9-2

结点类至少包括数据域和指针域的内容，函数成员中应该含有对数据和指针进行初始化的函数，以及插入和删除结点的函数。

单链表每个结点中只有一个指向后继结点的指针；双向链表中有两个用于连接其他结点的指针，一个指向前趋结点，一个指向后继结点。

9-3

链表类中声明的size是int型，所以最大是占4个字节，2147483647个元素。

9-4

双向链表中有两个指向其他结点的指针。

```
template<class T>
class Dnode{
private:
  Dnode<T>* prev;
  Dnode<T>* next;
public:
  T data;
  Dnode(const T &data, Dnode<T>* prev = 0, Dnode<T>* next = 0);  //构造
  void insertafter(Dnode<T>* p);   //结点后插入同类结点p
  void insertfront(Dnode<T>* p);   //结点前插入同类结点p
  Dnode<T>* deleteafter();     //删除后继结点，返回其地址
  Dnode<T>* deletefront();     //删除前趋结点，返回其地址
  Dnode<T>* nextnode();      //获取后继结点地址
  Dnode<T>* prevnode();      //获取前趋结点地址
  const Dnode<T>* nextnode() const;    //获取后继结点地址
  const Dnode<T>* prevnode() const;    //获取前趋结点地址
};

template<class T>
Dnode<T>::Dnode(const T& data, Dnode<T>* next/ * = 0 * /, Dnode<T>* prev/ * = 0 * /
):data(data), next(next),prev(prev){}

template<class T>
void Dnode<T>::insertafter(Dnode<T>* p){
  Dnode<T>* temptr=next;
  temptr->prev=p;
  p->next=next;
  next=p;
  p->prev=this;
}

template<class T>
void Dnode<T>::insertfront(Dnode<T>* p){
  Dnode<T>* temptr=prev;
  temptr->next=p;
  p->prev=prev;
  prev=p;
  p->next=this;
```

```
}

template<class T>
Dnode<T>* Dnode<T>::deleteafter(){
  Dnode<T>* temptr=next;
  if(next==0){
    return 0;
  }
  next=temptr->next;
  temptr->next->prev=this;
  return temptr;
}

template<class T>
Dnode<T>* Dnode<T>::deletefront(){
  Dnode<T>* temptr=prev;
  if(prev==0){
    return 0;
  }
  prev=temptr->prev;
  temptr->prev->next=this;
  return temptr;
}

template<class T>
Dnode<T>* Dnode<T>::nextnode(){
  return next;
}

template<class T>
Dnode<T>* Dnode<T>::prevnode(){
  return prev;
}

template<class T>
const Dnode<T>* Dnode<T>::nextnode() const{
  return next;
}

template<class T>
const Dnode<T>* Dnode<T>::prevnode() const{
  return prev;
}
```

9-5

```
#include <iostream>
#include "LinkedList.h"
using namespace std;
int main(){
  LinkedList<int>a;
  LinkedList<int>b;
  for(int i=0;i<5;i++){
    int item;
    cin>>item;
    a.insertRear(item);
  }
  for(int i=0;i<5;i++){
    int item;
```

```
    cin>>item;
    b.insertFront(item);
  }
  b.reset();
  while(!b.endOfList()){
    a.insertRear(b.data);
    b.next();
  }
}
```

9-6

```
#include <iostream>
#include "Node.h"
using namespace std;

template<class T>
class Orderlist{
public:           //在linkedlist类的基础上,同时已有结点类
  void insert(const T &item);   //有序递增插入
};

template<class T>
void Orderlist::insert(const T &item){
  currptr=front;  //当前遍历指针指向表头结点
  prevptr=front;
  while(currptr.data<item){
    prevptr=currptr;
    currptr=currptr->next();
  }
  Node<T>tem(item);     //创建结点
  prevptr->next=tem;    //在当前结点前插入
  tem.next=currptr;
}

int main(){
  Orderlist<int>a;
  OrderList<int>b;
  for(int i=0;i<5;i++){
    int item;
    cin>>item;
    a.insert(item);
  }
  for(int i=0;i<5;i++){
    int item;
    cin>>item;
    b.insert(item);
  }
  b.reset();
  while(!b.endOfList()){
    a.insert(b.data);
    b.next();
  }
}
```

9-7

栈是一种特殊的线性群体。

对栈中元素的访问是受限制的，压入栈和弹出栈都只能在栈顶进行。

9-8

队列是一种特殊的线性群体。

队列中元素的访问受限制，只能在队尾删除元素（出队），在队头添加元素（入队）

9-9

插入排序：每一步将一个待排序的元素按其关键字值的大小插入到已排序序列的适当位置上，知道待排序元素插入完为止。

9-10

```
#include <iostream>
using namespace std;
int main(){
    int a[]={1,3,5,7,9,11,13,15,17,19,2,4,6,8,10,12,14,16,18,20};
    for(int i=1;i<20;i++){
        int temp=a[i];
        int j=i;
        while(j>0&&a[j-1]>temp){
            a[j]=a[j-1];
            j--;
        }
        a[j]=temp;
        for(int i=0;i<20;i++){
            cout<<a[i]<<" ";
        }
        cout<<endl;
    }
}
```

9-11

选择排序：每次从待排序序列中选择一个关键字值最小的元素（升序排序时，降序排序则选最大），顺序排在已排序序列的最后（与未排序序列的第一个元素进行交换）