CS 32 Worksheet 4

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.

Solutions are written in red. The solutions for **programming** problems are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

If you have any questions or concerns please email arabellekezia@ucla.edu or brendon1097@gmail.com, or go to any of the LA office hours.

Concepts

Stacks, Queues

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 (JKC)
   Examples: "[()([])[[([])]]]" → Valid

   "(((([()))))" → Invalid

```
// The idea here is that our stack maintains the sequence of
// opening parentheses and brackets, and removes an opening
// symbol upon seeing the matching closing one. Note that if we
// have a closing symbol, but the stack is empty or the top of
// the stack is not the the matching opening symbol, then we've
// encountered an invalid sequence of parens and brackets.
bool isValid(string symbols) {
        stack<char> openers;
            for (int k = 0; k != symbols.size(); k++) {
                char c = symbols[k];
                switch (c) {
                    case '(':
                    case '[':
                        openers.push(c);
                        break;
                    case ')':
```

```
                    if (openers.empty() || openers.top()!='(')
                        return false;
                    openers.pop();
                    break;
                case ']':
                    if (openers.empty() || openers.top()!='[')
                        return false;
                    openers.pop();
                    break;
            }
        }
        return openers.empty() ;
}
```

2) 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.front() : 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){
        // use an auxiliary stack
        stack<int> S;
        while (!Q.empty()) {
            S.push(Q.front());
            Q.pop();
        }
        while (!S.empty()) {
            Q.push(S.top());
            S.pop();
        }
}
```

3) 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.)

```cpp
class Stack {
  //This implementation of Stack accepts only int. See if you can
  //make an implementation with templates!
public:
  bool empty() const;
  size_t size() const;
  int& top();
  const int& top() const;
  void push(const int& value);
  void pop();
private:
  queue<int> storage;
};

bool Stack::empty() const {
  return storage.empty();
}

size_t Stack::size() const {
  return storage.size();
}

int& Stack::top() {
  return storage.back();
}

const int& Stack::top() const {
  return storage.back();
}

void Stack::push(const int& value) {
  storage.push(value);
}

void Stack::pop() {
  int limit = storage.size() - 1;
  for (int n = 0; n < limit; n++) {
    storage.push(storage.front());
    storage.pop();
  }
  storage.pop();
}
```

4) 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.)

```cpp
class Queue {
  //This implementation of Queue accepts only int. See if you can
  //make an implementation with templates!
public:
  bool empty() const;
  size_t size() const;
  int& front();
  const int& front() const;
  int& back();
  const int& back() const;
  void push(const int& value);
  void pop();
private:
  //move items from pushStorage to popStorage while leaving back
  //item within pushStorage
  void moveItems();

  //storage for pushing items with one exception: always includes
  //back item if available
  stack<int> pushStorage;
  //storage for popping items: always includes front item
  stack<int> popStorage;
};

bool Queue::empty() const {
  return pushStorage.empty() && popStorage.empty();
}

size_t Queue::size() const {
  return pushStorage.size() + popStorage.size();
}

int& Queue::front() {
  return popStorage.top();
}

const int& Queue::front() const {
  return popStorage.top();
}

int& Queue::back() {
  if (size() == 1)
```

```cpp
      return popStorage.top();
    return pushStorage.top();
}

const int& Queue::back() const {
  if (size() == 1)
      return popStorage.top();
    return pushStorage.top();
}

void Queue::push(const int& value) {
  if (size() > 0)
    pushStorage.push(value);
  else
    popStorage.push(value);
}

void Queue::pop() {
  if (popStorage.size() > 0) {
    popStorage.pop();
    if (popStorage.size() == 0 && pushStorage.size() > 0)
      moveItems();
  }
  else {
    moveItems();
    popStorage.pop();
  }
}

void Queue::moveItems() {
  int& temp = pushStorage.top();
  bool backExists = false;
  if (pushStorage.size() > 1) {
    pushStorage.pop();
    backExists = true;
  }

  while (pushStorage.size() > 0) {
    popStorage.push(pushStorage.top());
    pushStorage.pop();
  }

  if (backExists)
    pushStorage.push(temp);
```

```
}
```

5) 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!

```cpp
void findNextInts(const int sequence[], int results[], int n) {
  if (n <= 0)
    return;

  stack<int> s;

    // push the first index to stack
  s.push(0);

    // iterate for rest of the elements
  for (int i = 1; i < n; i++) {
    int current = sequence[i];

      // Fill in results for preceding unfilled items
      // that are less than current.
    while (!s.empty() && current > sequence[s.top()]) {
      results[s.top()] = current;
      s.pop();
    }

    s.push(i);
  }

    // Remaining items don't have a later greater value
  while (!s.empty()) {
    results[s.top()] = -1;
    s.pop();
  }
}
```

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

```
45 8 - 42 2 / *
37 21 *
777
```

Inheritance and Polymorphism

1.) What changes do you have to make to the following program to have it
output "I'm Gene"?

HINT: You will need to use the virtual keyword!

```cpp
#include <iostream>
```

```
using namespace std;

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

class Person : public LivingThing {
  public:
        // repeating the word virtual is not required here (but
        // recommended as a reminder to a human reader)
      void intro() { cout << "I'm a person" << endl; }
};

class UniversityAdministrator : public Person {
  public:
        // repeating the word virtual is not required here (but
        // recommended as a reminder to a human reader)
      void intro() {
            cout << "I'm a university administrator" << endl;
      }
};

class Chancellor : public UniversityAdministrator {
  public:
        // repeating the word virtual is not required here (but
        // recommended as a reminder to a human reader)
      void intro() { cout << "I'm Gene" << endl; }
};

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

Difficulty: Easy

2.) Given the following class declarations, complete the implementation of each constructor so that the program compiles. Your implementations should correctly assign constructor arguments to class member variables.

HINT: You will need to use initializer lists!

```cpp
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;
};

Animal::Animal(string name)
    : m_name(name) {}

Cat::Cat(string name, int amountOfYarn)
    : Animal(name), m_amountOfYarn(amountOfYarn) {}

Himalayan::Himalayan(string name, int amountOfYarn)
    : Cat(name, amountOfYarn) {}

Siamese::Siamese(string name, int amountOfYarn, string toyName)
    : Cat(name, amountOfYarn), m_toyName(toyName) {}
```

Difficulty: Easy

3.) Would the following work in C++? Why or why not?

```cpp
class B;
```

```
class A : public B { … code for A … };
class B : public A { … code for B … };
```

Conceptually, this code is saying "A is a proper subset of B, and B is a proper subset of A", which is nonsense.

Practically, every object of a derived class contains an instance of the base class. If the code above were legal, a B object would contain an A object that contains a B object that contains an A object, ad infinitum.

Difficulty: Medium

4.) What is the output of the following program?

```
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;
}
```

Pet
Pet
Woof
~Pet

Undefined behavior after this, because Pet's destructor is not declared virtual.

Difficulty: Medium

5.) Suppose the class declaration for Pet was changed as shown below. What is the new output of the code in problem 4) with these new changes?

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

Pet
Pet
Woof
Dog ran away!
~Pet
~Pet

Difficulty: Medium

6.) The following code has several errors. Rewrite the code so that it can successfully compile. Try to catch the errors without using a compiler!

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

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

Difficulty: Easy

7.) Examine the following code and determine its output. (JKC)

```cpp
#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;
    }
```

```cpp
        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;
}
```

Difficulty: Hard

What a wonderful world! 0
me has just been birthed.
What a wonderful world! 0
me has just been birthed.
What a wonderful world! 0
me has just been birthed.
What a wonderful world! 0
me has just been birthed.
What a wonderful world! 0
me has just been birthed.
Coming in from me with 5
Coming in from me with 5
Coming in from me with 5
Coming in from me with 5
Coming in from me with 5
What a wonderful world! 0

More complex birth B
25
What a wonderful world! 0
More complex birth C
Hello World!!
25
Why do I have to leave this world!
Guess this is goodbye
Why do I have to leave this world!
Guess this is goodbye
Why do I have to leave this world!
Guess this is goodbye
Why do I have to leave this world!
Guess this is goodbye
Why do I have to leave this world!
Guess this is goodbye
Why do I have to leave this world!
Guess this is goodbye
Coming in from B with 25
Why do I have to leave this world!
Guess this is goodbye
Goodbye world!
Coming in from C with 10
Why do I have to leave this world!
Guess this is goodbye
Goodbye world!
Why do I have to leave this world!
Guess this is goodbye