**EECS2040 Data Structure Hw #3 (Chapter 4 Linked List)**

**due date 4/18/2022 by 109070025 林泓錩**

**Part 1 (2% of final Grade, due 4/18/2022)**

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 (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

1. **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).

**Ans:**

template<class T>

int Chain<T> :: Size()

{

int count = 0;

ChainNode<T> \*current = first;

while (current != 0)

{

count++;

current = current->link;

}

return count;

}

Let current = first, then if current is not 0, count++. Next, let current point to next node, if current = 0 => run out all element, return count.

1. **Formulate an algorithm** that will change the data field of **the kth node** (the first 1st node start at index 0) of L to the value given by Y. Explain your algorithm properly (using either text or graphs).

**Ans:**

template<class T>

void Chain<T> :: ChangeY(int k, const T& Y)

{

ChainNode<T> \*current = first;

if (first == 0) throw “error”;

while (k > 1)

{

current = current->link;

k--;

if (current == 0) return;

}

current->data = Y;

}

Let current = first, and if k is 1 and chain is not empty, we change first

node’s data field. If k > 1, we use loop by decreasing 1 of k meanwhile move current to the next node to find kth node. In the long run, if kth node is find, we change its data field.

1. **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).

**Ans:**

template<class T>

void Chain<T> :: Insert(int k, const T& x)

{

ChainNode<T> \*current = first, \*precedent = 0;

if (k == 1) first = new ChainNode<T>(x, first);

else

{

while (k > 1)

{

k--;

precedent = current;

current = current->link;

if (current == 0) return;

}

precedent ->link = new ChainNode<T>(x, current);

}

}

If k is 1, we construct new Chaincode with data is x, and it’s link field is point to first. If k>1, we find out kth node, construct new node with data field is x and link field point to kth node, and make precedent’s link field point to new node.

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

**Ans:**

template<class T>

void Chain<T> :: Delete()

{

if (first == 0) throw “error”;

ChainNode<T> \*current = first->link, \*Del;

while (current != 0)

{

Del = current->link;

if (Del == 0) break;

current->link = current->link->link;

current = current->link;

delete Del;

}

delete first;

}

We make current point to second node, if current != 0, We enter the loop, make Del = current->link. If Del is 0, we exit loop. If Del not 0, we make current->link point to the node that after two node of current. And make current = current->link, and delete Del. After loop is finish, we delete first

1. **Formulate an algorithm** divideMid that will divides the given list into two sublists of (almost) equal sizes. Suppose myList points to the list with elements 34 65 27 89 12 (in this order). The statement: myList.divideMid(subList); divides myList into two sublists: myList points to the list with the elements 34 65 27, and subList points to the sublist with the elements 89 12. Formulate a step-by-step algorithm to perform this task. Explain your algorithm properly (using either text or graphs).

**Ans:**

template<class T>

void Chain<T> :: divideMid(ChainNocde<T> \*dimid)

{

int count = 0;

ChainNode<T> \*current = first;

while (current != 0)

{

count++;

current = current->link;

}

int mid;

if(count/2==0) mid=count/2;

else mid = count/2 + 1;

ChainNode<T> \*current = first;

while (current->link != split)

{

current = current->link;

}

current->link = 0;

}

We let current = first, and use the way we do at (a) to count the size of nodes,

then we find the middle value of size, next that the current be it, and make its link be 0, by doing this, we make linked list into myList and subList.

1. **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).

**Ans:**

template<class T>

void Chain<T> :: Split(ChainNocde<T> \*split)

{

ChainNode<T> \*current = first;

while (current->link != split)

{

current = current->link;

}

current->link = 0;

}

We let current = first, and if the next node of current is split, we change

current’s link field to 0. Then we can split linked list into two different linked

list.

1. Assume L1 and L2 are two chains: L1 = (x1,x2,..,xn) and L2 = (y1,y2,…,ym), respectively. **Formulate an algorithm** that can **merge** the two chains together to obtain the chain L3 = (x1,y1,x2,y2,…,xm,ym,xm+1,..,xn) if n>m and L3 = (x1,y1,x2,y2,…,xn,yn,yn+1,..,ym) if n<m. Explain your algorithm properly (using either text or graphs).

**Ans:**

template<class T>

void Chain<T> :: merge(ChainNode<T> \*first\_x, ChainNode<T> \*first\_y)

{

ChainNode<T> \*current1 = first\_x, \*current2 = first\_y, \*temp1, \*temp2;

while (current1->link != 0 && current2->link != 0)

{

temp1 = current1->link;

temp2 = current2->link;

current1->link = current2;

current1 = temp1;

current2->link = temp1;

current2 = temp2;

}

if (current1->link == 0) current1->link = current2;

else

{

current2->link = current1->link;

current1->link = current2;

}

first\_y = first\_x;

}

We let current1 = first\_x, current2 = first\_y, and use loop and temp1 and temp2 to arranged the nodes in first\_x and first\_y crisscross, and watch which current is gonna be 0 first, then arrange the rest of the other after the arrangement, last make first\_y = first\_x => the start.

1. (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

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

**Ans:**

template<class T>

int Chain<T> :: Size()

{

int size = 1;

ChainNode<T> \*current = first;

while (current->link != first)

{

current = current->link;

size++;

}

return size;

}

At first let current = first, and then when current->link == first means we

have run all element of list, so we return size.

1. 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)

**Ans:**

template<class T>

void Chain<T> :: Insertfront(int x)

{

ChainNode<T> \*temp = new ChainNode<T>(x, first);

first = temp;

temp = first->link;

while (temp->link != first->link)

temp = temp->link;

temp->link = first;

}

Complexity: O(n)

For this is the curricular linked list, when we insert a new node at the front, the original first become the last and have to link the list, so another nodes have to move on, too, so the Complexity is O(n).

1. 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)

Ans:

template<class T>

void Chain<T> :: Insertback(int x)

{

ChainNode<T> \*temp1 = new ChainNode<T>(x, first), \*temp2;

temp2 = first;

while (temp2->link != first)

temp2 = temp2->link;

temp2->link =temp1;

}

Complexity: O(n)

For the same reason in (c).

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

**Ans:**

template<class T>

void Chain<T> :: Deletefront()

{

ChainNode<T> \*current = first;

while (current->link != first)

current = current->link;

current->link = first->link;

delete first;

first = current->link;

}

Complexity: O(n)

For the same reason in (c).

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

**Ans:**

template<class T>

void Chain<T> :: Deletelast()

{

ChainNode<T> \*current = first->link, \*previous = first;

while (current->link != first)

{

current = current->link;

previous = previous->link;

}

previous->link = first;

delete current;

}

Complexity: O(n)

For the same reason in (c).

1. Repeat (a) – (e) above and (b) – (g) 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

**Ans:**

(a)

template<class T>

int Chain<T> :: Size()

{

int size = 1;

ChainNode<T> \*current = head->link;

while (current->link != head->link )

{

current = current->link;

size++;

}

return size;

}

(b)

template<class T>

void Chain<T> :: Insertfront(int x)

{

ChainNode<T> \*temp = new ChainNode<T>(x, head->link);

head->link = temp;

temp = first->link;

while (temp->link != head->link ->link)

temp = temp->link;

temp->link = head->link;

}

Complexity: O(n)

(c)

template<class T>

void Chain<T> :: Insertback(int x)

{

ChainNode<T> \*temp1 = new ChainNode<T>(x, head->link), \*temp2;

temp2 = head->link;

while (temp2->link != head->link)

temp2 = temp2->link;

temp2->link =temp1;

}

Complexity: O(n)

(d)

template<class T>

void Chain<T> :: Deletefront()

{

ChainNode<T> \*current = head->link;

while (current->link != head->link)

current = current->link;

current->link = head->link ->link;

delete head->link;

head->link = current->link;

}

Complexity: O(n)

(e)

template<class T>

void Chain<T> :: Deletelast()

{

ChainNode<T> \*current = head->link ->link, \*previous = head->link;

while (current->link != head->link)

{

current = current->link;

previous = previous->link;

}

previous->link = head->link;

delete current;

}

Complexity: O(n)

(b)

template<class T>

void Chain<T> :: ChangeY(int k, const T& Y)

{

ChainNode<T> \*current = head->link;

if (k == 1) current->data = Y;

else

{

while (k > 1)

{

current = current->link;

k--;

}

current->data = Y;

}

}

If k is 1, then we change the data field of head->link to Y. If k>1, then we

find out the kth node, change its data to Y

(c)

Ans:

template<class T>

void Chain<T> :: Insert(int k, const T& x)

{

Node<T> \*current = head->link, \*previous = head;

while (k > 1)

{

k--;

previous = previous->link;

current = current->link;

}

previous->link = new ChainNode<T>(x, current);

}

(d)

template<class T>

void Chain<T> :: Delete()

{

Node<T> \*current = head->link->link, \*Del = head->link;

head->link = current;

delete Del;

while (current != head)

{

Del = current->link;

if (Del == head) break;

current->link = current->link->link;

current = current->link;

delete Del;

}

}

(e)

template<class T>

void Chain<T> :: divideMid(ChainNocde<T> \*dimid)

{

int count = 0;

ChainNode<T> \*current = head->link;

while (current != 0)

{

count++;

current = current->link;

}

int mid;

if(count/2==0) mid=count/2;

else mid = count/2 + 1;

ChainNode<T> \*current = head->link t;

while (current->link != split)

{

current = current->link;

}

current->link = 0;

}

(f)

template<class T>

ChainNode<T>\* Chain<T> :: Split(ChainNocde<T> \*split)

{

ChainNode<T> \*current = head->link, \*head2, last = head->link;

while (last->link != head) last = last->link;

while (current->link != split)

{

current = current->link;

}

current->link = head;

head2->data = -1;

head2->link = split;

last->link = head2;

return head2;

}

(g)

template<class T>

void Chain<T> :: merge(ChainNode<T> \*head\_x, ChainNode<T> \*head\_y)

{

ChainNode<T> \*current1 = head\_x->link, \*current2 = head\_y->link, \*temp1, \*temp2;

while (current1->link != head\_x && current2->link != head\_y)

{

temp1 = current1->link;

temp2 = current2->link;

current1->link = current2;

current1 = temp1;

current2->link = temp1;

current2 = temp2;

}

if (current1->link == head\_x)

{

current1->link = current2;

head\_y->link = head\_x->link;

head\_x = head\_y;

}

else

{

current2->link = current1->link;

current1->link = current2;

head\_y = head\_x;

}

}

1. (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;

};

1. Implement (pseudo code or C++) the stack data structure as a derived class of the class List<T>.

**Ans:**

template<class T>

class Stack : public List<T>

{

public:

Stack();

void Pop();

T Top();

void Push(const T& e)

private:

Node<T>\* top;

};

template<class T>

Stack<T> :: Stack() :: List<T>(){ top = 0; }

template <class T>

void Stack<T> :: Push(const T &e)

{

top = new Node<T>(e, top);

}

template <class T>

void Stack<T> :: Pop()

{

If (top == 0) throw “Error”;

Node<T> \*temp = top->link;

delete top;

top = temp;

}

template<class T>

T Stack<T> :: Top()

{

return top->data;

}

1. Implement (pseudo code or C++) the queue data structure as a derived class of the class List<T>.

**Ans:**

template <class T>

class Queue : public List<T>

{

public:

Queue ();

T Rear();

T front();

void Push(const T& e);

void Pop();

private:

Node<T> \*front, \*rear;

};

template<class T>

Queue<T> :: Queue() : List<T>()

{

front = 0;

rear = 0;

}

template <class T>

void Queue<T> :: Push(const T& e) : List<T> :: InsertBack(const T& e)

{

Node<T> \*current = first;

while (current->link != 0) current = current->link;

rear = current;

}

template <class T>

void Queue<T> :: Pop()

{

if (rear == 0 && front == 0) throw “Error”;

Node<T> \*current = front->link;

delete front;

front = current;

}

template <class T>

T Queue<T> :: Rear()

{

return rear->data;

}

template <class T>

T Queue<T> :: Front()

{

return front->data;

}

1. Let x1, x2,…, xn be the elements of a List<int> object. Each xi is an integer. Formulate an algorithm (pseudo code OK, C++ code not necessary) to compute the expression

**Ans:**

int List<int> :: sum()

{

Node<int> \*previous = first, \*current = first->link->link->link->link->link;

int total = 0;

while (current != 0)

{

total += previous->data \* current->data;

previous = previous->link;

current = current->link;

}

return total;

}