Advanced Programming

Due date: 2018. 09. 02 at 23:59:59

1. (Recursive) In Calculus, Riemann integral is one of the numerical methods to calculate the integral $\int_a^b f(x)dx$ of a function $f(x)$ over an interval $[a, b]$. The Riemann integral is the limit of Riemann sum S, which is an approximation of the integral calculated by splitting $[a, b]$ into arbitrary subintervals and calculate the following equation.

$$\mathrm{S} = \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i)$$

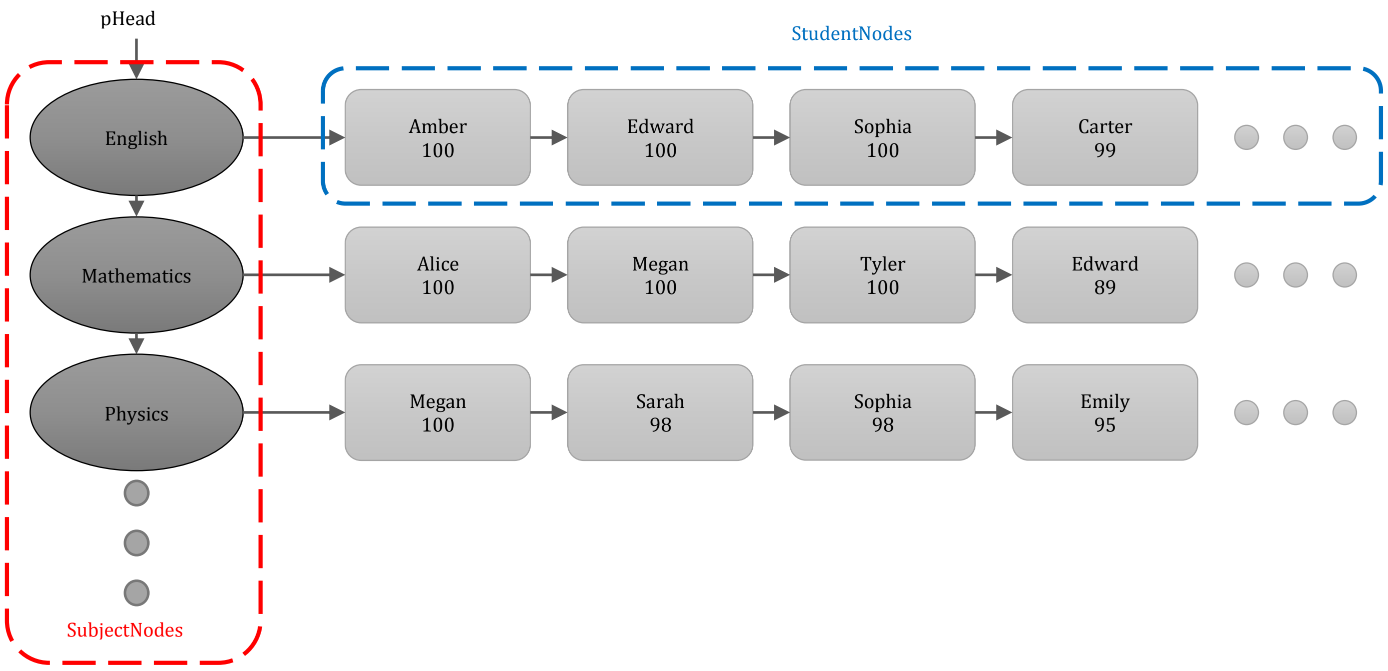
n is the number of subintervals, and $[x_i, x_{i+1}]$ is the i^{th} subinterval. Read the following pseudo code and implement Riemann sum to find a numerical approximation of $f(x) = x^2$ using a recursive function. The condition for recursion termination is when the absolute difference between Riemann sum values of two successive calls is less than the given threshold e. Note that the initial call to *RiemannSum* should include pre-calculated Riemann sum value with a single subinterval within the initial interval. Write a program to test your subroutine with different intervals of integration $[a, b]$ and the threshold e.

```
START $RiemannSum(S, a, b, e)$
        Let $a_0 = a,\ b_0 = a_1 = \frac{(a+b)}{2}$ and $b_1 = b$.
        Calculate $S_0 = (b_0 - a_0)f\left(\frac{a_0+b_0}{2}\right)$ and $S_1 = (b_1 - a_1)f(\frac{a_1+b_1}{2})$.
        if(| $S - (S_0 + S_1)$ | $\geq e$)
                return $RiemannSum(S_0, a_0, b_0, e) + RiemannSum(S_1, a_1, b_1, e)$;
        else
                return $S$;
END
```

2. (2D-linked list) Implement a score management system with a 2D-linked list. Use three classes in your implementation; *ScoreList*, *SubjectNode* and *StudentNode*. *ScoreList* reads and analyzes a text file named “score.txt” and makes the 2D-linked list out of the content of the file. The other two classes are node class for the 2D-linked list such that *SubjectNode* must be sorted in alphabetical order and *StudentNode* must be sorted by scores in descending order. The arrangement of nodes is shown in the following figure. If multiple *StudentNodes* have an identical score, sort them by their names in alphabetical order (Assume that students with the same name do not exists). Write a program that creates the 2D-linked list with *ScoreList* and demonstrates your code properly.

score.txt

	<English>	<Mathematics>	<Physics>
Edward	100	89	60
Daniel	80	80	75
Thomas	77	45	88
Carter	99	87	65
Tyler	67	100	89
Phiilp	86	87	63
Sophia	100	40	98
Amber	100	67	87
Sarah	55	75	98
Megan	92	100	100
Alice	55	100	77
Miranda	78	44	0
Victor	29	89	94
Ralph	60	69	68
Oliin	92	57	52
Harry	0	0	30
Emma	98	30	65
Clarla	65	88	45
Emily	89	50	95
Jessica	32	24	48



3. (2D-linked list) Add three member functions AvgOfSub, AvgOfStu, and RankOfStu to your class *ScoreList* in the problem 2. The functions traverse the 2D-linked list and calculate followings. Write the three functions with test code which can verify the functions. (Do not change the prototypes of the functions)

   a. Average score of a subject named *pSubjectName* over all students

```
double ScoreList::AvgOfSub(char* pSubjectName)
{
    ...
}
```

   b. Average score of a student named *pStudentName* over all subjects

```
double ScoreList::AvgOfStu(char* pStudentName)
{
    ...
}
```

   c. Rank of a student named *pStudentName*

```
int ScoreList::RankOfStu(char* pStudentName)
{
    ...
}
```

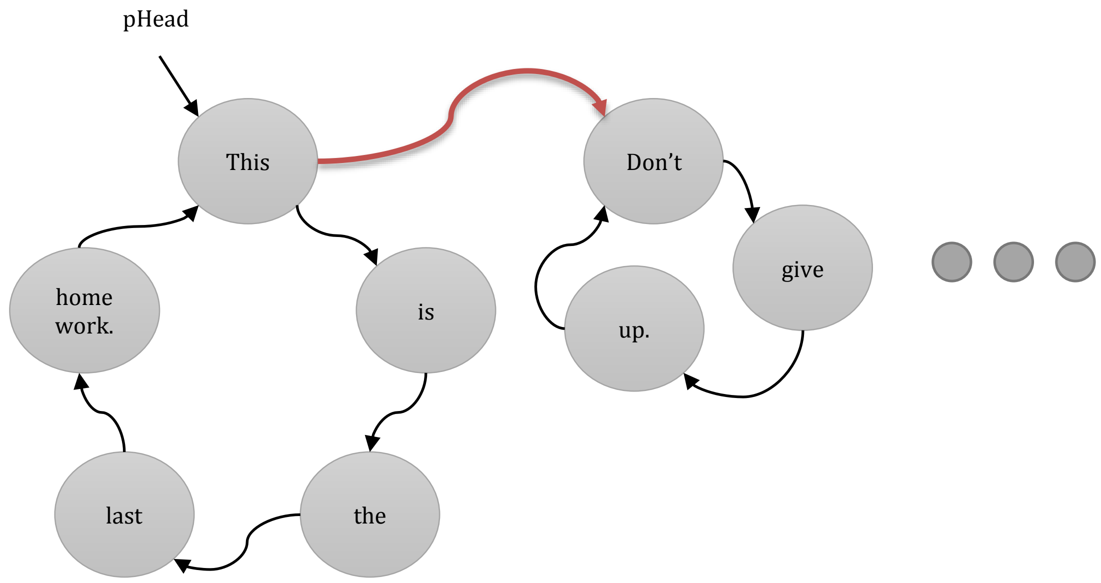
4. (2D-Circular linked list) Implement *my2D_CircularLinkedList* class to store the content from the text file "sample.txt". The class connects circular linked lists where each circular linked list represents a sentence in the file. A circular linked list is composed of *Node*s each of which stores a word in the corresponding sentence. The order of nodes in the circular linked list is the order of words in the sentence, and likewise, the order of nodes in the linked list is the order of sentences in the text file. Note that *Node* class should include both link pointer for the circular linked list (*pNextCircularList*) and the linked list (*pNextNode*). Assume that no single word in the text file is more than 32 characters long.

```
class Node
{
    private:
        Node* pNextCircularList;
        Node* pNextNode;
        char    word[32];

    public:
        Node();
        ~Node();
        Node* getNextCircularList();
        Node* getNextNode();
        char* getWord();
        void setNextCircularList(Node* pNext);
        void setNextNode(Node* pNext);
        void setWord(char* word);
}
```

sample.txt

```
This is the last homework.
Don't give up.
                    .
                    .
                    .
```



5. (Constructor & Operator overloading) Implement *Time* class that has three integer member variables, *m_hour*, *m_min*, *m_sec*, to represent time information. Write the constructor overloadings (see example (a) shown below) and operator overloading (+, -, +=, -=, ++, --, =, ==, <, <=, >, >=) for *Time* class and then demonstrate your code properly.

a) Examples of constructor overloading (Implement at least all the followings).

Time A(10, 30, 59);
A.print();
 	
10h 30m 59s
	Time B(3665);
	B.print();
	
	1h 1m 5s
		Time A(0, 59, 59)
		Time C = A;
		C.print();
		
		0h 59m 59s

b) Examples of overload operators.

Time A(18, 30, 59);
Time B(7, 29, 2);
Time C;
C = A + B ;
A.print();
B.print();
C.print();
 	
18h 30m 59s
7h 29m 2s
26h 0m 1s
	Time A(10, 30, 58)
	Time B;
	B = A++;
	A.print();
	B.print();
	B = ++A;
	A.print();
	B.print();
	
	10h 30m 59s
	10h 30m 58s
	10h 31m 0s
	10h 31m 0s
		Time A(5, 23, 10);
		Time B, C;
		C = B = A;
		B.print();
		C.print();
		
		5h 23m 10s
		5h 23m 10s

6. (Class template) Implement a linked-list with a class template. The linked-list should use two classes, *myList* and *Node*. *Node* class is composed of a pointer which points to a next *Node* and a template data which covers any data types. *myList* class is a 1-D linked list, which composed of following operations.

   a. bool Insert(Node<T>* pNode);

   b. bool Delete(T data);

   c. Node<T>* Search(T data);

7. (Class template) In the assignment #3-5, the class *myMatrix* has an integer type double-pointer *m_matrix* for storing a square matrix as a 2-D array. Modify the class *myMatrix* with template so that the 2-D array pointer (*m_matrix*) can cover any data types.

```
template <class T>
class myMatrix
{
  private:
    T** m_matrix;
    int    m_size;

  public:
    myMatrix( );
    ~myMatrix( );

    void createMatrix( int n );
    void printMatrix( );
    bool Add( myMatrix& m );
    bool Sub( myMatrix& m );
    bool Mul( myMatrix& m );
}
```

8. (Class inheritance & Function overriding) Implement two classes, *Professor* and *Student* inheriting *Person* class. *Person* class is inherited by *Professor* and *Student* class. *Professor* class has extra member variables of a professor number and a major, and *Student* class has additional member variables of a student number, a major and a school year. Override *Say* function for inherited classes to display all member variables in the class object. Write a program that creates *Professor* and *Student* class objects and test your code properly. (Do not change the prototype of *Say* function)

```
class Person
{
    protected:
        int age;
        char name[32];

    public:
        Person();
        ~Person();
        virtual void Say() = 0;
}
```

9. (Multiple inheritance & Function overriding) *Assistant* is a class describing information of teaching assistants, which includes some of member variables of both *Professor* and *Student* classes which are defined in the previous problem. Implement *Assistant* class by inheriting *Professor* and *Student* classes and override *Say* member function for *Assistant* class. Write a program to test your code.