CS 32 Worksheet 2

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

If you have any questions or concerns please contact apoorvapanse@ucla.edu or go to any of the LA office hours.

Concepts

Stacks, Queues, Inheritance

1.) Given a string of '(', ')', '[', and ']', write a function to check if the input string is *valid*. Validity is determined by each '(having a corresponding ')', and each '[' having a corresponding ']', with parentheses being properly nested and brackets being properly nested
Examples: “[()([])[[([][])]]]” → Valid
“((([(])))“ → Invalid

```
bool isValid(string parens) {
      // Fill in code here
}
```

2.) How many times mdo you need to add virtual (if any) to make this program output "I’m Gene", and where do you need to do so?

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

class LivingThing {
      public:
            void intro() { cout << "I’m a living thing" << endl; }
};

class Person : public LivingThing {
      public:
            void intro() { cout << "I’m a person" << endl; }
};

class UniversityAdministator : public Person {
      public:
```

```
            void intro() {
                  cout << "I'm a university administrator" << endl;
            }
};

class Chancellor : public UniversityAdministrator {
      public:
            void intro() { cout << "I'm Gene" << endl; }
};

int main() {
      LivingThing* thing = new Chancellor();
      thing->intro();
}
```

3.) Give an algorithm for reversing a queue Q. Only following standard operations are allowed on queue:
   a.) Q.push(x) : Add an item x to the back of the queue.
   b.) Q.pop() : Remove an item from the front of the queue.
   c.) Q.top() : Return the item at the front of the queue
   d.) Q.empty() : Checks if the queue is empty or not.

   You may use an additional data structure if you wish.

   Example:
   Input : Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
   Output :Q = [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]

```
void reverseQueue(queue<int>& Q){
      // your code goes here
}
```

4.) Implement a Stack class using only queues as data structures. This class should implement the *empty*, *size*, *top*, *push*, and *pop* member functions, as specified by the standard library's implementation of stack. (The implementation will not be very efficient.)

5.) Implement a Queue class using only stacks as data structures. This class should implement the *empty*, *size*, *front*, *back*, *push*, and *pop* member functions, as specified by the standard library's implementation of queue. (The implementation will not be very efficient.)

6.) Write a function *findNextInts* that takes in two integer arrays *sequence* and *results*, along with the size of both of them, which is *n*. This function assumes that *sequence* already contains a sequence of positive integers. For each

position *i* (from 0 to *n*-1) of *sequence*, this function should find the smallest *j* such that *j* > *i* and *sequence[j]* > *sequence[i]*, and put *sequence[j]* in *results[i]*; if there is no such *j*, put -1 in *sequence[i]*. Try to do this without nested for loops both iterating over the array! (Hint: `#include <stack>`).

```
void findNextInts(const int sequence[], int results[], int n);
```

Example:

```
int seq[] = {2, 6, 3, 1, 9, 4, 7 }; // Only positive integers!
int res[7];
findNextInts(seq, res, 7);
for (int i = 0; i < 7; i++) { // Should print: 6 9 9 9 -1 7 -1
  cout << res[i] << " ";
}
cout << endl;
```

Notice that the last value in *results* will always be set to -1 since there are no integers in *sequence* after the last one!

7.) Given the declarations for following classes and their constructors, complete the implementations for the constructors such that the code would compile. The implementations should assign the constructor arguments to the member variables. HINT: You will need to use initializer lists!

```
class Animal {
public:
      Animal(string name);
private:
      string m_name;
};

class Cat : public Animal {
public:
      Cat(string name, int amountOfYarn);
private:
      int m_amountOfYarn;
};

class Himalayan : public Cat {
public:
      Himalayan(string name, int amountOfYarn);
};
```

```
class Siamese: public Cat {
public:
      Siamese(string name, int amountOfYarn, string toyName);
private:
      string m_toyName;
};
```

8.) Would something like the following work in C++? Why or why not?

```
class B;
class A : public B {// Code for A};
class B : public A {// Code for B};
```

9.) What is the output of the following code?

```
class Pet {
public:
      Pet() { cout << "Pet" << endl; }
      ~Pet() { cout << "~Pet" << endl; }
};

  // This is an unusual class that derives from Pet but also
  // contains a Pet as a data member.
class Dog : public Pet {
public:
      Dog() { cout << "Woof" << endl; }
      ~Dog() { cout << "Dog ran away!" << endl; }
private:
      Pet buddy;
};

int main() {
      Pet* milo = new Dog;
      delete milo;
}
```

10.) Now suppose the class declaration for Pet is as shown below. What is the output of the code in 9) with these new changes?

```
class Pet {
public:
      Pet() { cout << "Pet" << endl; }
      virtual ~Pet() { cout << "~Pet" << endl; }
```

```
};
```

11.) The following code has several errors. Rewrite the code so that it would successfully compile. Try to catch the errors without the use of a compiler.

```
class LivingThing {
private:
      int age;
};

class Person : public LivingThing {
public:
      Person(int a) { age = a; }
      void birthday() {
            age++;
      }
};
```

12.) Evaluate the following postfix expression, show your work: 9 5 * 8 - 6 7 * 5 3 - / *

13.) Look through the following code and write the output.

```
#include <iostream>
#include <string>

using namespace std;

class A {
public:
      A() : m_val(0) {
            cout << "What a wonderful world! " << m_val << endl;
      }
      virtual ~A() { cout << "Guess this is goodbye " << endl; }
      virtual void saySomething() = 0;
      virtual int giveMeSomething() = 0;
private:
      int m_val;
};

class B : public A {
public:
      B() : m_str("me"), m_val(1) {
            cout << m_str << " has just been birthed."  << endl;
```

```
        }
        B(string str, int val) : m_str(str), m_val(val) {
                cout << "More complex birth " << m_str << endl;
        }
        ~B() {
                cout << "Why do I have to leave this world!" << endl;
        }
        virtual void saySomething() {
                cout << "Coming in from " << m_str << " with "
                     << giveMeSomething() << endl;
        }
        virtual int giveMeSomething() { return m_val*5; }
private:
        int m_val;
        string m_str;
};

class C {
public:
        C() : m_val(2) {
                m_b = new B("C", m_val);
                cout << "Hello World!!" << endl;
        }
        C(B b, int val) :  m_val(val) {
                m_b = new B(b);
                cout << m_b->giveMeSomething() << endl;
        }
        ~C() {
                m_b->saySomething();
                delete m_b;
                cout << "Goodbye world!" << endl;
        }
private:
        B* m_b;
        int m_val;
};

int main() {
        B* b_arr = new B[5];
        for(int i = 0; i < 5; i++) {
                b_arr[i].saySomething();
        }
        B b("B", 5);
```

```
    A* a = &b;
    cout << a->giveMeSomething() << endl;
    C c;
    C c2(b, b.giveMeSomething());
    delete [] b_arr;
}
```