#### HW4 電資二賴庭岳 4112064228

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

```
#include<iostream>
                                                     void chain::insert(int val){
using namespace std;
                                                       node *temp = new node();
struct node
                                                       temp->value = val;
                                                       temp->right = nullptr;
                                                       if (first == nullptr) {
  int value:
  node *right=0;
                                                          first = temp;
                                                          last = temp;
class chain
                                                       } else {
                                                          last->right = temp;
                                                          last = temp;
private:
  node *last;
  node *first;
                                                       return;
public:
  chain(): last(0), first(0) {}
                                                     chain& merge(chain& a,chain& b){
  ~chain(){
                                                       node* tmpa=a.First();
                                                       node* tmpb=b.First():
    node* current = first;
     while (current != nullptr) {
                                                       static chain c:
                                                       while(tmpa!=NULL or tmpb!=NULL){
       node* temp = current;
       current = current->right;
                                                          if(tmpa!=NULL){
       delete temp;
                                                            c.insert(tmpa->value);
     }
                                                            tmpa=tmpa->right;
  }
  void insert(int val);
                                                          if(tmpb!=NULL){
                                                            c.insert(tmpb->value):
  int top(){
     return last->value;
                                                            tmpb=tmpb->right;
  void show_all(){
     node *tmp=first;
                                                       return c;
     while(tmp!=NULL){
       cout<<tmp->value<<" ":
                                                     int main(){
       tmp=tmp->right;
                                                       chain a,b,c;
                                                       a.insert(1);a.insert(3);a.insert(5);
                                                       //a.show all();
     cout<<endl:
                                                       b.insert(2);b.insert(4);b.insert(6);
  //void merge(const chain& b);
                                                       c=merge(a,b);
  node* First() const {
                                                       c.show_all();
     return first;
                                                       //cout<="123";
  };
```

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

```
#include<iostream>
                                                   chain& merge(chain& a,chain& b){
using namespace std;
                                                      node* tmpa=a.First();
                                                      node* tmpb=b.First();
struct node{
  int value;
                                                      static chain c:
  node *right=0;
                                                      while(tmpa!=NULL or tmpb!=NULL){
                                                        if(tmpa==NULL){
class chain{
                                                           c.insert(tmpb->value);
private:
                                                           tmpb=tmpb->right;
  node *last;
                                                           continue;
  node *first;
                                                        if(tmpb==NULL){
public:
  chain(): last(0), first(0) {}
                                                           c.insert(tmpa->value);
```

```
~chain(){
                                                             tmpa=tmpa->right;
    node* current = first;
                                                             continue:
     while (current != nullptr) {
       node* temp = current;
                                                          if(tmpa->value > tmpb->value ){
       current = current->right;
                                                             c.insert(tmpb->value);
       delete temp;
                                                             tmpb=tmpb->right;
                                                             continue;
     }
  }
  void insert(int val);
                                                          if(tmpb->value > tmpa->value ){
  int top(){return last->value;}
                                                             c.insert(tmpa->value);
  void show all(){
                                                             tmpa=tmpa->right;
     node *tmp=first;
                                                             continue;
     while(tmp!=NULL){
                                                          }
       cout<<tmp->value<<" ";
                                                       }
       tmp=tmp->right;
                                                       return c;
                                                     int main(){
     cout<<endl;
                                                       chain a,b,c;
                                                       a.insert(3);
  node* First() const {return first;};
                                                        a.insert(4);
void chain::insert(int val){
                                                        a.insert(5);
  node *temp = new node();
                                                        //a.show all();
  temp->value = val;
                                                        b.insert(1);
  temp->right = nullptr;
                                                        b.insert(7);
  if (first == nullptr) {
                                                        b.insert(8);
     first = temp;
                                                        c=merge(b,a);
     last = temp;
                                                        c.show all();
                                                        //cout<<"123";
  } else {
     last->right = temp;
     last = temp;
  return;
```

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

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

```
if (first == nullptr) {
chain(): last(NULL), first(NULL) {}
                                                            first = temp:
  ~chain(){
                                                            last = temp;
    node* current = first;
                                                         } else {
    while (current != nullptr) {
                                                            last->right = temp;
       node* temp = current;
                                                            last = temp;
       current = current->right;
       delete 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>
                                                      template<class T>
using namespace std;
                                                      void chain<T>::insert(T val) {
template<class T>
                                                        node<T> *temp = new node<T>();
struct node {
                                                        temp->value = val;
                                                        temp->right = nullptr;
  T value;
  node *right;
                                                        if (first == nullptr) {
                                                           first = temp;
template<class T>
                                                           last = temp;
class chain {
                                                        } else {
                                                           last->right = temp;
private:
  node<T> *last;
                                                           last = temp;
  node<T> *first;
                                                      int main() {
public:
  chain() : first(nullptr), last(nullptr) {}
                                                        chain<char> a;
                                                        a.insert('a');
  void insert(T val);
  node<T>* First() const { return first; }
                                                        a.insert('1');
                                                        a.insert('b');
  void copyToArray(T* arr, size t maxSize) {
     node<T>* current = first;
                                                        char arr[3];
     size t index = 0:
                                                        a.copyToArray(arr, 3);
     while (current != nullptr && index <
maxSize) {
                                                        for (int i = 0; i < 3; i++) {
        arr[index++] = current->value;
                                                           cout << arr[i] << " ";
        current = current->right;
     }
                                                        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>
chain<T>& merge(const chain<T> a,const chain<T> b){
node<T>* tmpa=a.First();
```

```
T value:
                                                       node<T>* tmpb=b.First();
  node *right;
                                                       static chain<T> v:
  bool s:
                                                      int k=1:
                                                      while(tmpa->s or tmpb->s or k){
                                                        if(tmpa->s or k){
template<class T>
class chain {
                                                         v.insert(tmpa->value);
                                                         tmpa=tmpa->right;
private:
  node<T> *last;
  node<T> *first;
                                                        if(tmpb->s or k){
                                                         v.insert(tmpb->value);
                                                         tmpb=tmpb->right;
 chain() : first(nullptr), last(nullptr) {}
                                                       }
 void insert(T val);
 T top(){return first->value;}
                                                       //cerr<<(tmpa->s or tmpb->s or k)<<"
 node<T>* First() const { return first; }
                                                     "<<tmpa->s<<" "<<tmpb->s<<endl;
 void show(){
  node<T>*tmp =first;
                                                      return v;
  int k=1;
  while(tmp->s or k){
                                                     int main() {
    cout<<tmp->value<<" ";
                                                      chain<char> a,b,c;
    tmp=tmp->right;
                                                      a.insert('a');
    k=0;
                                                      a.insert('b');
  }
                                                      a.insert('c');
                                                      b.insert('1');
 }
                                                      b.insert('2');
template<class T>
                                                      b.insert('3');
void chain<T>::insert(T val) {
                                                      b.insert('4');
 node<T> *temp = new node<T>();
                                                      c=merge(a,b);
 temp->value = val;
                                                      c.show();
 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;
}
```

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

```
#include <iostream>
                                                   template<class T>
using namespace std;
                                                   void chain<T>::insert(T val) {
template<class T>
                                                    node<T> *temp = new node<T>();
struct node {
                                                    temp->value = val;
  T value;
                                                    temp->right = nullptr;
  node *right;
                                                     temp->s=1;
  bool s:
                                                    if (first == nullptr) {
                                                      temp->s=false;
template<class T>
                                                      first = temp;
                                                      last = temp;
class chain {
private:
                                                    } else {
```

```
node<T> *last;
                                                         last->right = temp;
  node<T> *first:
                                                         last = temp:
                                                         last->right=first;
public:
 chain() : first(nullptr), last(nullptr) {}
                                                      int main() {
 void insert(T val);
 T top(){return first->value;}
                                                        chain<char> a;
 node<T>* First() const { return first; }
                                                        a.insert('a');
                                                        a.insert('b');
 void show(){
  node<T>*tmp =first;
                                                        a.insert('c');
  int k=1;
                                                        char arr[3];
  while(tmp->s or k){
                                                        a.copyToArray(arr, 3);
    cout<<tmp->value<<" ":
                                                        for (int i = 0; i < 3; i++) {
    tmp=tmp->right;
    k=0;
                                                         cout << arr[i] << " ";
  }
 void copyToArray(T* arr, size t maxSize) {
                                                        return 0;
  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:
 }
};
```

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

```
#include<iostream>
                                                    double Polynomial::getPolyValue(double x){
                                                      // Return 0 if polynomial is empty
using namespace std;
                                                      if (head == nullptr){
class Polynomial;
                                                         cout << "Polynomial is empty." << endl;
class Term{
  friend class Polynomial;
                                                         return 0.0;
public:
  double coef;
                                                      double value = 0.0;
  int exp;
                                                      Term *current = head;
  Term *link;
                                                         value += current->coef * pow(x,
public:
  Term(double c, int e){
                                                    current->exp);
     coef = c;
                                                         current = current->link;
     exp = e:
                                                      } while (current != head);
     link = nullptr;
                                                      return value;
                                                    int main(){
class Polynomial
                                                      Polynomial poly;
                                                      // Header node
                                                      poly.head = new Term(1.0, 4);
public:
  Polynomial(): head(nullptr) {}
                                                      poly.head->link = new Term(2.0. 3):
  double getPolyValue(double);
                                                      poly.head->link->link = poly.head;
  Term *head;
                                                      double x = 2.5:
                                                      double value = poly.getPolyValue(x);
```

```
cout << "Value of the polynomial at x = " << x
<< " is: " << value << endl;
  return 0;
}</pre>
```

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

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