**CS1410-40 Final Exam**

1. (5 pts) Name three applications for the data structures (1) stack and (2) queue.

Stack:

Queue:

2. (5 pts) Consider the following dynamic allocation:

void\* ptr = std::malloc(1024);

std::free(ptr);

When you call free, the operating system deallocates the memory pointed to by ptr, i.e., returning the 1024 bytes of memory to the heap. How does the operating system know 1024 bytes to free by just looking at the pointer argument ptr?

3. (5 pts) What’s the difference among the following three function calls with pointer arguments?

a. void func(const int\* pointer);

b. void func(int\* const pointer);

c. void func(int\* pointer);

4. (5 pts) How many levels of pointers can we have?

5. (5 pts) Explain the difference between stack memory and heap memory.

6. (5 pts) What will the following code do?

int SIZE = 5;

double array[SIZE];

for (int i = 1; i <= SIZE; i++) {

array[i] = 0.5;

}

7. (5 pts) What is RAII? Explain why it is useful and give a specific use case of RAII.

8. (5 pts) What is a memory pool and why do we need it?

9. (2 pts) What is the difference between the following two declarations of an array?

1. int array[1000];
2. int \*array = static\_cast<int\*>malloc(1000\*sizeof(int));

10. (8 pts) We have covered the amortized analysis of vector::push\_back. If we double the capacity each time we need to enlarge the vector for a newly inserted element, we can derive O(3) amortized complexity over a sequence of N push\_back operations. Now, suppose we triple the size of the capacity (i.e., C🡪3C) instead of doubling it, what is the amortized complexity over a sequence of N push\_back operations?

11 (5 pts). Write a template function that takes two arguments of arithmetic data type (e.g., int, float, double) and return the sum of them

// TODO

// Usage of your function

std::cout << sum<int>(2, 9) << ‘\n’; // 11

std::cout << sum<float>(1.04f, 2.96f) << ‘\n’; // 4

12 (10 pts). Implement the stack data structure in a Stack class using ONLY std::queue.

class Stack {

std::queue<int> \_queue;

public:

/\*\* Push element x onto stack. \*/

void push(int x) {

}

/\*\* Removes the element on top of the stack and returns that element. \*/

int pop() {

}

/\*\* Get the top element. \*/

int top() const {

}

/\*\* Returns whether the stack is empty. \*/

bool empty() const {

}

};

13 (15 pts). You are given an array of N elements. Write a program to generate all possible combinations of r elements in the array. For example, if the array is {1, 2, 3, 4} and r is 2, then your output should be {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, and {3, 4}.

// TODO

void generate\_r\_combinations(const std::vector<int>& vec, int r);

// Usage

int main()

{

    std::vector<int> vec = {1, 2, 3, 4};

    int r = 2;

    generate\_r\_combinations (vec, r);

}

14 (5 pts). You are given a text file input.txt that contains a set of alphabets. Write a program that reads in the file and offsets each alphabet by plus one, and stores the result to another file output.txt. For example, if input.txt contains a string, abcde, then your program should generate output.txt that contains the string bcdef.

You can assume the input does not contain ‘z’.

15. (15 pts) We have discussed in the class how to write a program to check the balanced brackets in an expression using std::stack. For example:

**Input**: exp = “[]”   
**Output**: yes

**Input**: exp = “{}[]”   
**Output**: yes

**Input**: exp = “[()]{}{[()()]()}”   
**Output**: yes

**Input**: exp = “[(])”   
**Output**: no

**Input**: exp = “[()]{}{[()()]()}[]{(})”   
**Output**: no

Rewrite the program using *recursion* rather than std::stack. Discuss the pros and cons of each implementation.