CS103: Final Practice
Last Updated: 04/19/24

Note: The EXAM is printed SINGLE-SIDED!

Name: ___Solutions_____

Student ID: _____ Email: _____@usc.edu

I understand I may ONLY access Gradescope, Codio (no past assignments), Blackboard, and EdStem (no external editors, ChatGPT, StackOverflow, etc.): _____ (initials)

Time Submitted: _____ (if leaving the testing location early write the time that you left in the blank)

Page	2	3-4	5-6	Total
Points	-	-	-	-pts
Suggested Time	10 min.	15 min.	15min.	40min.

Instructions:

- Use the question prompts on the following pages to **fill in your answers on Gradescope..Sp24 Fi. (Link to Gradescope / Codio on Blackboard.. Assignments). You must write your code in Codio (use of outside editors is an academic integrity violation) and then paste/upload your code to Gradescope.**
- Check the screen at the front of your testing room periodically for clarifications/announcements.
- The Gradescope answer form will mark your submission as LATE after 8:30 and it will **NOT be accepted**. Be sure to submit by then.

Common Reference Functions:

- You **MAY NOT** use any functions from the <algorithm> or <string> (C++ string) library.
- You may use functions described below:

<vector> functions:

- `.size()` – Returns the number of items in the vector.
- `.push_back(val)` – Adds a new element, `val`, to the back, resizing if necessary
- `.pop_back()` – Removes the back element
- `.resize(new_size)` – Resizes the vector to the given size.

<string> functions (for the `string` type):

- `.size()` – Returns the number of characters in the string.
- `.substr(int first)` – Returns the substring starting at index `first` through the end of the string.
- `.substr(int first, int num)` – Returns the substring of `num` characters starting at index `first`.

Examples:

```
string s1 = "abcd";
s1.size();      // returns 4
s1.substr(1);   // returns "bcd"
s1.substr(1,2); // returns "bc".
```

1. Examine the code for class **Thing** below and how it is used in main(), then for each prototype indicate if it would be **correct and necessary** to add it to the class to ensure the code in main() **will compile and run without any errors or leaks.** Do not worry about the implementation, just indicate if the function is one that should be prototyped in the class

```
// Assume appropriate #includes
class Thing {
private:
    int x;
    string* s;
public:
    Thing(string mys) : x(5)
      { s = new string(mys);  }
    // What member functions are necessary
    // to support the use of the class in main()
};
int main() {
    Thing t1("cs103"), t2("final"), t3("exam");
    Thing t4;
    t4 = t1 + t2 + t3;
    if( t3 < t1 ){
        t2 += " is almost over";
    }
    cout << t2 << " " << t4 << endl;
    return 0;
}
```

For each prototype, indicate **Y (Yes)** if it should be included or **N (No)** if it should not.
Only say Yes if the function/prototype is NECESSARY and CORRECT for what is shown in `main()`.

1.1. **Y / N:** `~Thing();`
1.2. **Y / N:** `Thing& operator=(Thing& src, Thing& dest);`
1.3. **Y / N:** `Thing operator+(const Thing& op1, const Thing& op2) const;`
1.4. **Y / N:** `Thing& operator+(const Thing& op1) const;`
1.5. **Y / N:** `Thing operator+(const Thing& op1) const;`
1.6. **Y / N:** `std::ostream& operator<<(std::ostream& ostr) const;`
1.7. **Y / N:** `friend std::ostream& operator<<(std::ostream& ostr, const Thing& t);`
1.8. **Y / N:** `Thing& operator+=(const char* s);`
1.9. **Y / N:** `Thing& operator+=(Thing& self, string s) const;`
1.10. **Y / N:** `bool operator<(const Thing& rhs) const;`

2. Examine the recursive function below.

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

string f1(string sa, string sb)
{
    if(sa.size() < 2) {
        cout << "B " << sb << endl;
        return sb;
    }
    else {
        string sc = f1(sa.substr(2), sb + "*");
        cout << "A " << sc << endl;
        sc += sa[0];
        return sc;
    }
}
int main()
{
    string s, instr;
    cin >> instr;
    s = f1(instr, "");
    cout << s << endl;
    return 0;
}
```

Suppose the user enters: `wxyz` to the `cin` prompt. Answer the following questions:

2.1. Which of the following is the first line of output that will be printed by the program?

❑ `A **` ✅ `B **` ❑ `B` ❑ `A *` ❑ `A wx` ❑ `B yz` ❑ None of these

2.2. Which of the following is the second line of output that will be printed by the program?

✅ `A **` ❑ `B **` ❑ `B` ❑ `A **y` ❑ `A wx` ❑ `B yz` ❑ None of these

2.3. Which of the following is the last line of output that will be printed by the program?

❑ `**wx` ❑ `**yz` ❑ `wx` ❑ `yz` ✅ `**yw` ❑ `**xw` ❑ None of these

3. Consider a program that must read in a CSV (Comma Separated Value) file, where data on each line corresponds to a row, and column separation is indicated by a comma (`,`) and then be able to look up entries with a spreadsheet-like naming convention (A3, B1, C2), etc.

```
     A         B         C     ...   Z
0: <datA0>, <datB0>, <datC0>,     <datZ0>
1: <datA1>, <datB1>, <datC1>,     <datZ1>
2: <datA2>, <datB2>, <datC2>,     <datZ2>
```

Your program will be run with a filename passed on the command line (e.g. `./prog parsingprac1.in`). The program must read in the data from the given file, store it in an appropriate data structure, and then allow the user to type in column/row pairs like `A2` or `B 3` or `C 1` from the keyboard and then output the value from the file that was at that location. Note: A capital letter will be provided first indicating the column followed by an integer row. Also, any amount of whitespace may occur between input values. Continue this program until the keyboard input fails.

`parsingprac1.in`	**Desired Look-up Naming Convention**

`cfolt,29,BUAD`
`acote,40,CSCI`
`redekopp,45,CECS`
	`  A  B  C`
	`0: cfolt  29  BUAD`
	`1: acote  40  CSCI`
	`2: redekopp  45  CECS`

The code has been started on the next page. You must choose the best/correct code for each blank in the program.

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

int main(int argc, char* argv[]) {
    ifstream ifile(____);                  // Blank 1
    string line, field;
    ____________ data;                     // Blank 2
    while(1) {
        getline(______, line);             // Blank 3
        if(____________) break;            // Blank 4
        vector<string> row;
        stringstream ss1(line);
        while(________________) {          // Blank 5
            stringstream ss2(field);
            row.push_back(field);
        }
        ______________________;            // Blank 6
    }
    ifile.close();
    int r; char c;
    while(____________) {                  // Blank 7
        int col = _________;               // Blank 8
        cout << data[r][col] << endl;
    }
    return 0;
}
```

1. **Blank 1**
   - ❑ `"parsingprac1.in"`
   - ❑ `char*`
   - ✅ `argv[1]`
   - ❑ **Blank - No code needed**

2. **Blank 2**
   - ❑ `vector<string>`
   - ❑ `string`
   - ✅ `vector<vector<string> >`

3. **Blank 3**
   - ✅ `ifile` ❑ `stringstream` ❑ `cin`

4. **Blank 4**
   - ❑ `ifile.getline(line, 80)`
   - ❑ `getline(ifile, line)`
   - ✅ `ifile.fail()`
   - ❑ `ifile >> line`

5. **Blank 5**
   - ✅ `getline(ss1,field,',')`
   - ❑ `getline(ifile,field,',')`
   - ❑ `getline(ss1,',')`
   - ❑ `ifile >> field`

6. **Blank 6**
   - ❑ `data.back() = row;`
   - ✅ `data.push_back(row);`
   - ❑ `data.push_back(field);`

7. **Blank 7**
   - ❑ `getline(ifile, c, r)`
   - ❑ `getline(cin, c)`
   - ✅ `cin >> c >> r`
   - ❑ `ss1 >> c >> r`

8. **Blank 8**
   - ❑ `toInteger(c)`
   - ❑ `static_cast<int>(c)`
   - ✅ `c - 'A';`
   - ❑ `c - '0';`