EECS2040 Data Structure Hw #3 (Chapter 4 Linked List)
due date 4/25/2021
by 107061123 孫元駿

Format: Use a text editor to type your answers to the homework problem. You need to submit your HW in an HTML file or a DOC file named as **Hw3-SNo.doc** or **Hw3-SNo.html**, where SNo is your student number. Send the **Hw3-SNo.doc or Hw3-SNo.html** file to me (tljong@mx.nthu.edu.tw) via e-mail. Inside the file, you need to put the **header and your student number, name (e.g., EE2410 Data Structure Hw #3 (Chapter 4 of textbook) due date 4/25/2021 by SNo, name)** first, and then the **problem** itself followed by your **answer** to that problem, one by one. The grading will be based on the correctness of your answers to the problems, and the **format**. Fail to comply with the aforementioned format (file name, header, problem, answer, problem, answer,…), will certainly degrade your score. If you have any questions, please feel free to ask me.

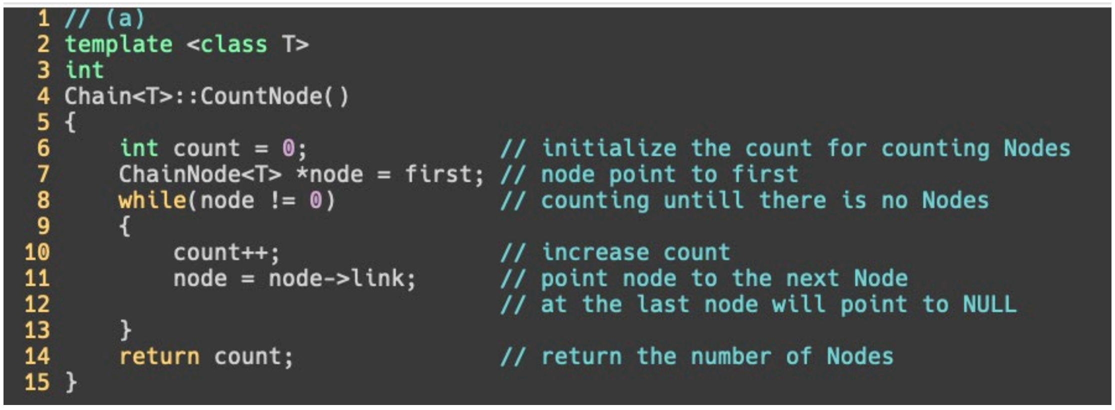
Part 1 (due 4/25/2021)

1. (30%) Given a template linked list **L** instantiated by the Chain class with a pointer **first** to the first node of the list as shown in Program 4.6 in textbook. The node is a ChainNode object consisting of a template data and link field.

```
template < class T > class Chain;    // 前向宣告

template < class T >
class ChainNode {
friend class Chain <T>;
private:
        T data;
        ChainNode<T>* link;
};

template <class T>
class Chain {
public:
        Chain( ) {first = 0;} // 建構子將 first 初始化成 0
        // 鏈的處理運算
        .
        .
private:
        ChainNode<T> * first;
}
```

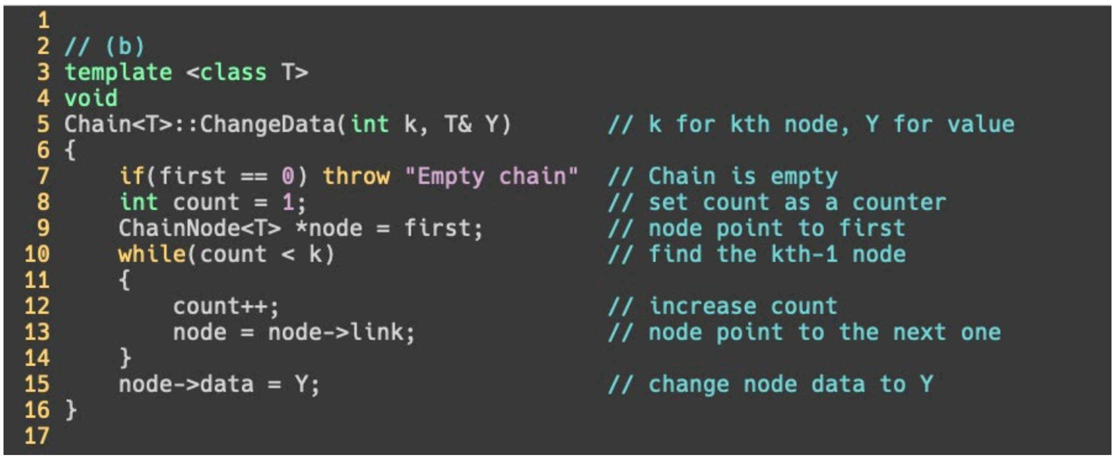
Program 4.6

(a) **Formulate an algorithm** (pseudo code OK, C++ code not necessary) which will count the number of nodes in L. Explain your algorithm properly (using either text or graphs).



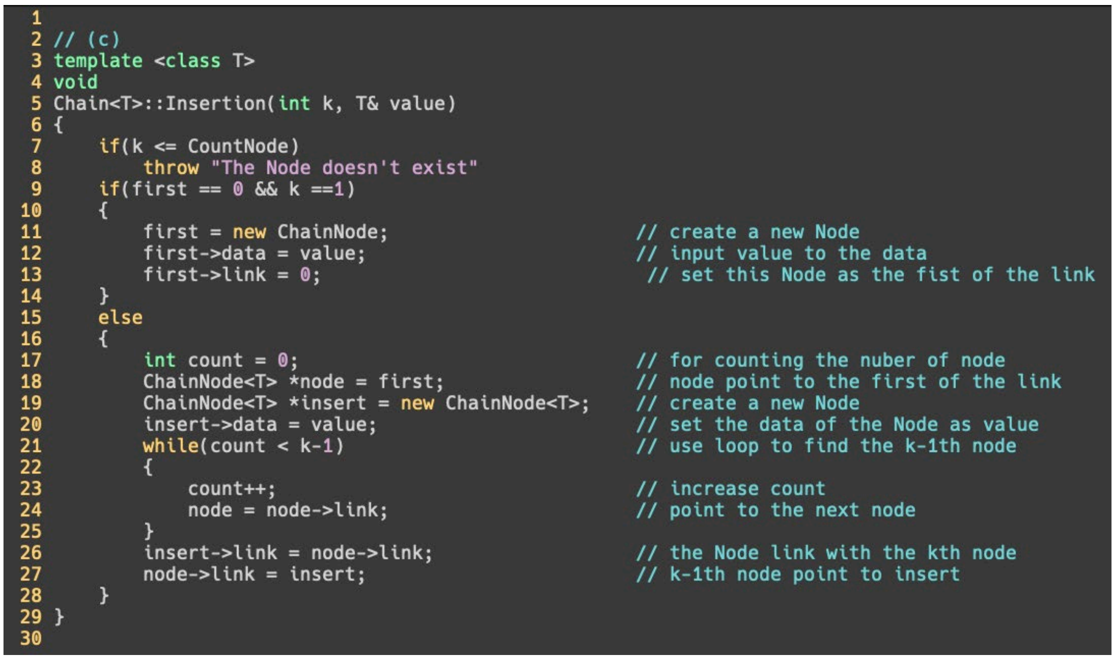
```
// (a)
template <class T>
int
Chain<T>::CountNode()
{
    int count = 0;                // initialize the count for counting Nodes
    ChainNode<T> *node = first; // node point to first
    while(node != 0)            // counting untill there is no Nodes
    {
        count++;                // increase count
        node = node->link;      // point node to the next Node
                                // at the last node will point to NULL
    }
    return count;               // return the number of Nodes
}
```

(b) **Formulate an algorithm** that will change the data field of **the kth node** of L to the value given by Y. Explain your algorithm properly (using either text or graphs).



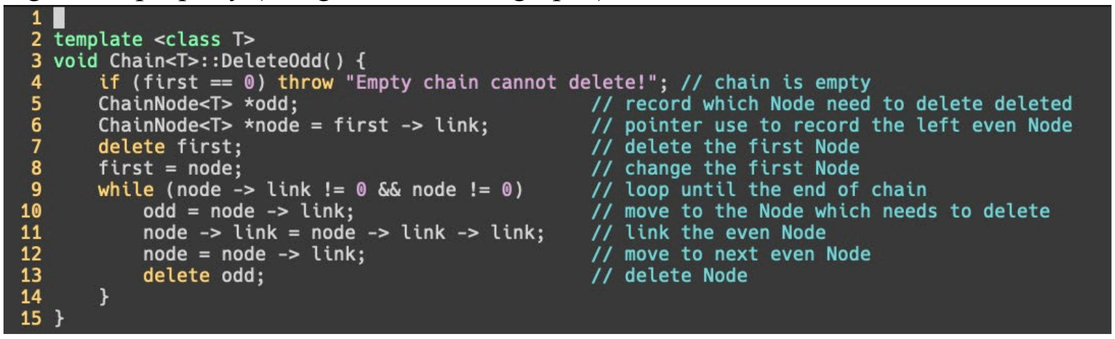
```

// (b)
template <class T>
void
Chain<T>::ChangeData(int k, T& Y)       // k for kth node, Y for value
{
    if(first == 0) throw "Empty chain"  // Chain is empty
    int count = 1;                      // set count as a counter
    ChainNode<T> *node = first;         // node point to first
    while(count < k)                    // find the kth-1 node
    {
        count++;                        // increase count
        node = node->link;              // node point to the next one
    }
    node->data = Y;                     // change node data to Y
}

```

(c) **Formulate an algorithm** that will perform an insertion to the immediate **before of the kth node** in the list L. Explain your algorithm properly (using either text or graphs).



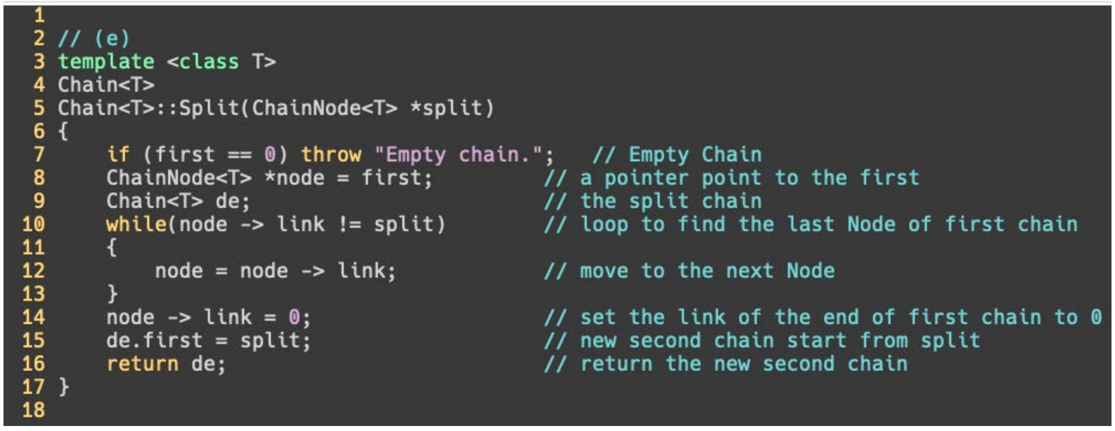
```cpp

// (c)
template <class T>
void
Chain<T>::Insertion(int k, T& value)
{
    if(k <= CountNode)
        throw "The Node doesn't exist"
    if(first == 0 && k ==1)
    {
        first = new ChainNode;                     // create a new Node
        first->data = value;                       // input value to the data
        first->link = 0;                            // set this Node as the fist of the link
    }
    else
    {
        int count = 0;                             // for counting the nuber of node
        ChainNode<T> *node = first;                // node point to the first of the link
        ChainNode<T> *insert = new ChainNode<T>;   // create a new Node
        insert->data = value;                      // set the data of the Node as value
        while(count < k-1)                         // use loop to find the k-1th node
        {
            count++;                               // increase count
            node = node->link;                     // point to the next node
        }
        insert->link = node->link;                 // the Node link with the kth node
        node->link = insert;                       // k-1th node point to insert
    }
}

```

(d) **Formulate an algorithm** that will **delete every other node** of L beginning with node first (i.e., the first, 3th, 5th,…nodes of L are deleted). Explain your algorithm properly (using either text or graphs).



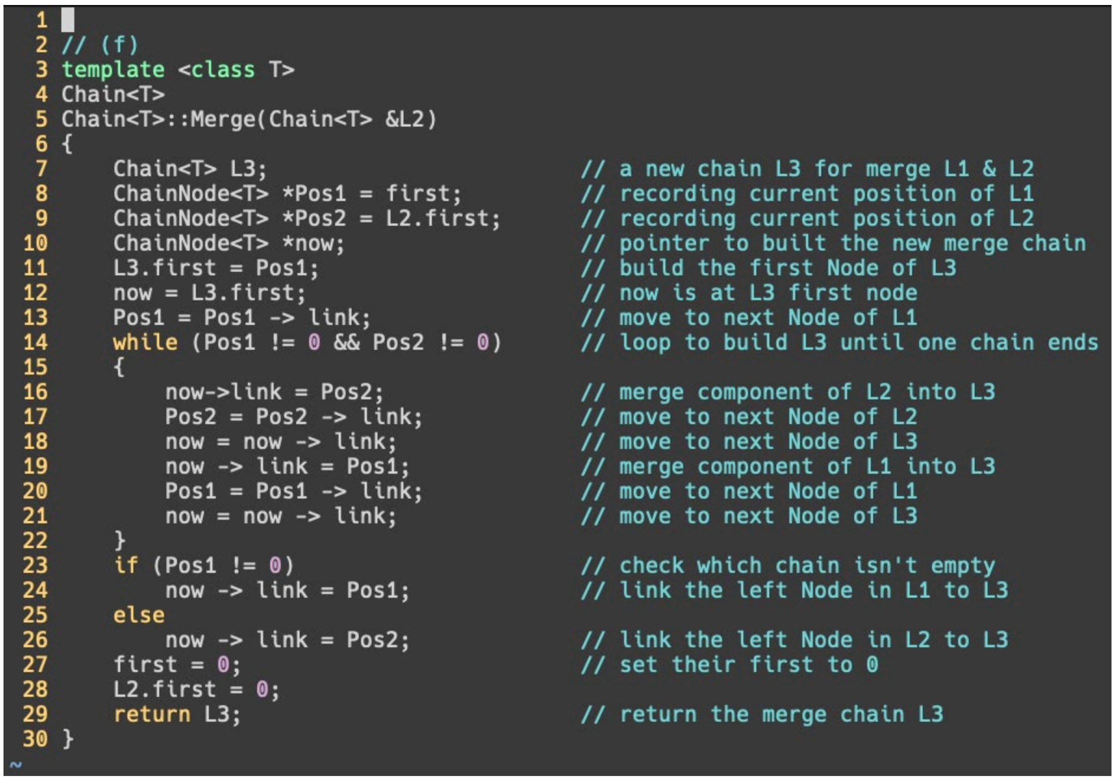
```cpp

template <class T>
void Chain<T>::DeleteOdd() {
    if (first == 0) throw "Empty chain cannot delete!"; // chain is empty
    ChainNode<T> *odd;                          // record which Node need to delete deleted
    ChainNode<T> *node = first -> link;         // pointer use to record the left even Node
    delete first;                               // delete the first Node
    first = node;                               // change the first Node
    while (node -> link != 0 && node != 0)      // loop until the end of chain
        odd = node -> link;                     // move to the Node which needs to delete
        node -> link = node -> link -> link;    // link the even Node
        node = node -> link;                    // move to next even Node
        delete odd;                             // delete Node
    }
}
```

(e) **Formulate an algorithm** that will deconcatenate (or split) a linked list L into two linked list. Assume the node denoted by the pointer variable split is to be the first node in the second linked list. Formulate a step-by-step algorithm to perform this task. Explain your algorithm properly (using either text or graphs).



```cpp

// (e)
template <class T>
Chain<T>
Chain<T>::Split(ChainNode<T> *split)
{
    if (first == 0) throw "Empty chain.";   // Empty Chain
    ChainNode<T> *node = first;          // a pointer point to the first
    Chain<T> de;                         // the split chain
    while(node -> link != split)         // loop to find the last Node of first chain
    {
        node = node -> link;             // move to the next Node
    }
    node -> link = 0;                    // set the link of the end of first chain to 0
    de.first = split;                    // new second chain start from split
    return de;                           // return the new second chain
}

```

(f) Assume L_1 and L_2 are two chains: $L_1 = (x_1,x_2,..,x_n)$ and $L_2 = (y_1,y_2,\dots,y_m)$, respectively. **Formulate an algorithm** that can merge the two chains together to obtain the chain $L_3 = (x_1,y_1,x_2,y_2,\dots,x_m,y_m,x_{m+1},..,x_n)$ if n>m and $L_3 = (x_1,y_1,x_2,y_2,\dots,x_n,y_n,y_{n+1},..,y_m)$ if n<m. Explain your algorithm properly (using either text or graphs).



```

// (f)
template <class T>
Chain<T>
Chain<T>::Merge(Chain<T> &L2)
{
    Chain<T> L3;                        // a new chain L3 for merge L1 & L2
    ChainNode<T> *Pos1 = first;         // recording current position of L1
    ChainNode<T> *Pos2 = L2.first;      // recording current position of L2
    ChainNode<T> *now;                  // pointer to built the new merge chain
    L3.first = Pos1;                    // build the first Node of L3
    now = L3.first;                     // now is at L3 first node
    Pos1 = Pos1 -> link;                // move to next Node of L1
    while (Pos1 != 0 && Pos2 != 0)      // loop to build L3 until one chain ends
    {
        now->link = Pos2;               // merge component of L2 into L3
        Pos2 = Pos2 -> link;            // move to next Node of L2
        now = now -> link;              // move to next Node of L3
        now -> link = Pos1;             // merge component of L1 into L3
        Pos1 = Pos1 -> link;            // move to next Node of L1
        now = now -> link;              // move to next Node of L3
    }
    if (Pos1 != 0)                      // check which chain isn't empty
        now -> link = Pos1;             // link the left Node in L1 to L3
    else
        now -> link = Pos2;             // link the left Node in L2 to L3
    first = 0;                          // set their first to 0
    L2.first = 0;
    return L3;                          // return the merge chain L3
}
```

2. (55%) Given a circular linked list L instantiated by class CircularList containing a private data member, **first** pointing to the first node in the circular list as shown in Figure 4.14.

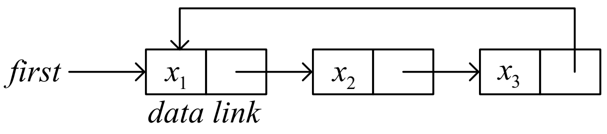


Fig. 4.14 A circular linked list

formulate algorithms (pseudo code OK, C++ code not necessary) to

(a) count the number of nodes in the circular list. Explain your algorithm properly (using either text or graphs)

[pseudo code]

```
if(first == 0) return 0;
else {
```

```
Start from first;
Number = 1;
while (next node != first) {
    go to next node;
    Number++;
}
return Number;
}
```

從 first 開始算，如果下個 node 不是 first，就往下一個計算，最後得到 node 數量。

(b) insert a new node at the front of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
if(first == 0) add a new node;
else {
    Start from first;
    while (next node != first) {
            go to next node;
    }
    Add new node behind this node;
    Link new node to first;
    New node becomes new first;
}
```

Time complexity:
O(1), where n = 0
O(n), where n means the number of Nodes

先判斷有沒有 node。
找到最後一個 node，加入一個新的 node 在最後，並串連到第一個 node，並把 first 指向新加入的 node。

(c) insert a new node at the back (right after the last node) of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
if(first == 0) add a new node;
else {
   Start from first;

   while (next node != first) {
         go to next node;
   }
   Add new node behind this node;
   Link new node to first;
}
```

Time complexity:
O(1), where n = 0
O(n), where n means the number of Nodes

找到最後一個 node，加入一個新的 node 在最後，並串連到第一個 node

(d) delete the first node of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
if(first->link == first) {
   delete first;
   first = 0;
} else {
   Start from first;

   while (next node != first) {
         go to next node;
   }
   Link this node to first's next;
   Delete first;
   New first = this node's next;
}
```

Time complexity:
O(1), where n = 0
O(n), where n means the number of Nodes

找到最後一個 node，將它 link 到 first 的下一個，刪掉舊的 first，設定新的 first。

(e) delete the last node of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs).

```
if(first->link == first) {
   delete first;
   first = 0;
} else {
   Start from first;

   while (next node != first) {
         remember this node;
      go to next node;
   }
   Link remembered node to first;
   Delete this node;
}
```

Time complexity:
Ｏ(1), where n = 0
Ｏ(n), where n means the number of Nodes

找到最後一個 node 以及倒數第二個 node，將倒數第二個接到 first，刪掉最後一個。

(f) Repeat (a) – (f) in Problem 1 above if the circular list is modified as shown in Figure 4.16 below by introducing a dummy node, header.

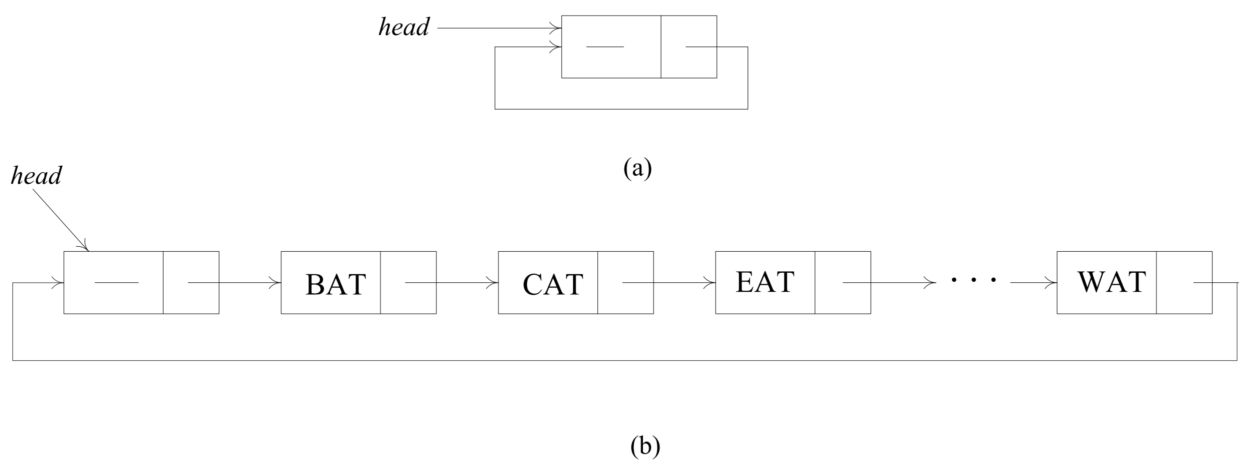


Figure 4.16 Circular list with a header node

(a) count the number of nodes in the circular list. Explain your algorithm properly (using either text or graphs)

```
template <class T>
int CircularList <T>::Size() {
  int numOfNodes = 1;
     for (ChainNode<T>* ptr = head -> link; ptr != head; ptr = ptr -> link)
numOfNodes++;
      return numOfNodes;
}
```

Explanation: Travel through L and count. The head node is a node so count from 1, if ptr back to head means we are done.

(b) insert a new node at the front of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
template <class T>
void CircularList <T>::Push_Front() {
  ChainNode<T>* NewNode = new ChainNode<T>;
  NewNode -> link = head -> link;
  head -> link = NewNode;
}
```

Explanation: First we have to create a new node link to the front of the chain then head -> link link to NewNode.

Complexity: O(1) the time is always the constant.

(c)insert a new node at the back (right after the last node) of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
template <class T>
void CircularList <T>::Push_Back() {
   ChainNode<T>* ptr = head -> link;
   for (;ptr -> link != head; ptr = ptr -> link);
   ChainNode<T>* NewNode = new ChainNode<T>;
   NewNode -> link = head;
   ptr -> link = NewNode;
}
```

Explanation: We have to find the last node link to a new node then the new node link to head.

Complexity: If there are n elements in L, the time complexity is O(n) because we need to find the last element.

(d)

delete the first node of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)

```
template <class T>
void CircularList <T>::Delete_Front() {
  if (Size() == 0) throw invalid_argument("The chain is empty cannot
delete!");
  ChainNode<T>* ptr = head -> link;
  head -> link = ptr -> link;
  delete ptr;
}
```

Explanation: If the chain is empty we cannot delete. Use a pointer point to the front of the chain and let the second node become the first one after that we can delete the first node.

Complexity: O(1) the time is always the constant.

(e)
delete the last node of the list. Discuss the time complexity of your algoithm.
Explain your algorithm properly (using either text or graphs).

```
template <class T>
void CircularList <T>::Delete_Back() {
        if (Size() == 0) throw invalid_argument("The chain is empty cannot
     delete!");
         ChainNode<T>* ptr = head;
         for (;ptr -> link -> link != head; ptr = ptr -> link);
         delete ptr -> link;
         ptr -> link = head;
}
```

Explanation: We need to find the node before the last one then delete the last node after that link the node before the last node to head because it becomes the last one.

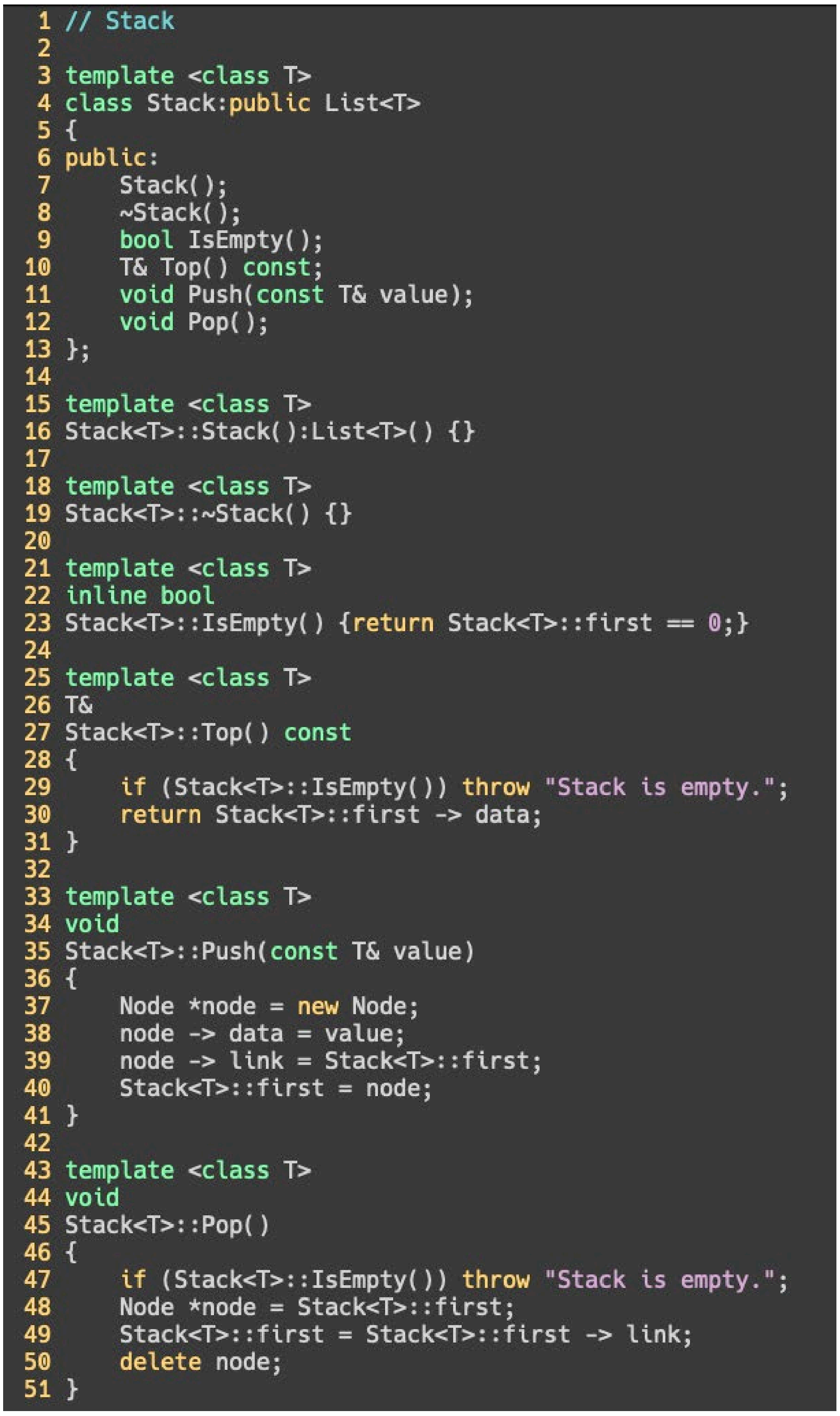
Complexity: If there are n elements in L, the time complexity is O(n) because we need to find the last element.

3. (15%) The class List<T> is shown below,

```
template <class T> class List;
template <class T>
class Node{
friend class List<T>;
private:    T data;
            Node* link;
};
template <class T>
class List{
public:
      List(){first = 0;}
      void InsertBack(const T& e);
```

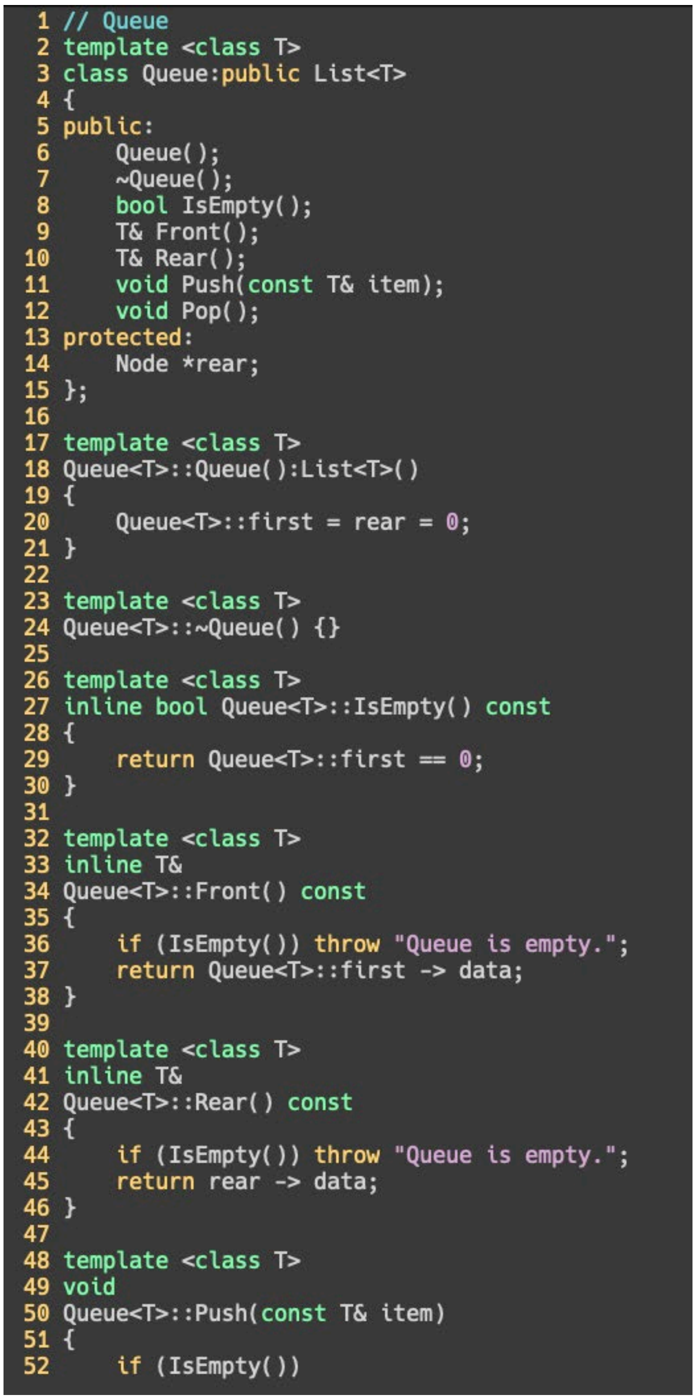
```
        void Concatenate(List<T>& b);
        void Reverse();
        class Iterator{
        ….
        };
        Iterator Begin();
        Iterator End();
private:
        Node* first;
};
```

(a) Implement the stack data structure as a derived class of the class List<T>.

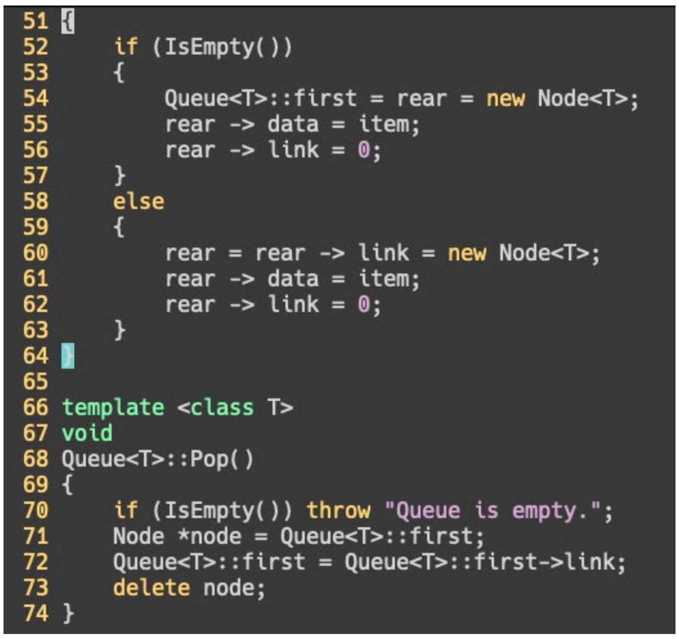


```cpp
// Stack

template <class T>
class Stack:public List<T>
{
public:
    Stack();
    ~Stack();
    bool IsEmpty();
    T& Top() const;
    void Push(const T& value);
    void Pop();
};

template <class T>
Stack<T>::Stack():List<T>() {}

template <class T>
Stack<T>::~Stack() {}

template <class T>
inline bool
Stack<T>::IsEmpty() {return Stack<T>::first == 0;}

template <class T>
T&
Stack<T>::Top() const
{
    if (Stack<T>::IsEmpty()) throw "Stack is empty.";
    return Stack<T>::first -> data;
}

template <class T>
void
Stack<T>::Push(const T& value)
{
    Node *node = new Node;
    node -> data = value;
    node -> link = Stack<T>::first;
    Stack<T>::first = node;
}

template <class T>
void
Stack<T>::Pop()
{
    if (Stack<T>::IsEmpty()) throw "Stack is empty.";
    Node *node = Stack<T>::first;
    Stack<T>::first = Stack<T>::first -> link;
    delete node;
}
```

(b) Implement the queue data structure as a derived class of the class List<T>.

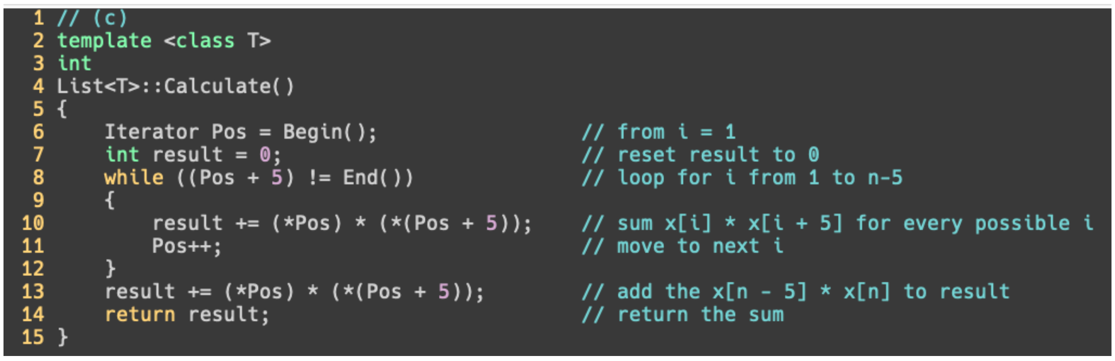


```cpp
// Queue
template <class T>
class Queue:public List<T>
{
public:
    Queue();
    ~Queue();
    bool IsEmpty();
    T& Front();
    T& Rear();
    void Push(const T& item);
    void Pop();
protected:
    Node *rear;
};

template <class T>
Queue<T>::Queue():List<T>()
{
    Queue<T>::first = rear = 0;
}

template <class T>
Queue<T>::~Queue() {}

template <class T>
inline bool Queue<T>::IsEmpty() const
{
    return Queue<T>::first == 0;
}

template <class T>
inline T&
Queue<T>::Front() const
{
    if (IsEmpty()) throw "Queue is empty.";
    return Queue<T>::first -> data;
}

template <class T>
inline T&
Queue<T>::Rear() const
{
    if (IsEmpty()) throw "Queue is empty.";
    return rear -> data;
}

template <class T>
void
Queue<T>::Push(const T& item)
{
    if (IsEmpty())
```



```cpp
{
    if (IsEmpty())
    {
        Queue<T>::first = rear = new Node<T>;
        rear -> data = item;
        rear -> link = 0;
    }
    else
    {
        rear = rear -> link = new Node<T>;
        rear -> data = item;
        rear -> link = 0;
    }
}

template <class T>
void
Queue<T>::Pop()
{
    if (IsEmpty()) throw "Queue is empty.";
    Node *node = Queue<T>::first;
    Queue<T>::first = Queue<T>::first->link;
    delete node;
}
```

(c) Let $x_1, x_2, \ldots, x_n$ be the elements of a List<int> object. Each x_i is an integer. Formulate an algorithm (pseudo code OK, C++ code not necessary) to compute the expression $\sum_{i=1}^{n-5}(x_i \times x_{i+5})$



```cpp
// (c)
template <class T>
int
List<T>::Calculate()
{
    Iterator Pos = Begin();                     // from i = 1
    int result = 0;                             // reset result to 0
    while ((Pos + 5) != End())                  // loop for i from 1 to n-5
    {
        result += (*Pos) * (*(Pos + 5));        // sum x[i] * x[i + 5] for every possible i
        Pos++;                                  // move to next i
    }
    result += (*Pos) * (*(Pos + 5));            // add the x[n - 5] * x[n] to result
    return result;                              // return the sum
}
```

Part 2 Coding (due 5/8)

You should submit:

(a) All your source codes (C++ file).

(b) Show the execution trace of your program.

1. (30%) Fully code and test the C++ template class List<T> shown in Part 1 Problem 3 above. You must include:
   a. A constructor which constructs an initially empty list.
   b. A destructor which deletes all nodes in the list.
   c. InsertFront() function to insert at the front of the list.

d. DeleteFront() and DeleteBack() to delete from either end.
e. Front() and Back() functions to return the first and last elements of the list, respectively.
f. A function Get(int i) that returns the ith element in the list.
g. Delete(int i) to delete the ith element
h. Insert(int i, T e) to insert as the ith element
i. Overload the output operator << to out put the List object.
j. As well as functions and forward iterator as shown above.

Write a client program (main()) to **demonstrate** those functions you developed.

2. (35%) Develop a C++ class Polynomial to represent and manipulate univariate polynomials with double coefficients (use circular linked list with header nodes). Each term of the polynomial will be represented as a node. Thus a node in this system will have three data members as below.

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an **available-space list** and associated functions GetNode() and RetNode() described in Section 4.5. The external (i.e., for input and output) representation of a univariate polynomial will be assumed to be a sequence of integers and doubles of the form: n, c_1, e_1, c_2, e_2, c_3, e_3,…, c_n, e_n, where e_i represents an integer exponent and c_i a double coefficient; n gives the number of terms in the polynomial. The exponents of the polynomial are in decreasing order.

Write and **test** the following functions:
(a) Istream& operator>>(istream& is, Polynomial& x): Read in an input polynomial and convert it to its circular list representation using a header node.
(b) Ostream& operator<<(ostream& os, Polynomial& x): Convert x from its linked list representation to its external representation and output it.
(c) Polynomila::Polynomial(const Polynomial& a): copy constructor
(d) Const Polynomila& Polynomial::operator=(const Polynomial& a) const[assignment operator]: assign polynomial a to *this.
(e) Polynomial::~ Polynomial(): desctructor, return all nodes to available-space list

(f) Polynomial operator+ (const Polynomial& b) const: Create and return the polynomial *this + b
(g) Polynomial operator- (const Polynomial& b) const: Create and return the polynomial *this – b
(h) Polynomial operator* (const Polynomial& b) const: Create and return the polynomial *this * b
(i) double Polynomial::Evaluate(double x) const: Evaluate the polynomial *this and return the result.

3. (35%) The class definition for sparse matrix in Program 4.29 is shown below.

```
struct Triple{int row, col, value;};
class Matrix; // 前向宣告
class MatrixNode {
friend class Matrix;
friend istream& operator>>(istream&, Matrix&); // 為了能夠讀進矩陣
private:
    MatrixNode *down , *right;
    bool head;
    union { // 沒有名字的 union
        MatrixNode *next;
        Triple triple;
    };
    MatrixNode(bool, Triple*); // 建構子
}

MatrixNode::MatrixNode(bool b, Triple *t)   // 建構子
{
    head = b;
    if (b) {right = down = this;} // 列/行的標頭節點
    else triple = *t; // 標頭節點串列的元素節點或標頭節點
}

class Matrix{
friend istream& operator>>(istream&, Matrix&);
public:
    ~Matrix(); // 解構子
private:
    MatrixNode *headnode;
};
```

Based on this class, do the following tasks.
(a) Write the C++ function, **operator**+(**const** Matrix& b) **const**, which returns the

matrix ***this** + b.

(b) Write the C++ function, **operator***(const Matrix& b) **const**, which returns the matrix ***this** * b.

(c) Write the C++ function, **operator**<<(), which outputs a sparse matrix as triples (i, j, a_{ij}).

(d) Write the C++ function, Transpose(), which transpose a sparse matrix.

(e) Write and test a copy constructor for sparse matrices. What is the computing time of your copy constructor?

Write a client program (main()) to **demonstrate** those functions you developed.