HW4 電資二賴庭岳 4112064228

1.Let x = (x1, x2, . . . , xn) and y = (y1, y2, . . . , yn) …

```
#include<iostream>
using namespace std;
struct node
{
    int value;
    node *right=0;
};
class chain
{
private:
    node *last;
    node *first;
public:
    chain() : last(0), first(0) {}
    ~chain(){
       node* current = first;
        while (current != nullptr) {
            node* temp = current;
            current = current->right;
            delete temp;
        }
    }
    void insert(int val);
    int top(){
        return last->value;
    }
    void show_all(){
        node *tmp=first;
        while(tmp!=NULL){
            cout<<tmp->value<<" ";
            tmp=tmp->right;
        }
        cout<<endl;
    }
    //void merge(const chain& b);
    node* First() const {
        return first;
    };
};
```

```
void chain::insert(int val){
    node *temp = new node();
    temp->value = val;
    temp->right = nullptr;
    if (first == nullptr) {
        first = temp;
        last = temp;
    } else {
        last->right = temp;
        last = temp;
    }
    return ;
}
chain& merge(chain& a,chain& b){
    node* tmpa=a.First();
    node* tmpb=b.First();
    static chain c;
    while(tmpa!=NULL or tmpb!=NULL){
        if(tmpa!=NULL){
            c.insert(tmpa->value);
            tmpa=tmpa->right;
        }
        if(tmpb!=NULL){
            c.insert(tmpb->value);
            tmpb=tmpb->right;
        }
    }
    return c;
}
int main(){
    chain a,b,c;
    a.insert(1);a.insert(3);a.insert(5);
    //a.show_all();
    b.insert(2);b.insert(4);b.insert(6);
    c=merge(a,b);
    c.show_all();
    //cout<<"123";
}
```

2. Let x = (x1, x2, . . . , xn) and y = (y1, y2, . . . , yn) …

```
#include<iostream>
using namespace std;
struct node{
    int value;
    node *right=0;
};
class chain{
private:
    node *last;
    node *first;
public:
    chain() : last(0), first(0) {}
```

```
chain& merge(chain& a,chain& b){
    node* tmpa=a.First();
    node* tmpb=b.First();
    static chain c;
    while(tmpa!=NULL or tmpb!=NULL){
        if(tmpa==NULL){
            c.insert(tmpb->value);
            tmpb=tmpb->right;
            continue;
        }
        if(tmpb==NULL){
            c.insert(tmpa->value);
    ~chain(){
       node* current = first;
        while (current != nullptr) {
            node* temp = current;
            current = current->right;
            delete temp;
        }
    }
    void insert(int val);
    int top(){return last->value;}
    void show_all(){
        node *tmp=first;
        while(tmp!=NULL){
            cout<<tmp->value<<" ";
            tmp=tmp->right;
        }
        cout<<endl;
    }
    node* First() const {return first;};
};
void chain::insert(int val){
    node *temp = new node();
    temp->value = val;
    temp->right = nullptr;
    if (first == nullptr) {
        first = temp;
        last = temp;
    } else {
        last->right = temp;
        last = temp;
    }
    return ;
}
```

```
            tmpa=tmpa->right;
            continue;
        }
        if(tmpa->value > tmpb->value ){
            c.insert(tmpb->value);
            tmpb=tmpb->right;
            continue;
        }
        if(tmpb->value > tmpa->value ){
            c.insert(tmpa->value);
            tmpa=tmpa->right;
            continue;
        }
    }
    return c;
}
int main(){
    chain a,b,c;
    a.insert(3);
    a.insert(4);
    a.insert(5);
    //a.show_all();
    b.insert(1);
    b.insert(7);
    b.insert(8);
    c=merge(b,a);
    c.show_all();
    //cout<<"123";
}
```

3.Write a C++ template function to output all elements of a chain…

```
#include<iostream>
using namespace std;
template<class T>
class chain{
private:
  struct node{
    T value;
    node *right;
  };
  node *last;
  node *first;
public:
  friend std::ostream &operator<<(std::ostream
&os ,const chain<T> &x){
    node *tmp=x.First();
    while(tmp!=NULL){
      os<<tmp->value<<" ";
      tmp=tmp->right;
    }
    os<<endl;
    return os;
```

```
      void insert(T val);
    T top(){
        return last->value;
    }
    void show_all(){
        node *tmp=first;
        while(tmp!=NULL){
            cout<<tmp->value<<" ";
            tmp=tmp->right;
        }
        cout<<endl;
    }
    node* First() const {
        return first;
    };
};
template<class T>
void chain<T>::insert(T val){
    node *temp = new node();
    temp->value = val;
    temp->right = nullptr;
```

```
  }
chain() : last(NULL), first(NULL) {}
    ~chain(){
       node* current = first;
        while (current != nullptr) {
            node* temp = current;
            current = current->right;
            delete temp;
        }
    }
```

```
    if (first == nullptr) {
        first = temp;
        last = temp;
    } else {
        last->right = temp;
        last = temp;
    }
    return ;
}
int main(){
  chain<char> a,b,c;
  a.insert('a');
  a.insert('1');
  a.insert('b');
  cout<<a;
}
```

4.Let x = (x1, x2, . . . , xn) be the elements of a chain…

```
#include <iostream>
using namespace std;
template<class T>
struct node {
    T value;
    node *right;
};
template<class T>
class chain {
private:
    node<T> *last;
    node<T> *first;

public:
    chain() : first(nullptr), last(nullptr) {}
    void insert(T val);
    node<T>* First() const { return first; }

    void copyToArray(T* arr, size_t maxSize) {
        node<T>* current = first;
        size_t index = 0;
        while (current != nullptr && index <
maxSize) {
            arr[index++] = current->value;
            current = current->right;
        }
    }
};
```

```
template<class T>
void chain<T>::insert(T val) {
    node<T> *temp = new node<T>();
    temp->value = val;
    temp->right = nullptr;
    if (first == nullptr) {
        first = temp;
        last = temp;
    } else {
        last->right = temp;
        last = temp;
    }
}
int main() {
    chain<char> a;
    a.insert('a');
    a.insert('1');
    a.insert('b');

    char arr[3];
    a.copyToArray(arr, 3);

    for (int i = 0; i < 3; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

5.Do Exercise 1 of Section 4.3 for the case of circularly linked lists.

```
#include <iostream>
using namespace std;
template<class T>
struct node {
```

```
template<class T>
chain<T>& merge(const chain<T> a,const
chain<T> b){
  node<T>* tmpa=a.First();
```

```
    T value;
    node *right;
    bool s;
};
template<class T>
class chain {
private:
    node<T> *last;
    node<T> *first;

public:
  chain() : first(nullptr), last(nullptr) {}
  void insert(T val);
  T top(){return first->value;}
  node<T>* First() const { return first; }
  void show(){
    node<T>*tmp =first;
    int  k=1;
    while(tmp->s or k){
      cout<<tmp->value<<" ";
      tmp=tmp->right;
      k=0;
    }
  }
};
template<class T>
void chain<T>::insert(T val) {
  node<T> *temp = new node<T>();
  temp->value = val;
  temp->right = nullptr;
  temp->s=1;
  if (first == nullptr) {
    temp->s=false;
    first = temp;
    last = temp;
  } else {
    last->right = temp;
    last = temp;
    last->right=first;
  }
}
```

```
  node<T>* tmpb=b.First();
  static chain<T> v;
  int k=1;
  while(tmpa->s or tmpb->s or k){
    if(tmpa->s or k){
      v.insert(tmpa->value);
      tmpa=tmpa->right;
    }
    if(tmpb->s or k){
      v.insert(tmpb->value);
      tmpb=tmpb->right;
    }
    k=0;
    //cerr<<(tmpa->s or tmpb->s or k)<<"
"<<tmpa->s<<" "<<tmpb->s<<endl;
  }
  return v;
}
int main() {
  chain<char> a,b,c;
  a.insert('a');
  a.insert('b');
  a.insert('c');
  b.insert('1');
  b.insert('2');
  b.insert('3');
  b.insert('4');
  c=merge(a,b);
  c.show();

}
```

6. Do Exercise 4 of Section 4.3 for the case of circularly linked lists.

```
#include <iostream>
using namespace std;
template<class T>
struct node {
    T value;
    node *right;
    bool s;
};
template<class T>
class chain {
private:
```

```
template<class T>
void chain<T>::insert(T val) {
  node<T> *temp = new node<T>();
  temp->value = val;
  temp->right = nullptr;
  temp->s=1;
  if (first == nullptr) {
    temp->s=false;
    first = temp;
    last = temp;
  } else {
    node<T> *last;
    node<T> *first;

public:
  chain() : first(nullptr), last(nullptr) {}
  void insert(T val);
  T top(){return first->value;}
  node<T>* First() const { return first; }
  void show(){
    node<T>*tmp =first;
    int  k=1;
    while(tmp->s or k){
      cout<<tmp->value<<" ";
      tmp=tmp->right;
      k=0;
    }
  }
  void copyToArray(T* arr, size_t maxSize) {
    node<T>* current = first;
    size_t index = 0;
    int k=1;
    while ((current != nullptr && index <
maxSize)or k) {
      arr[index++] = current->value;
      current = current->right;
      k=0;
    }
  }
};
```

```
    last->right = temp;
    last = temp;
    last->right=first;
  }
}
int main() {
  chain<char> a;
  a.insert('a');
  a.insert('b');
  a.insert('c');
  char arr[3];
  a.copyToArray(arr, 3);

  for (int i = 0; i < 3; i++) {
    cout << arr[i] << " ";

  }
  return 0;
}
```

7.Write a C++ Function to evaluate a polynomial at the point x…

```
#include<iostream>
using namespace std;
class Polynomial;
class Term{
    friend class Polynomial;
public:
    double coef;
    int exp;
    Term *link;
public:
    Term(double c, int e){
        coef = c;
        exp = e;
        link = nullptr;
    }
};
class Polynomial
{
public:
    Polynomial() : head(nullptr) {}
    double getPolyValue(double);
    Term *head;
};
```

```
double Polynomial::getPolyValue(double x){
    // Return 0 if polynomial is empty
    if (head == nullptr){
        cout << "Polynomial is empty." << endl;
        return 0.0;
    }
    double value = 0.0;
    Term *current = head;
    do{
        value += current->coef * pow(x,
current->exp);
        current = current->link;
    } while (current != head);
    return value;
}
int main(){
    Polynomial poly;
    // Header node
    poly.head = new Term(1.0, 4);
    poly.head->link = new Term(2.0, 3);
    poly.head->link->link = poly.head;
    double x = 2.5;
    double value = poly.getPolyValue(x);
    cout << "Value of the polynomial at x = " << x
<< " is: " << value << endl;
    return 0;
}
```

8.Devise a linked representation for a list in which…

```
#include <iostream>
using namespace std;
template <class T>
class DequeNode{
public:
    T value;
    DequeNode<T> *next;
    DequeNode<T> *prev;
    DequeNode(const T &data) : value(data),
next(nullptr), prev(nullptr) {}
};
template <class T>
class Deque{
private:
    DequeNode<T> *front;
    DequeNode<T> *rear;
public:
    Deque() : front(nullptr), rear(nullptr) {}
    bool isEmpty() const{
        return front == nullptr;
    }
    void insertFront(const T &value){
        DequeNode<T> *newNode = new
DequeNode<T>(value);
        if (isEmpty()){
            front = rear = newNode;
        }
        else{
            newNode->next = front;
            front->prev = newNode;
            front = newNode;
        }
    }
    // Push a value at the end
    void insertRear(const T &value){
        DequeNode<T> *newNode = new
DequeNode<T>(value);
        if (isEmpty()){
            front = rear = newNode;
        }else{
            rear->next = newNode;
            newNode->prev = rear;
            rear = newNode;
        }
    }
void deleteFront(){
        if (isEmpty()){
            cout << "Deque is empty. No elements to
delete." << endl;
            return;
        }
            DequeNode<T> *temp = front;
        if (front == rear){
            front = rear = nullptr;
        }else{
            front = front->next;
            front->prev = nullptr;
        }
        delete temp;
    }
    // Delete the last value
    void deleteRear(){
        if (isEmpty()){
            cout << "Deque is empty. No elements to
delete." << endl;
            return;
        }
        DequeNode<T> *temp = rear;
        if (front == rear){
            front = rear = nullptr;
        }else{
            rear = rear->prev;
            rear->next = nullptr;
        }
        delete temp;
    }
    void displayList() const{
        if (isEmpty()){
            cout << "Deque is empty." << endl;
            return;
        }
        DequeNode<T> *current = front;
        while (current != nullptr){
            cout << current->value << " ";
            current = current->next;
        }
        cout << endl;
    }
};
int main()
{
    Deque<int> deque;
    deque.insertFront(2);
    deque.insertFront(1);
    deque.insertRear(3);
    deque.insertRear(4);
    deque.displayList(); // Output: 1 2 3 4
    deque.deleteFront();
    deque.deleteRear();
    deque.displayList(); // Output: 2 3
    return 0;
}
```