COSC230 Practical 4: Stacks and Queues (Practical for Week 6)

The purpose of this practical is to use and understand the stack and queue data structures and their associated operations, and to see how they are represented in the Standard Template Library (STL).

A stack is a last-in first-out data structure, and supports the operations: `push(element)`, `pop()`, and `top_element()`. Stacks are useful when stored data needs to be retrieved in reverse order, thereby enabling simple implementations of backtracking during search, recursion, and returning from function calls. A queue is a first-in first-out data structure, and supports: `enqueue(element)`, `dequeue()`, and `first_element()`. Queues are useful when data needs to be stored for a time, and then processed in the same order in which it was stored. In this sense a queue is similar to a buffer. Stacks and queues also support the operations `clear()` and `is_empty()`. Note that stack and queue operations have different names in their STL implementations.

Question 1: **Backtracking to Exit a Mouse Maze**

Download and run the `Mouse Maze` code from Moodle. This code builds a mouse maze using a two-dimensional character array that you enter one line at a time on the command line. Note: all rows must have the same length. The passages are marked by `0`s, and the walls are marked by `1`s. The mouse starts at the letter `m`, and exits at the letter `e`. After you have entered the maze the program will pad the perimeter with `1`s. For each position, the mouse can go in one of four directions: right, left, down, or up. It always tries moves in this order. A stack is used by the program in the process of escaping the maze. To remember untried paths, the positions of the untried neighbours of the current position are stored on the stack in the reverse order of the four mouse moves: upper neighbour, lower neighbour, left neighbour, right neighbour. Only open passages are stored, not walls. To execute a mouse move the program takes the top element off the stack. Any untried neighbours are then stored for that position. This process repeats continuously until either the mouse reaches the exit, or else all

possibilities have been exhausted and the mouse finds itself trapped. Using $x - y$ coordinates starting at the bottom-left (or some other consistent labelling method) for the position of each neighbour, write down the state of the stack for each move made by the mouse in solving the following maze:

```
0001
e100
1101
m000
```

Where does the mouse backtrack? How many times does it backtrack?

Question 2: **A Number base Converter**

Write a program that makes use of a stack to convert a number in decimal notation to binary notation. Then generalize your program to handle any number base. Remember from AMTH140, that converting a number from decimal notation into binary or hexadecimal notation can be done with repeated division using the Quotient-Remainder Theorem. For example, converting 57_{10} to binary can be done with repeated division by two as:

$$\begin{aligned} 57 &= 2 \times 28 + 1, \\ 28 &= 2 \times 14 + 0, \\ 14 &= 2 \times 7 + 0, \\ 7 &= 2 \times 3 + 1, \\ 3 &= 2 \times 1 + 1, \\ 1 &= 2 \times 0 + 1, \end{aligned}$$

giving: $57_{10} = 111001_2$. The Quotient-Remainder Theorem states that given any integer n and positive integer d, there exist unique integers q and r such that

$$n = dq + r \quad \text{and} \quad 0 \leq r < d.$$

For example, in the first line given above, $n = 57$, and $d = 2$. The theorem then guarantees that the integers $q = 28$ and $r = 1$ (where $0 \leq r < 2$) exist and are unique. To find these unique integers, use $q = n \; div \; d$ and $r = n \; mod \; d$. The last piece is to read off the stored remainders of repeated division in reverse order. When generalizing to number bases greater than 9 you will need to define the additional symbols A, B, C, etc., as in the case of hexadecimal notation, for example.

Question 3: **Discovering Palindromes**

A palindrome is a string that reads the same forward or backward; that is, the sequence of letters are the same when read from left to right, or from right to left. For example, the word "radar" is a palindrome. Write a program that determines whether an input string is a palindrome. Consider data structures that store and process data in the same order, and in reverse order. To test your program, consider the palindromes "A man, a plan, a canal: Panama", and "race car".

Solutions

Question 1:

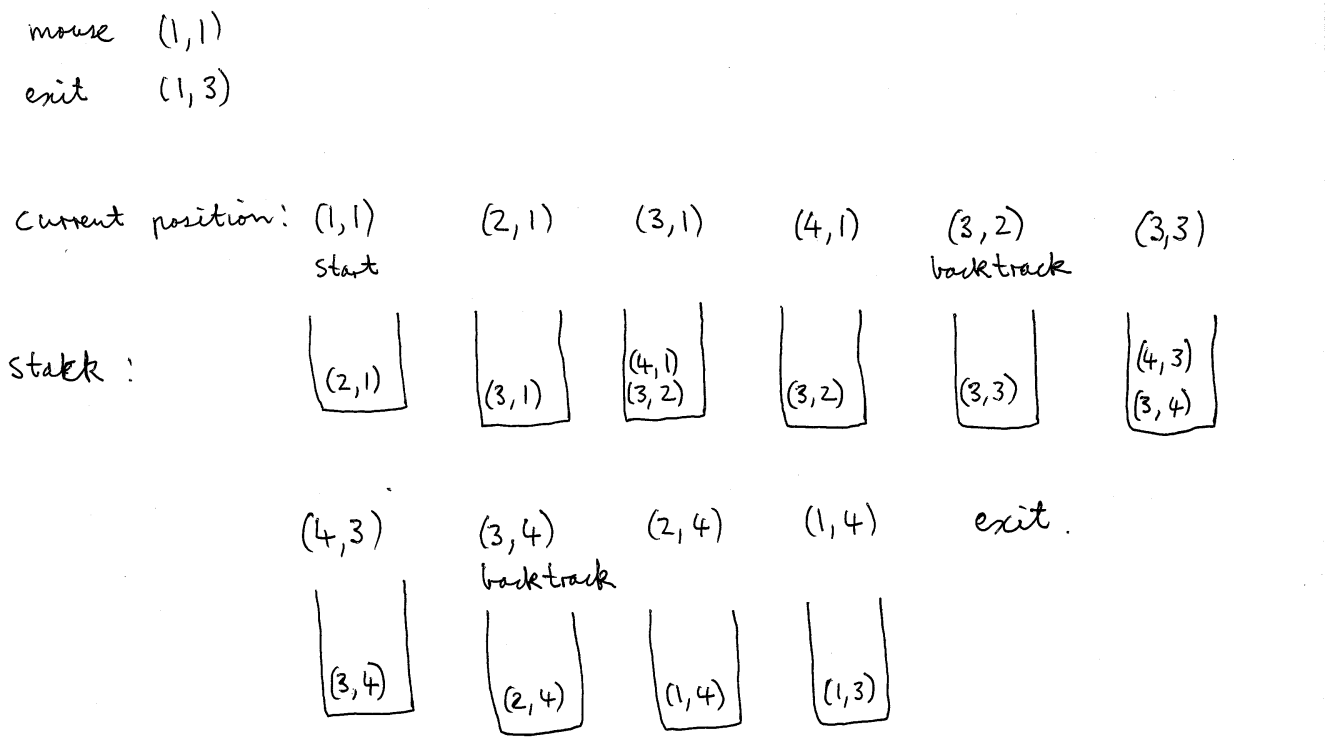

Figure 1: State of the Mouse Maze stack for each position between Start and Exit.

Question 2:

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

char find_symbol(int a)
{
    char b[] = {'A', 'B', 'C', 'D', 'E', 'F'};

    return b[a % 10];
}

void print_decimal_to_numberbase(int dec, int base)
{
    stack<int> s;

    while(dec != 0)
    {
        // Repeated division of "dec" by "base"
        s.push(dec % base); // remainder
        dec = dec / base;   // quotient
    }

    while(!s.empty())
    {
        if (s.top() <= 9)
            cout << s.top();
        else
            cout << find_symbol(s.top());
        s.pop();
    }
    cout << endl;
}

int main()
{
    print_decimal_to_numberbase(74, 16);

    return 0;
}
```

Question 3:

```
#include <iostream>
#include <stack>
#include <queue>
#include <cctype> // toupper converts all lower-case characters to upper case.
using namespace std;

int main()
{
    stack<char> s;
    queue<char> q;
    char letter;
    int mismatches = 0;

    cout << "Enter a line to see if it is a palindrome
        (terminate with Return, Ctr-D):" << endl;

    while (cin >> letter)
    {
        if (isalpha(letter)) {
            s.push(toupper(letter));
            q.push(toupper(letter));
        }
    }

    while (!s.empty() && !q.empty())
    {
        if (q.front() != s.top())
            ++mismatches;
        s.pop();
        q.pop();
    }

    if (mismatches == 0)
        cout << "That is a palindrome." << endl;
    else
        cout << "That is not a palindrome." << endl;

    return 0;
}
```