第一章

链　　表

何谓链表

我们知道，一般用数组存放一组数据时，必须事先定义固定的长度（即元素个数）。这在某些问题的解决中，并不是特别的适用。例如，记录不同班级的学生数据时，由于各班人数不同，会出现开辟过大的数组导致内存浪费、开辟过小的数组导致数组元素不够用的情况。而链表可以根据需要动态开辟内存单元，是一种常见的重要数据结构。图 1.1 所示为最简单的一种链表结构。

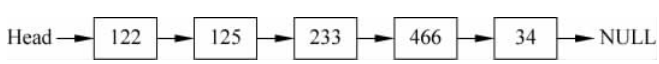

图　1.1

链表如同铁链一样，一环扣一环，中间是不能断开的。打个通俗的比方：幼儿园老师带领小朋友出来散步，老师牵着第一个小朋友的手，第一个小朋友的手牵着第二个小朋友的手……这就是一个“链”，最后一个小朋友的手是空的。

老师即“头指针”变量，图 1.1 中以 Head 表示，它存放一个地址。链表中每一个元素称为“结点”，每个结点都应该包括两部分：一为实际元素值，一为下一结点的地址。

最后一个元素不指向其他元素，它被称为“表尾”，以“NULL”表示，“NULL”在 C++语言里指向“空地址”。

很显然，这种链表的数据结构，必须要用指针变量才能实现。

简单静态链表

下面的代码是一个简单的静态链表，它由 3 个学生的数据（学号，成绩）的结点组成。请考虑：(1)head 的作用？(2)p 的作用？

```
1  //简单静态链表
2  #include <iostream>
3  using namespace std;
4
5  struct  student
6  {
7    long num;
8    float score;
9    struct student *next;                    //该指针指向 student 类型的结构体
10 };                                         //注意必须有分号
11
12 int main()
13 {
14   struct student a,b,c, *head, *p;
15   a.num=34341;   a.score=81.5;            //赋值
16   b.num=34343;   b.score=97;
17   c.num=34344;   c.score=82;
18   head=&a;                                //将 a 的地址给 head
19   a.next=&b;
20   b.next=&c;
21   c.next=NULL;
22   p=head;
23   do                                      //输出记录
24   {
25       cout<<p->num<<" "<<p->score<<endl;
26       p=p->next;
27   }while(p!=NULL);
28   getchar();
29 }
```

处理动态链表的函数

1. malloc

函数原型为

```
void *malloc(unsigned int size);
```

作用是在内存的动态存储区中分配一个长度为 size 的连续空间。此函数返回的是一个指向分配域起始地址的指针，如果此函数未能成功地执行(如内存空间不足)，则返回空指针(NULL)。

2. calloc

函数原型为

```
void *calloc(unsigned n,unsigned size);
```

作用是在内存的动态存储区中分配 n 个长度为 size 的连续空间。函数返回一个指向分配域起始地址的指针；如果分配不成功，则返回 NULL。

3. free()

函数原型为

```
void free(void * p);
```

作用是释放由 p 指向的内存区，使这部分内存区能被其他变量使用。

动态链表的准备工作

一个完善的动态链表程序应该具有以下基本功能：建立链表、插入结点、删除结点、打印链表、释放链表等。扩展的动态链表程序还可能有获得链表长度、获得当前结点、查找结点位置、连接两个链表、比较两个链表等功能。下面将逐个实现其功能代码。

为了程序的易读性和可扩展性，有时需要在程序开头先进行预定义处理。请务必领会下面代码的用意，否则将影响对以后代码的理解。

```
#include <stdlib.h>
#include <stdio.h>
typedef int ElemType;                    //以 ElemType 代表 int 型数据
typedef struct List * link;              //以 link 代表链表指针
typedef struct List Lnode;               //以 Lnode 代表链表结点

struct List
{
   ElemType data;                        //此处仅以一个整型变量为例
   struct List * next;
};
```

主函数的建立：下面的主函数只是一个简单调用各功能子函数的示范例子，读者可自行修改和添加代码以完成更复杂的任务。请根据主函数的代码考虑各功能子函数的原型应如何建立。

```
int main()
{
   int l;link head1;link head2;
   head1=create(head1);                                  //建立链表 1
   head2=create(head2);                                  //建立链表 2
   connect(head1,head2);                                 //连接两个链表
   head1=insert(head1,888,5);                            //在位置为 5 处插入元素 888
   head1=del(head1,3);                                   //删除一个结点
   display(head1);                                       //打印链表
   printf("\n  lenth is %d\n", lenth(head1));            //打印链表长度
   printf("\n  get is %d\n", get(head1,3));              //获得当前结点值
   printf("\n  locate 12 is %d", locate(head1,12));      //查找元素 12 所在的位置
   head=setnull(head);                                   //释放链表
}
```

链表的建立

由主函数调用 create()函数的方式可知,该函数应该返回一个结点的指针,输入的参数也应该是一个结点指针。参考代码如下。

```
1  link create(link Head)
2  {
3     ElemType newData;
4     link NewPoint;
5     /* 先建立一个结点 */
6     Head=(link)malloc(sizeof(Lnode));
7     printf("please input number: \n");
8     scanf("%d",&newData);
9     Head->data=newData;                    //结点赋值
10    Head->next=NULL;                       //结点指向空地址
11
12    while(1)                               //继续添加结点
13    {
14       NewPoint=(link)malloc(sizeof(Lnode));   //开辟一个结点空间
15       if(NewPoint==NULL)                  //如果开辟空间失败,则返回
16          break;                           //此判断语句在某些类型的竞赛中用处不大,可忽略
17       printf("please input number: input '-1' means exit\n");
18       scanf("%d",&newData);
19       if (newData==-1)                    //输入-1 则添加结点结束并返回 head
20          return Head;
21       NewPoint->data=newData;
22       NewPoint->next=Head;
23       Head=NewPoint;
24    }
25    return Head;
26 }
```

链表的显示

该子函数无返回值,输入参数为链表的头指针,请考虑:指针 p 的作用是什么?如果不用指针 p,直接用 Head 这个指针会出现什么后果?

```
1  void display(link Head)
2  {
3     link p;p=Head;
4     if(p==NULL)
5        printf("\nList is empty\n");
6     else
7        do
8        {
9           printf("%d ",p->data);
```

```
        p=p->next;
    }while(p!=NULL);
}
```

结点的插入

```
link insert(link Head,ElemType x,int i)
{
  link NewPoint,p=Head;
  int j=1;
  NewPoint=(link)malloc(sizeof(Lnode));
  NewPoint->data=x;
  if(i==1)      //如果插入位置为第一个结点时
  {   NewPoint->next=Head;   Head=NewPoint; }
  else
  {
    while(j<i-1 && p->next!=NULL)
    {      p=p->next;   j++;   }
    if(j==i-1)
    {
      NewPoint->next=p->next;
      p->next=NewPoint;
    }
    else   printf("insert is Failure,i is not right!");
  }
  return Head;
}
```

结点的删除

例如下面这个链表：

(A)→(B)→(C)→(D)→(E)→…→NULL

因为链表中唯一能够找到元素的办法是通过它上一个元素的指针，所以如果我们将某个元素直接删除，这个元素所指向的元素和它之后的所有元素都没有办法再找到了，为了解决这个问题，一般采取这样的办法：记录下要删除的元素之前的那个元素，在删除元素之前，把这个元素的指针指向要删除的那个元素指针指向的元素，比如说现在要删除这个链表的元素B，先找到它之前的元素 A，在删除 B 之前将 A 的指针指向 C，然后再删除 B，这样在删除 B 之前，就已经把 B 从链表里面剔了出来，如图 1.2 所示。

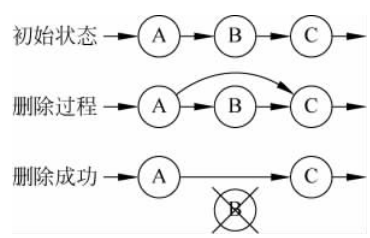

图　1.2

```
link del(link Head,int i)
{
```

```
3     int j=1; link p,t; p=Head;
4     if(i==1)
5     {    p=p->next;    free(Head);    Head=p; }
6     else
7     {
8        while(j<i-1 && p->next !=NULL)
9        {   p=p->next;        j++;     }
10       if(p->next!=NULL && j==i-1)
11          {   t=p->next;        p->next=t->next;      }
12       if(t!=NULL)   free(t);
13    }
14    return Head;
15 }
```

上面的例子中操作符 free 用来删除一个变量，即在内存中释放某个变量所对应的地址，使得之后系统可以利用这块内存做别的事情，这样比较节省内存空间。我们应养成及时释放无用的内存空间的好习惯。

获得结点元素值

```
1  ElemType get(link Head,int i)
2  {
3     int j=1;
4     link p; p=Head;
5     while(j<i && p!=NULL)
6     {      p=p->next;      j++;   }
7     if(p!=NULL)
8           return(p->data);
9     else
10          printf("data is error!");
11          return -1;
12 }
```

查找结点元素 X 的位置

```
1  int locate(link Head,ElemType x)
2  {
3     int n=0;link p;p=Head;
4     while(p!=NULL && p->data !=x)
5     {   p=p->next;       n++;   }
6     if(p==NULL)
7           return -1;
8     else
9           return n+1;
10 }
```

返回链表的长度

```
1  int lenth(link Head)
2  { int len=0;  link p;  p=Head;
3    while(p!=NULL)
4    {     len++;     p=p->next;  }
5    return len;
6  }
```

连接两个链表

```
1  link connect(link Head1,link Head2)
2  {
3     link p; p=Head1;
4     while(p->next !=NULL)
5     { p=p->next;  }
6     p->next=Head2;
7     return Head1;
8  }
```

比较两个链表是否相同

```
1  int compare(link Head1,link Head2)
2  {
3     link p1,p2; p1=Head1; p2=Head2;
4     while(1)
5     {
6        if((p1->next==NULL) &&(p2->next==NULL))
7              return 1;
8        if(p1->data!=p2->data)
9              return 0;
10       else
11       {     p1=p1->next;     p2=p2->next;   }
12    }
13 }
```

释放链表

```
1  link setnull(link Head)
2  {
3     link p;p=Head;
4     while(p!=NULL)
5     {
6        p=p->next;
```

```
        free(Head);
        Head=p;
    }
    return Head;
}
```

完整的链表程序

参考代码如下。

```
#include <stdlib.h>
#include <stdio.h>
typedef int ElemType;
typedef struct List *link;
typedef struct List Lnode;

struct List
{
   ElemType data;
   struct List *next;
};

link setnull(link Head)
{
   link p;p=Head;
   while(p!=NULL)
   {
       p=p->next;
       free(Head);
       Head=p;
   }
   return Head;
}

link insert(link Head,ElemType x,int i)
{
   link NewPoint,p=Head;
   int j=1;
   NewPoint=(link)malloc(sizeof(Lnode));
   NewPoint->data=x;
   if(i==1)
   {
      NewPoint->next=Head;
      Head=NewPoint;
   }
   else
   {
      while(j<i-1 && p->next!=NULL)
      {
```

```
40          p=p->next;
41          j++;
42        }
43        if(j==i-1)
44        {
45          NewPoint->next=p->next;
46          p->next=NewPoint;
47        }
48        else   printf("insert is Failure,i is not right!");
49      }
50      return Head;
51  }
52
53  link create(link Head)
54  {
55      ElemType newData;
56      link NewPoint;
57
58      Head=(link)malloc(sizeof(Lnode));
59      printf("please input number: \n");
60      scanf("%d",&newData);
61      Head->data=newData;
62      Head->next=NULL;
63
64      while(1)
65      {
66        NewPoint=(link)malloc(sizeof(Lnode));
67        if(NewPoint==NULL)
68            break;
69        printf("please input number: input '-1' means exit\n");
70        scanf("%d",&newData);
71        if (newData==-1)
72          return Head;
73        NewPoint->data=newData;
74        NewPoint->next=Head;
75        Head=NewPoint;
76      }
77      return Head;
78  }
79
80  int lenth(link Head)
81  {
82      int len=0;
83      link p;
84      p=Head;
85      while(p!=NULL)
86      {
87        len++;
88        p=p->next;
89      }
```

```
90      return len;
91  }
92
93  ElemType get(link Head,int i)
94  {
95      int j=1;
96      link p; p=Head;
97      while(j<i && p!=NULL)
98      {
99          p=p->next;
100         j++;
101     }
102     if(p!=NULL)
103         return(p->data);
104     else
105         printf("data is error!");
106     return -1;
107 }
108
109 int locate(link Head,ElemType x)
110 {
111     int n=0;link p;p=Head;
112     while(p!=NULL && p->data !=x)
113     {
114         p=p->next;
115         n++;
116     }
117     if(p==NULL)
118         return -1;
119     else
120         return n+1;
121 }
122
123 void display(link Head)
124 {
125     link p;p=Head;
126     if(p==NULL)
127         printf("\nList is empty\n");
128     else
129         do
130         {
131             printf("%d ",p->data);
132             p=p->next;
133         }while(p!=NULL);
134 }
135
136 link connect(link Head1,link Head2)
137 {
138     link p;
139     p=Head1;
```