string infile = "foo.in";          string outfile = "foo.out";
ifstream instr(infile.c_str());    ofstream outstr(outfile.c_str());
```

  can read from / write to files streams just like standard streams

```
instr >> x;                        outstr << "Howdy" << endl;
```

  when done with an fstream, should close it

```
instr.close();                     ostr.close();
```

  note: the `getline` function takes an input stream and a string variable as arguments, and reads an entire line of text from that file into the string, e.g.,

```
string line1, line2;
getline(cin, line);          // reads line1 from standard input
getline(instr, line2);       // reads line2 from file
```

```
// Demonstration program to show the use of input and output streams to
// read from and write to files.
//
// The program reads in lines of text from an input file (foo.txt) and
// echoes the lines to an output file (copy.txt), effectively copying
// the file one line at a time.
//
// Author:  Dave Reed

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    ifstream istr("foo.txt");  // OPEN INPUT STREAM, CONNECTED TO foo.txt
    ofstream ostr("copy.txt");    // OPEN OUTPUT STREAM, CONNECTED TO copy.txt

    string line;
    while (getline(istr, line)) { // REPEATEDLY READ LINE OF TEXT FROM FILE
        ostr << line << endl;     // ECHO LINE TO OUTPUT FILE (WITH endl)
    }

    istr.close();                 // CLOSE INPUT STREAM
    ostr.close();                 // CLOSE OUTPUT STREAM

    return 0;
}
```

- `istrstream` class `#include <strstream>`

  `istrstream` is an input string stream (useful for processing line-oriented text)

  construct with one argument, a C-style string, e.g.,

  ```
  istrstream instr("foo bar bizbaz");
  ```

  can read from string streams just like standard streams

  ```
  instr >> num1 >> num2;
  ```

```
// Demonstration program to show the use of input string streams to
// read and process line-oriented data.
//
// The program reads in lines of integers from the file nums.dat
// and displays the average of the integers on each line.
//
// Author:  Dave Reed
////////////////////////////////////////////////////////////////////

#include <iostream>                   // NEEDED FOR cin
#include <fstream>                    // NEEDED FOR ifstream
#include <strstream>                  // NEEDED for istrstream
#include <string>                     // NEEDED for string
using namespace std;

int main()
{
    ifstream infile("nums.dat");      // OPEN INPUT STREAM

    string str;
    while ( getline(infile, str) ) {  // REPEATEDLY READ LINE FROM FILE,
        istrstream istr(str.c_str()); // CREATE INPUT STRING STREAM TO
                                      // HOLD STRING CONTENTS (NOTE:
                                      // MUST CONVERT C++ STRING INTO
                                      // C-STYLE STRING)

        int num, count = 0, sum = 0;
        while (istr >> num) {         // CAN READ FROM INPUT STRING
            sum += num;               // JUST LIKE ANY OTHER STREAM
            count++;
        }

        if (count == 0) {
            cout << "There were no numbers to average." << endl;
        }
        else {
            cout << "The average of the " << count << " numbers is "
                 << (double)sum/count << endl;
        }
    }

    return 0;
}
```

`cctype` library of routines for manipulating and testing characters

`bool isalpha(ch)`	returns true if ch is alphanumeric
`bool isupper(ch)`	returns true if ch is an upper case letter ('A'..'Z')
`bool islower(ch)`	returns true if ch is a lower case letter ('a'..'z')
`bool isdigit(ch)`	returns true if ch is a digit ('0'..'9')
`bool isspace(ch)`	returns true if ch is whitespace (space, tab, CR)
`bool isspunct(ch)`	returns true if ch is a punctuation mark
`char toupper(ch)`	returns upper case version of ch (or just ch if not a letter)
`char tolower(ch)`	returns lower case version of ch (or just ch if not a letter)

`iomanip` library of I/O manipulators

`setiosflags(ios::fixed)`	sets flags so that real values are displayed with a fixed number of digits
`setprecision(N)`	sets flags so that real values are displayed with N digits to the right of the decimal place
`setw(N)`	sets flags so that next value displayed will be right-justified in a field of N characters (useful for aligning columns)

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;

int main()
{
    ifstream istr("words.txt");              // OPEN INPUT STREAM, CONNECTED TO words.txt

    int spaceCount = 0, digitCount = 0,      // INITIALIZE COUNTERS
        alphaCount = 0, totalCount = 0;

    string line;
    while (getline(istr, line)) {            // REPEATEDLY READ LINE OF TEXT FROM FILE
        for (int i = 0; i < line.length(); i++) {  // FOR EACH CHARACTER IN THE LINE,
            if (isalpha(line.at(i))) {             //   IF IT'S A LETTER, INCR. COUNT
                alphaCount++;                      //
            }                                      //
            else if (isdigit(line.at(i))) {        //   IF IT'S A DIGIT, INCR. COUNT
                digitCount++;                      //
            }                                      //
            else if (isspace(line.at(i))) {        //   IF IT'S A SPACE, INCR. COUNT
                spaceCount++;                      //
            }                                      //
        }
        totalCount += line.length();         // INCREMENT TOTAL CHAR COUNT
    }

    istr.close();                            // CLOSE INPUT STREAM

    cout << setiosflags(ios::fixed) << setprecision(1);  // FIX TO 1 DIGIT TO RIGHT
                                                         // OF DECIMAL PLACE
    cout << "Total # of chars: " << totalCount << endl;
    cout << "  # of letters: " << setw(5) << alphaCount  // DISPLAY ALIGNED COUNTS
         << "  (" << (alphaCount * 100.0)/totalCount << "%)" << endl;
    cout << "  # of digits : " << setw(5) << digitCount
         << "  (" << (digitCount * 100.0)/totalCount << "%)" << endl;
    cout << "  # of spaces : " << setw(5) << spaceCount
         << "  (" << (spaceCount * 100.0)/totalCount << "%)" << endl;

    return 0;
}
```