

9-2

节点类中应该至少包含下一节点的地址（指针）、本节点的数据两个成员。

单向链表的节点只存储下一节点的指针，双向链表的既存储下一节点又存储上一节点的指针。

9-3

链表一般为动态申请内存，上限为可用内存大小（堆内存）。

9-4

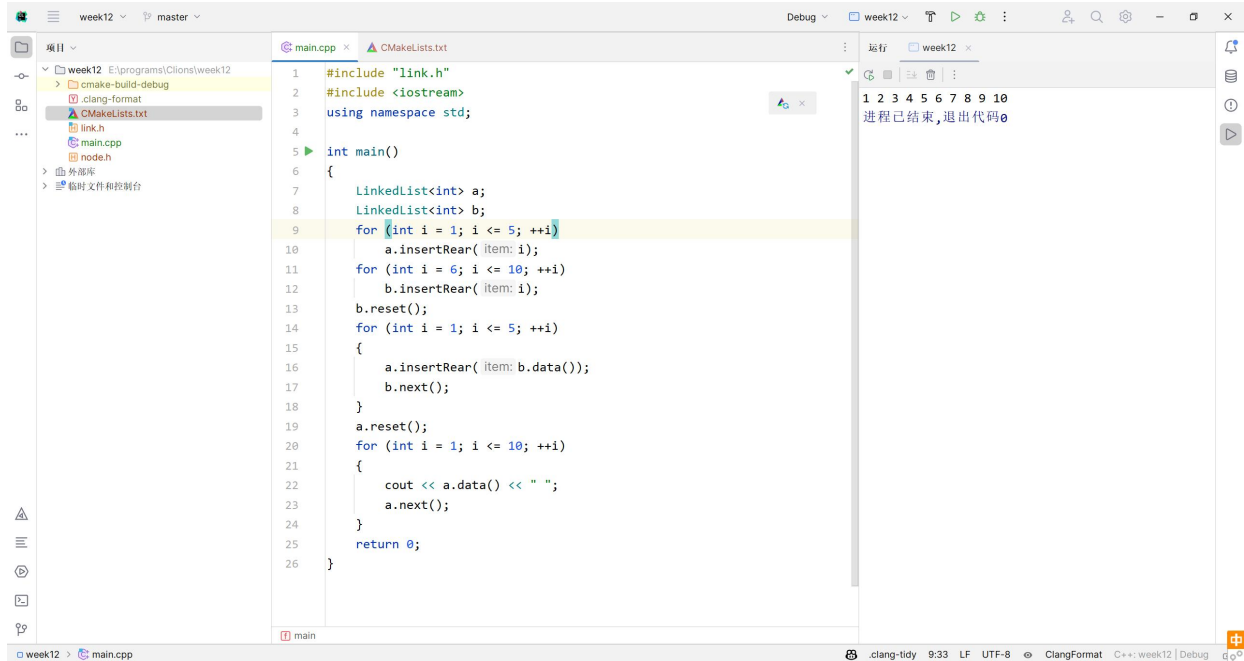
双向链表中节点会存储前后两个节点的指针。

```
1  template<class T>
2  class Dnode
3  {
4  public:
5      T data;
6      Dnode<T> *prior, *next;
7      Dnode(const T &x, Dnode<T> *p = NULL, Dnode<T> *n = NULL) : data(x),
      prior(p), next(n) {}
8  };
```

9-5

```
1  #include "link.h"
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      LinkedList<int> a;
8      LinkedList<int> b;
9      for (int i = 1; i <= 5; ++i)
10         a.insertRear(i);
11     for (int i = 6; i <= 10; ++i)
12         b.insertRear(i);
13     b.reset();
14     for (int i = 1; i <= 5; ++i)
15     {
16         a.insertRear(b.data());
17         b.next();
18     }
19     a.reset();
20     for (int i = 1; i <= 10; ++i)
21     {
22         cout << a.data() << " ";
23         a.next();
24     }
```

```
25     return 0;
26 }
```



9-6

```
1 template <class T>
2 class OerderList : public LinkedList<T>
3 {
4 public:
5     void insert(const T &x)
6     {
7         LinkedList<T>::reset();
8         while (!LinkedList<T>::endOfList() && LinkedList<T>::data() < x)
9             LinkedList<T>::next();
10        LinkedList<T>::insertAt(x);
11    }
12 };
```

9-7

栈是一种数据结构，是一种线性表，只支持入栈和出栈操作，符合的是先入后出原则。

栈是一种特殊的线性表，只能在表的一端进行插入和删除操作，这一端称为栈顶，另一端称为栈底。

9-8

与栈类似，队列也是一种线性表的数据结构，支持进队和出队两个操作，符合先入先出原则。

队列是一种特殊的线性表，它只允许在表的前端 (**front**) 进行删除操作，而在表的后端 (**rear**) 进行插入操作

9-9

插入排序是一种时间复杂度为 $O(n^2)$ 的排序算法，当前元素前面的序列已经排序完成，对当前数据去找它在前面有序序列中的位置并插入进去。

插入排序是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

插入排序在实现上，通常采用**in-place**排序（即只需用到 $O(1)$ 的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

最坏情况下，每次插入都要交换，比较次数为 $1+2+3+\dots+n-1$ ，时间复杂度为 $O(n^2)$ 。

最好情况下，每次插入都不用交换，时间复杂度为 $O(n)$ 。

插入排序的平均时间复杂度为 $O(n^2)$ 。

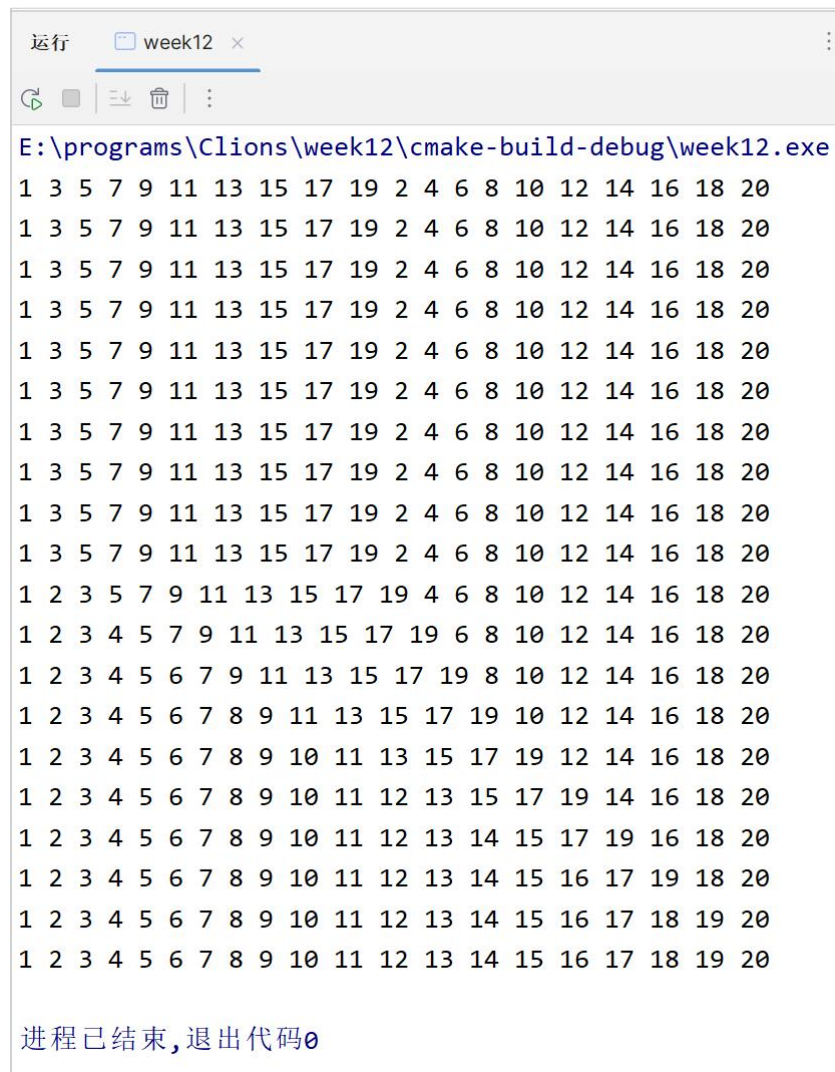
插入排序的空间复杂度为 $O(1)$ 。

插入排序是稳定的排序算法。

9-10

```
1  #include <iostream>
2  using namespace std;
3
4  //输出数组内容
5  void PrintArray(int data[],int n)
6  {
7      for(int i=0;i<n;i++)
8      {
9          cout<<data[i]<<" ";
10     }
11     cout<<endl;
12 }
13 //直接插入排序
14 void InsertSort(int data[],int n)
15 {
16     for(int i=1;i<n;++i)
17     {
18         int j=i-1;
19         while(j>=0&&data[j]>data[i])
20             --j;
21         int tmp=data[i];
22         for(int k=i;k>j;--k)
23             data[k]=data[k-1];
24         data[j+1]=tmp;
25         PrintArray(data,n);
26     }
27 }
28
29 int main()
30 {
31     int data[]={1,3,5,7,9,11,13,15,17,19,2,4,6,8,10,12,14,16,18,20};
32     int n=sizeof(data)/sizeof(int);
33     PrintArray(data,n);
34     InsertSort(data,n);
```

```
35     return 0;
36 }
37
```



```
运行 week12 x
E:\programs\Clions\week12\cmake-build-debug\week12.exe
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19 2 4 6 8 10 12 14 16 18 20
1 2 3 5 7 9 11 13 15 17 19 4 6 8 10 12 14 16 18 20
1 2 3 4 5 7 9 11 13 15 17 19 6 8 10 12 14 16 18 20
1 2 3 4 5 6 7 9 11 13 15 17 19 8 10 12 14 16 18 20
1 2 3 4 5 6 7 8 9 11 13 15 17 19 10 12 14 16 18 20
1 2 3 4 5 6 7 8 9 10 11 13 15 17 19 12 14 16 18 20
1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 19 14 16 18 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 19 16 18 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 18 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

进程已结束,退出代码0
```

9-11

选择排序是一种时间复杂度也为 $O(n^2)$ 的排序算法，它也是让当前元素前面的序列已经排序完成，对剩下的序列每次遍历找极值（最大或最小）然后将极值与排序完成的部分序列的后一个元素交换， n 次交换后排序完成。