1 Searching for Symbols in ASCII Art [24]

In this problem we will search a large ASCII Art canvas for matches to a target pattern. For example, given the ASCII Art Tic Tac Toe game on the right, we can search for the 'X' symbol on the board. Or the 'O' symbol. Or the height=5 column of '│' characters that make the vertical bar of the Tic Tac Toe board. You will write a function named `search_for_symbols` that takes in 2 arguments, `canvas` and `symbol`. Both inputs are STL vectors of STL strings. The function should return all locations of that pattern or symbol on the canvas. The return type will be an STL vector of positions where each position is an STL vector of integers. Note: Your function program should be general, and work with any size canvas and any size symbol.

In the example below we search for all of the 'O' symbols on the board:

```
std::vector<std::string> symbol;
symbol.push_back(" *** ");
symbol.push_back(" * *");
symbol.push_back(" * *");
symbol.push_back(" * *");
symbol.push_back(" *** ");
std::vector<std::vector<int> > positions = search_for_symbols(tic_tac_toe_canvas,symbol);
for (int i = 0; i < positions.size(); i++) {
    assert (positions[i].size() == 2);
    std::cout << "An O is located at: " << positions[i][0] << "," << positions[i][1] << std::endl;
}
```

And this is the expected output:

```
An O is located at: 0,6
An O is located at: 12,12
```

1.1 Assumptions & Error Checking & Realistic Expectations [5]

You will implement the `search_for_symbols` function on the next page. You will not need to do any error checking in your function. But let’s pause and consider what assumptions you are making about the input and output. Write 3-4 well-written and concise sentences. You may find it easier to write the function on the next page, then return to answer this part.

Solution: This code will only find matches if all foreground and background characters match exactly between the canvas and the symbol strings. The code also assumes that both the canvas and the symbol are rectangular – that is, the width of each row is the same. Specifically we assume that the empty space at the end of the row is explicitly represented with space characters. The found matching locations may overlap with each other, this may or may not be the intended behavior.
1.2 Now Implement the search_for_symbols Function [ /16]

Solution:

```cpp
std::vector<std::vector<int>> search_for_symbols(const std::vector<std::string> &canvas,
                                                const std::vector<std::string> &symbol) {
    std::vector<std::vector<int>> answer;
    for (int row = 0; row < canvas.size() - symbol.size() + 1; row++) {
        for (int col = 0; col < canvas[0].size() - symbol[0].size() + 1; col++) {
            bool match = true;
            for (int s_r = 0; s_r < symbol.size() && match == true; s_r++) {
                for (int s_c = 0; s_c < symbol[0].size(); s_c++) {
                    if (canvas[row+s_r][col+s_c] != symbol[s_r][s_c]) {
                        match = false;
                        break;
                    }
                }
            }
            if (match) {
                std::vector<int> position;
                position.push_back(row); position.push_back(col);
                answer.push_back(position);
            }
        }
    }
    return answer;
}
```

1.3 Order Notation [ /3]

Assuming the canvas has dimensions $c_w \times c_h$, the symbol has dimensions $s_w \times s_h$, and the symbol appears $p$ times on the canvas, what is the order notation for the running time of this function?

Solution: $O(c_w \times c_h \times s_w \times s_h)$

2 Stack Sleuthing [ /15]

Carefully examine the contents of the stack after program execution shown on the right.

Your task for this problem is to complete the source code on the left so that it produces this stack.

Solution:

```cpp
int foo(int a, int b) {
    int answer;
    if (a < b)
        answer = a*b;
    else
        answer = foo(a-1,b+2);
    return answer;
}

int main() {
    int x = 8;
    int y = 3;
    int tmp = foo(x,y);
    std::cout << tmp << std::endl;
}```
3 Common C++ Programming Errors [ /12]

For each code fragment below, choose the letter that best describes the program error. Hint: Each letter will be used exactly once.

A ) Accessing data beyond the array bounds
B ) Uninitialized memory
C ) Infinite loop
D ) Compile error: type mismatch
E ) Does not contain an error
F ) Memory leak
G ) Math error (incorrect answer)
H ) Syntax error

int* apple;
int banana[5] = {1, 2, 3, 4, 5};
apple = &banana[2];
*apple = 6;

float* floating_pt_ptr = new float;
*floating_pt_ptr = 5.3;
floating_pt_ptr = NULL;

float a = 2.0;
float b = -11.0;
float c = 12.0;
float pos_root =
   -b + sqrt(b*b - 4*a*c) / 2*a;
float neg_root =
   -b - sqrt(b*b - 4*a*c) / 2*a;

double x;
for (int i = 0; i < 10; i++) {
   x += sqrt(double(i));
}

std::vector<std::string> temperature;
temperature.push_back(43.5);

unsigned int x;
for (x = 10; x >= 0; x--) {
   std::cout << x << std::endl;
}

int balance = 100;
int withdrawal;
std::cin >> withdrawal;
if (withdrawal <= balance)
   balance -= withdrawal;
else
   std::cout << "failure\n";

int grades[4] = { 1, 2, 3, 4 };
std::cout << "grades" << grades[1]
   << " " << grades[2]
   << " " << grades[3]
   << " " << grades[4]
   << std::endl;

4 Classy Running [ /34]

In this problem you will implement a simple class named RunTracker to keep track of an athlete’s running workouts. IMPORTANT: Read through all 3 pages of this problem before beginning your portion of the implementation.

First we create a number of runners, and store their recent runs by distance (in miles):

RunTracker george("George", "Smith");
RunTracker sally("Sally", "Williams");
RunTracker chris("Chris", "Jones");
george.addRun(6.7); sally.addRun(3.1); chris.addRun(2.5); george.addRun(5);
sally.addRun(13.1); chris.addRun(2); george.addRun(9.6); chris.addRun(26.2);
sally.addRun(13.1); george.addRun(11.7);

std::vector<RunTracker> runners;
runners.push_back(george); runners.push_back(sally); runners.push_back(chris);

We’d like to count how many times each runner ran a specific distance (or more). For example:

std::cout << "Runners who have completed the half marathon distance: " << std::endl;
for (unsigned int i = 0; i < runners.size(); i++) {
   int num = runners[i].numRunsAtLeast(13.1);
   if (num >= 1)
      std::cout << " " << runners[i].getName() << "(" << num << ")" << std::endl;
}
Runners who have completed the half marathon distance:
  Williams, Sally (2)
  Jones, Chris (1)

Finally, we’d like to print all runners, ordered by total distance completed. The following code:

```cpp
/* ONE LINE OF CODE OMITTED: YOU WILL FILL IN THIS LINE IN PART 1 */

for (unsigned int i = 0; i < runners.size(); i++) {
    std::cout << std::setw(18) << std::left << runners[i].getName() << " " << std::setw(10) << std::fixed << std::setprecision(1) << runners[i].totalDistance() << std::endl;
}
std::cout << std::endl;
```

Should result in the following output:

```
Williams, Sally 29.3
Jones, Chris 30.7
Smith, George 33.0
```

### 4.1 Omitted Line of Code [ /4]

What is the missing line of code above? It may depend on your answers on the next page(s).

**Solution:**

```cpp
std::sort(runners.begin(), runners.end(), byTotalDistance);
```

### 4.2 RunTracker Class Declaration [ /14]

Using the sample code on the previous pages as your guide, write the class declaration for the `RunTracker` object. That is, write the header file (`run_tracker.h`) for this class. You don’t need to worry about the `#include` lines or other pre-processor directives. Focus on getting the member variable types and member and non-member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Save the implementation of all functions for the `run_tracker.cpp` file, which is the next part.

**Solution:**

```cpp
class RunTracker {
public:
  // CONSTRUCTOR
  RunTracker(const std::string &fname, const std::string &lname);
  // ACCESSOR
  std::string getName() const;
  float totalDistance() const;
  float numRunsAtLeast(float min) const;
  // MODIFIER
  void addRun(float distance);
private:
  // REPRESENTATION
  std::string fname_;
  std::string lname_;  
  std::vector<float> runs_;  
};

// HELPER FUNCTION FOR SORTING
bool byTotalDistance(const RunTracker &a, const RunTracker &b);
```
Now implement all of the functions prototyped in the run_tracker.h file, as they would appear in the corresponding run_tracker.cpp file.

Solution:

```cpp
// CONSTRUCTOR
RunTracker::RunTracker(const std::string &fname, const std::string &lname) {
    fname_ = fname;
    lname_ = lname;
}

// ACCESSORS
std::string RunTracker::getName() const {
    return lname_ + ", " + fname_;}

float RunTracker::totalDistance() const {
    float answer = 0;
    for (int i = 0; i < runs_.size(); i++) { answer += runs_[i]; }
    return answer;
}

float RunTracker::numRunsAtLeast(float min) const {
    int answer = 0;
    for (int i = 0; i < runs_.size(); i++) {
        if (runs_[i] >= min) { answer++; }
    }
    return answer;
}

// MODIFIER
void RunTracker::addRun(float distance) {
    runs_.push_back(distance);
}

// NON-MEMBER HELPER FUNCTION FOR SORTING
bool byTotalDistance(const RunTracker &a, const RunTracker &b) {
    return a.totalDistance() < b.totalDistance();
}
```
The code below is missing its variable declaration & initialization. Fill in these lines so the code compiles and runs without syntax errors, memory errors or leaks. Show your work by diagramming the memory.

```c
int main() {
    Solution:
    char*** a = new char**;
    char*** b = a;
    *b = new char*[3];
    char** c = (*b);
    c[2] = new char[2];
    char* d[2];
    d[1] = new char;
    c[2][0] = 'R';
    (*a)[0] = NULL;
    (*b)[2][1] = 'P';
    c[1] = NULL;
    *(d[1]) = 'I';
    d[0] = NULL;
    // CREATE
    // MEMORY
    // DIAGRAM
    delete [] (*a)[2];
    delete b;
    delete [] c;
    delete d[1];
}
```

Solution:
```
char*** a:
|---|
char*** b:
|---|
char** c:
|---|
char* d:
|---|
```

Memory Diagram:
```
stack
// Memory Diagram
// Diagram
```

```c
delete [] (*a)[2];
```