CS 32 Worksheet Week 5

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. Although exams are online this quarter, it is still 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 your LA or go to any of the LA office hours.

Concepts: Inheritance, Polymorphism, Recursion

Inheritance, Polymorphism

1. What does the following code output, and what changes do you have to make to it to have it output "`I'm Gene`"?

   HINT: You will need to use the virtual keyword!

```
#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 UniversityAdministrator : 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();
}
```

Time: 5 mins

2. What is the output of the following code?

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

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;
` }
```

Time: 5 mins

3. Suppose the class declaration for Pet was changed as shown below. What is the output of the code in problem 2) with these new changes?

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

Time: 5 mins

4. Would 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 … };
```

Time: 5 mins

Recursion

1) Given a singly-linked list class LL with a member variable *head* that points to the first *Node* struct in the list, write a function to recursively delete the whole list, void LL::deleteList(). Assume each Node object has a next pointer.

```
struct Node {
      int data;
      Node* next;
};

class LL {
  public: // other functions such as insert not shown
      void deleteList(); // implement this function
  private: // additional helper allowed
      Node* m_head;
};

Idea: delete until cant
Nodde == nullptr
Var to save the location of n+1
Delete node n
Call func
```

Time: 10 mins

2) Implement the function `isPalindrome` recursively. The function should return whether the given string is a palindrome. A palindrome is described as a word, phrase or sequence of characters that reads the same forward and backwards.

```
bool isPalindrome(string foo);

isPalindrome("kayak"); // true
isPalindrome("stanley yelnats"); // true
isPalindrome("LAs rock"); // false (but the sentiment is true
:))
```

Time: 15 mins

3) Write a recursive function `isPrime` to determine whether a given positive integer input is a prime number or not. You may add an auxiliary helper function if necessary.

Example:

isPrime(11) → true

isPrime(4) → false

```
bool isPrime(int num) {
      // Fill in code here
}
```

Time: 15 mins

4) Implement the following recursive function:

```
string longestCommonSubsequence(string s1, string s2);
```

The function should return the longest common subsequence of characters between the two strings `s1` and `s2`. Basically, it should return a maximum length string of characters that are common to both strings and are in the same order in both strings.

Example:

```
string res = longestCommonSubsequence("smallberg",
"nachenberg");

//res should contain "aberg" as seen in the green chars

res = longestCommonSubsequence("los angeles",
"computers");

//res should contain the string "oes"
```

Time: 20 mins

Additional Practice Problems

Inheritance, Polymorphism

1) 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!

```
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;
};
```

2) 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++;
      }
};
```

3) Examine the following code and determine its 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(const 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[3];
        for(int i = 0; i < 3; 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;
    }
```

Recursion

1) What does the following code output and what does the function LA_power do?

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

      int LA_power(int a, int b)
      {
         if (b == 0)
           return 0;
         if (b % 2 == 0)
           return LA_power(a+a, b/2);

         return LA_power(a+a, b/2) + a;
      }

      int main()
      {
        cout << LA_power(3, 4) << endl;
      }
```

2) Implement the recursive function `merge` that merges two sorted linked lists `l1` and `l2` into a single sorted linked list. The lists are singly linked; the last node in a list has a null next pointer. The function should return the head of the merged linked list. No new Nodes should be allocated while merging.

   Example:

   `l1:  1 -> 4 -> 6 -> 8`

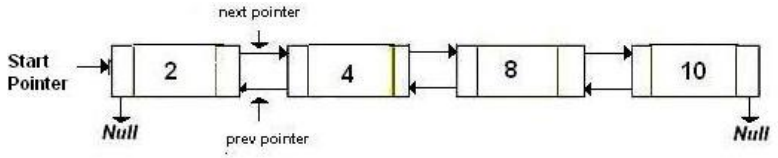
   `l2:  3 -> 9 -> 10`

   After merge: `1 -> 3 -> 4 -> 6 -> 8 -> 9 -> 10`

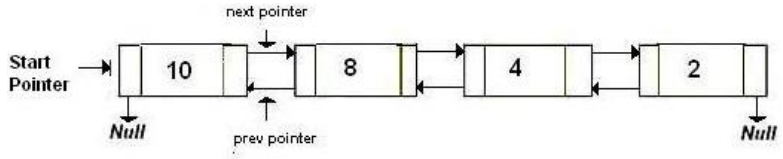
```
      // Node definition for singly linked list
struct Node {
      int val;
      Node* next;
};

Node* merge(Node* l1, Node* l2);
```

3) Implement `reverse`, a recursive function to reverse a doubly linked list. It returns a pointer to the new head of the list. The integer value in each node must not be changed (but of course the pointers can be).

Example:
Original:



After:




```
// Node definition for doubly linked list
struct Node {
      int val;
      Node* next;
      Node* prev;
};

Node* reverse(Node* head);
```