MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

Week 5 Tutorial

Department of Computing and Software

Instructor:

Omid Isfahanialamdari

February 15, 2022

Question 1.1 Multiple Choice

Each question has just one correct answer. Therefore multiple selections will not get a mark.

1. Which of the following statements are **true**:
   (S1) when class B inherits from class A, it absorbs A's capabilities, then it can customize or modify them.
   (S2) a base class object is also an object of that class's derived classes.

   (a) both S1 and S2 (b) only S1 (c) only S2 (d) none of them

Question 1.2 Multiple Choice

2. Which of the following statements are **true**:
   (S1) when a derived class inherits a public member of a base class, it can make the public member private.
   (S2) when an object of a derived class is destroyed, first the destructor of the derived-class is called, then the destructor of the base-class is called.

(a) both S1 and S2 (b) only S1 (c) only S2 (d) none of them

Question 1.3 Multiple Choice

3. Which of the following statements is **false**:
   (a) with polymorphism, same function call can cause various actions to occur.
   (b) polymorphism can be realized via virtual functions and dynamic binding in the inheritance relationship between classes.
   (c) in function overloading, functions with the same name must only differ in the number of parameters.
   (d) the nodes of singly linked list may not be stored in contiguous memory locations.

Question 2

The following code fragment does not compile. Explain what is the issue and also explain how to correct it:
Hint: there is only one issue.

```
class A {
    public:
        A(int inp) {
            a = inp;
        }
        int a;
};

class B : public A {
    public:
        B(int a1, int b1) {
            A tmp = A(a1);
            b = b1;
        }
        int b;
};
```

Question 2

```
class A {
    public:
        A(int inp) {
            a = inp;
        }
        int a;
};

class B : public A {
    public:
        B(int a1, int b1) {
            A tmp = A(a1);
            b = b1;
        }
        int b;
};
```

```
class B : public A {
    public:
        B(int a1, int b1): A(a1) {
            //A tmp = A(a1);
            b = b1;
        }
        int b;
};
```

- constructor for 'B' must explicitly initialize the base class 'A' which does not have a default constructor

Question 3

Suppose array A that is given as input to Insertion Sort algorithm has two entries that are equal to each other. An example is $A = [5, \mathbf{12}, 14, \mathbf{12}, 1]$. After running the insertion sort algorithm, will the equal entries maintain their same relative order or their relative order will change? For the example above, will the algorithm swap the two 12 entries? **Explain your answer.**

Note: An implementation of Insertion Sort algorithm is given below for your reference.

```
void insertionSort(int *arr, int n) {
    for (int i = 1; i < n; i++){
        int cur = arr[i];
        int j = i - 1;
        while ((j >= 0) && (arr[j] > cur)){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = cur;
    }
}
```

Question 3

```
void insertionSort(int *arr, int n) {
    for (int i = 1; i < n; i++){
        int cur = arr[i];
        int j = i - 1;
        while ((j >= 0) && (arr[j] > cur)){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = cur;
    }
}
```

- arr = [5, 12, 14, 12, 1]

- No swap! Obviously when the algorithm reaches the second 12, it will not enter into the inner loop, so there will be no swap.

Question 4

In the following, a C++ implementation of a **Tuple** class is given. Three constructors are defined for this class. There are two functions **f1** and **f2** defined outside of the class. Notice that each constructor of the class outputs a message in addition to what it is supposed to do—only these constructors output messages. Please pay attention to constructors and when they are called.

Question 4

```
#include <iostream>
using namespace std;
class Tuple {
    public:
        Tuple(): t1(0), t2(0) {
            cout << "Constructor no 1" << endl;
        }
        Tuple(int a, int b=0) :t1(a), t2(b){
              cout << "Constructor no 2" << endl;
        }
        Tuple(const Tuple& t){
            t1 = t.t1;
            t2 = t.t2;
            cout << "Constructor no 3" <<  endl;
        }

        int t1, t2;  // member variables
};  // end Tuple class declaration

void f1(Tuple &t){
    t.t1 = 0;
}
Tuple f2(){
    return Tuple(2);
}
int main( ) {
    Tuple tuples1[3];  // Question 4.1
    Tuple t(1,2);      // Question 4.2
    f1(t);             // Question 4.3
    f2();              // Question 4.4
    Tuple *t_ptr = &t; // Question 4.5
    Tuple a(t);        // Question 4.6
    return EXIT_SUCCESS;
}
```

Constructor no 1
Constructor no 1
Constructor no 1

Constructor no 2

no output

Constructor no 2

no output

Constructor no 3

Question 5

In the following, a C++ declaration of a Singly Linked List and its Node classes is given. The `SLinkedList` class has a `size` member variable. Constructor of `SLinkedList` initializes `size` to zero, `addFront` increments it and `removeFront` decrements it, so that it stores the number of elements in the list correctly. For example, the size of the following `SLinkedList` is 4.

An exception class called `TooFewNodesException` is also implemented that will be used in part 2 of this question.

Question 5

```
typedef int Elem;
class Node {
    private:
        Elem elem;
        Node* next;
        friend class SLinkedList;
};

class SLinkedList {
    public:
        SLinkedList();
        ~SLinkedList();
        bool empty() const;
        void addFront(const Elem& e);
        void removeFront();
        bool contains(const Elem& e);
        const Elem& secondLast() const;
    private:
        Node* head;
        int size;
};
```

Question 5

```
typedef int Elem;
class Node {
    private:
        Elem elem;
        Node* next;
        friend class SLinkedList;
};

class SLinkedList {
    public:
        SLinkedList();
        ~SLinkedList();
        bool empty() const;
        void addFront(const Elem& e);
        void removeFront();
        bool contains(const Elem& e);
        const Elem& secondLast() const;
    private:
        Node* head;
        int size;
};
```

```
class TooFewNodesException {
    private:
        string errMsg;
    public:
        TooFewNodesException(const string& msg) {
            errMsg = msg;
        }
        string getMessage() const {
            return errMsg;
        }
};
```

Question 5

```
typedef int Elem;
class Node {
    private:
        Elem elem;
        Node* next;
        friend class SLinkedList;
};

class SLinkedList {
    public:
        SLinkedList();
        ~SLinkedList();
        bool empty() const;
        void addFront(const Elem& e);
        void removeFront();
        bool contains(const Elem& e);
        const Elem& secondLast() const;
    private:
        Node* head;
        int size;
};
```

Part 1

Implement the **contains** function as declared in the `SLinkedList` class. The function returns **true** if the parameter e is in the list and returns **false** otherwise. Please implement the function in the blank space.

```
bool SLinkedList::contains(const Elem& e) {
    Node* cur = head;
    while(cur != NULL) {
        if (cur->elem == e){
            return true;
        }
        cur = cur->next;
    }
    return false;
}
```

Question 5

```
typedef int Elem;
class Node {
    private:
        Elem elem;
        Node* next;
        friend class SLinkedList;
};

class SLinkedList {
    public:
        SLinkedList();
        ~SLinkedList();
        bool empty() const;
        void addFront(const Elem& e);
        void removeFront();
        bool contains(const Elem& e);
        const Elem& secondLast() const;
    private:
        Node* head;
        int size;
};
```

Part 2

Implement the **secondLast** function as declared in the SLinkedList class. The function returns the second last element of the list. For the example above, it will return 35. Note that, the second last element is **undefined** for lists with too few nodes. Your implementation of the **secondLast** function must throw a **TooFewNodesException** exception with a proper message in that situation.

Please implement the function in the blank space.

```
const Elem& SLinkedList::secondLast() const{
    if (size < 2 ){
        throw TooFewNodesException("Too few nodes are in the list!");
    }
    Node* cur = head;
    while(cur->next->next != NULL) {
        cur = cur->next;
    }
    return cur->elem;
}
```