作业 2 线性结构的存储结构与应用

一、线性表的顺序存储结构及功能实现

1.顺序存储结构的实现

顺序存储结构即用一段地址连续的存储单元依次存储线性表的数据元素。由于线性表中存储的是**同种数据类型**，且存储单元连续，因此可以使用**一维数组**来实现顺序存储结构。

定义一个结构体，内包含数组 data（用以存储数据）以及当前数组长度 length（存储当前数据量的大小）。对于之后的增删改查等功能，只需通过对表头的指针对相应位置进行操作即可。

```
typedef struct SeqList
{
    int data[MaxSize];
    int length;
}SeqList;
```

2.顺序存储结构的功能实现。

通过程序编写实现以下功能，

1) 数据信息初始化

通过文件读取，依次读入数据（文本中均为整型 int 数据类型）。每读入一个数据，当前数据数组长度随之发生改变。

```
//初始化
void initialize(SeqList *seqlist)
{
    seqlist->length = 0;
    ifstream infile;
    infile.open("data_test.txt");
    for(int i = 0;i < MaxSize; i++)
    {
        infile>>seqlist->data[i];
        if(seqlist->data[i] == -1)
            break;
        seqlist->length++;
    }
    infile.close();
    cout<<"The length of SeqList is :"<<seqlist->length<<endl;
}
```

2) 输出当前数据信息

自数组头依次输出数据，同时输出当前数组长度。

```
//打印数据
void Print(SeqList *seqlist)
{
    for(int i = 0; i < MaxSize; i++)
    {
        if(seqlist->data[i] == -1)
            break;
        cout<<seqlist->data[i]<<endl;
    }
    cout<<"The length of SeqList is :"<<seqlist->length<<endl;
}
```

3) 删除顺序表中第 k 个数据

首先检查输入是否为正且小于当前数组长度，随后找到目标节点，将在其之后的数据依次往前移动一位，同时数组长度-1，实现数据的单个删除。

```
//删除第k个数据
void Delete(SeqList *seqlist, int k)
{
    //排除非法输入
    if(k > seqlist->length || k < 0)
        cout<<"The wrong input!"<<endl;
    else
    {
        for(int i = k - 1; i < MaxSize; i++)
        {
            seqlist->data[i]=seqlist->data[i+1];
            if(seqlist->data[i] == -1)
                break;
        }
    //长度变化
        seqlist->length--;
        Print(seqlist);
    }
}
```

4) 查找某一数据信息

遍历匹配目标数据信息，输出此数据以及其位置。

```
//查找数据
void Search(SeqList *seqlist, int x)
{
    for(int i = 0; i < MaxSize; i++)
    {
        if(seqlist->data[i] == -1)
            break;
        if(seqlist->data[i] == x)
            cout<<"您查找的数据为"<<seqlist->data[i]<<",在顺序表中位置为"<<i+1<<endl;
    }
}
```

5) 修改数据信息

输入为将被修改数据的位置，由于顺序表，直接对该位置数据进行修改即可。

```
//修改数据
void Change(SeqList *seqlist, int x)
{
    if(x > seqlist->length || x < 0)
        cout<<"The wrong input"<<endl;
    else
    {
        cout<<"将被修改的数据为:"<<seqlist->data[x-1]<<endl;
        int m;
        cout<<"要将其修改为: "<<endl;
        cin>>m;
        seqlist->data[x-1] = m;
    }
}
```

6) 插入数据信息

由顺序表存储空间连续的特性，由后往前遍历，依次将数据后移一位，搜索

至目标位置后，输入插入数据即可实现数据的插入。

```
void Insert(SeqList *seqlist, int x)
{
    if(x > seqlist->length || x < 0)
        cout<<"The wrong input"<<endl;
    else if(seqlist->length == MaxSize)
        cout<<"当前顺序表空间已满"<<endl;
    else
    {
        int m;
        cout<<"插入的数据为："<<endl;
        cin>>m;
        for(int i = seqlist->length; i > x - 1; i--)
            seqlist->data[i]=seqlist->data[i-1];

        seqlist->data[x-1] = m;
        seqlist->length++;
        seqlist->data[seqlist->length] = -1;
    }
}
```

7) 排序

排序选择效率最高的插入排序。

```
//选择效率最高的插入排序
void sort(SeqList *seqlist)
{
    for(int i = 1;i < seqlist->length; i++)
    {
        int key = seqlist->data[i];
        int j = i - 1;
        while((j >= 0) && (key < seqlist->data[j]))
        {
            seqlist->data[j+1] = seqlist->data[j];
            j--;
        }
        seqlist->data[j+1] = key;
    }
    Print(seqlist);
}
```

8) 删除所有重复元素

对于排序好的序列，重复元素必相邻。因此从第一个数据依次往后遍历，比较当前位置与下一相邻位置的数据是否相等，若相等，删除当前位置数据。同时由于删除时后续数据依次往前移动，所以需要再次比较当前位置与下一相邻位置的数据是否相等。否则当出现两个以上重复元素时会出现删除不完全的情况。

```
//对于已排好序的线性表,删除重复元素
void Delete_repeat(SeqList *seqlist)
{
    sort(seqlist);
    for(int i = 0; i < seqlist->length; i++)
        if(seqlist->data[i] == seqlist->data[i+1])
        {
            Delete(seqlist, i+1);
            i--;
        }
    //Print(seqlist);
}
```

作业 2 线性结构的存储结构与应用

9) 线性表“逆置”

由于顺序储存结构长度已知且储存空间连续，因而可以采用头尾数据依次交换的形式实现线性表的“逆置”

```
141  //逆序
142  void Invert_order(SeqList *seqlist)
143  {
144      int tail = seqlist->length - 1;
145      int head = 0;
146      while(tail >= (seqlist->length/2))
147      {
148          int temp = seqlist->data[head];
149          seqlist->data[head] = seqlist->data[tail];
150          seqlist->data[tail] = temp;
151          head++;
152          tail--;
153      }
154      Print(seqlist);
155  }
```

10) 线性表循环右移 k 位

两层循环嵌套，每一次将当前队尾数据插入到队头，其余数据依次向后移动一位。左移操作基本相似。

```
157  //循环右移k位
158  void move_k_right(SeqList *seqlist, int k)
159  {
160      while(k > 0)
161      {
162          int temp = seqlist->data[seqlist->length-1];
163          for(int i = seqlist->length-1; i >= 0; i--)
164              seqlist->data[i]=seqlist->data[i-1];
165          seqlist->data[0] = temp;
166          k--;
167      }
168      for(int i = 0; i < MaxSize; i++)
169      {
170          if(seqlist->data[i] == -1)
171              break;
172          cout<<seqlist->data[i]<<endl;
173      }
174  }
```

11) 线性表循环左移 k 位

```
175  //循环左移k位
176  void move_k_left(SeqList *seqlist, int k)
177  {
178      while(k > 0)
179      {
180          int temp = seqlist->data[0];
181          for(int i = 0; i < seqlist->length; i++)
182              seqlist->data[i]=seqlist->data[i+1];
183          seqlist->data[seqlist->length-1] = temp;
184          k--;
185      }
186      for(int i = 0; i < MaxSize; i++)
187      {
188          if(seqlist->data[i] == -1)
189              break;
190          cout<<seqlist->data[i]<<endl;
191      }
192  }
```

作业 2 线性结构的存储结构与应用

12) 合并两线性表

利用顺序结构储存空间连续的特性，将线性表 2 依次插入到线性表 1 的队尾，同时数组长度随即发生变化。

```
194  //合并两个已排好序的线性表的算法。
195  void combine(SeqList *seqlist1, SeqList *seqlist2)
196  {
197      //排除非法输入
198      if(seqlist1->length + seqlist2->length > MaxSize)
199          cout<<"The wrong input"<<endl;
200      int head = 0;
201      for(int i = seqlist1->length ; i < MaxSize; i++)
202      {
203          seqlist1->data[i] = seqlist2->data[head];
204          head++;
205          if(seqlist1->data[i] == -1)
206              break;
207          seqlist1->length++;
208      }
209      for(int i = 0; i < MaxSize; i++)
210      {
211          if(seqlist1->data[i] == -1)
212              break;
213          cout<<seqlist1->data[i]<<endl;
214      }
215      cout<<"The length of SeqList is :"<<seqlist1->length<<endl;
216
217  }
```

二、线性表的链式存储结构及功能实现

1.顺序存储结构的实现

线性的链式存储结构是指用任意的存储单元来依次存放线性表的结点，这组单元既可以是连续的，也可以是不连续的，甚至是零散的分布在内存中的任意位置上。因此，简单的存储**同种数据类型**的线性表可以通过单链表实现，同时链表中的结点的逻辑次序和物理次序不一定相同。

定义链表结点结构如下：

```
6   typedef struct LinkList
7   {
8       int data;
9       LinkList *next;
10  }*pLinkList;
```

2.顺序存储结构的功能实现。

通过程序编写实现以下功能，

1) 数据信息初始化

通过文件读取，依次读入数据（文本中均为整型 int 数据类型）。每读入一个数据，为新节点申请内存同时将指针指向下一节点。记录数据长度。

作业 2 线性结构的存储结构与应用

```
//初始化
pLinkList initialize(struct LinkList *phead)
{
    int length = 0;
    ifstream infile;
    struct LinkList *p;
    p = applicate_memory();
    phead = p;
    p->next = NULL;
    infile.open("data_test.txt");
    while(!infile.eof())
    {
        struct LinkList *pNew;
        pNew = applicate_memory();
        if (pNew == NULL)
        {
            cout<<"No enough memory to import!\n"<<endl;
            exit(0);
        }
        infile>>pNew->data;
        pNew->next = NULL;
        p->next = pNew;
        p = p->next;
        length++;
    }
    infile.close();
    cout<<"The length of LinkList is :"<<length<<endl;
    return phead;
}
```

2) 输出当前数据信息

自链表头依次输出数据，同时输出当前链表长度。

```
//打印数据
void Print(struct LinkList *phead)
{
    struct LinkList *p;
    p = phead->next;
    int i = 1;
    while(p != NULL)
    {
        cout<<p->data<<endl;
        p = p->next;
        i++;
    }
   cout<<"The length of SeqList is :"<<i-1<<endl;
}
```

3) 删除顺序表中第 k 个数据

找到目标节点，将其前一节点的指针域指向此节点的下一节点，同时释放此节点内存。

```
void Delete(struct LinkList *phead, int k)
{
    struct LinkList *p;
    p = phead->next;
    int i = 1;//计数器
    while(p != NULL)
    {
        if(i == k)
        {
            struct LinkList *q;
            q = p->next;
            p->next = q->next;
            free(q);
            break;
        }
        i++;
        p = p->next;
```

4) 查找某一数据信息

遍历匹配目标数据信息，输出此数据以及其位置。

```
68  //查找数据
69  void Search(struct LinkList *phead, int x)
70  {
71      struct LinkList *p;
72      p = phead->next;
73      int i = 1;//计数器
74      while(p != NULL)
75      {
76          if(p->data == x)
77              cout<<"您查找的数据为"<<p->data<<",在顺序表中位置为"<<i<<endl;
78          i++;
79          p = p->next;
80      }
81
82  }
```

5) 修改数据信息

自表头遍历至目标位置，对该位置数据域进行修改。

```
84  //修改数据
85  void Change(struct LinkList *phead, int x)
86  {
87      struct LinkList *p;
88      p = phead->next;
89      int i = 1;//计数器
90      while(p != NULL)
91      {
92          if(i == x)
93          {
94              cout<<"将被修改的数据为:"<<p->data<<endl;
95              int m;
96              cout<<"要将其修改为："<<endl;
97              cin>>m;
98              p->data = m;
99          }
100         i++;
101         p = p->next;
102     }
103 }
```

6) 插入数据信息

自表头遍历至目标位置，让前一节点指向此插入节点，此节点的指针与指向原来此位置的节点。

```
106 void Insert(struct LinkList *phead, int x)
107 {
108     struct LinkList *p;
109     p = phead->next;
110     int i = 1;//计数器
111
112     int m;
113     cout<<"在位置"<<x<<"插入的数据为："<<endl;
114     cin>>m;
115     struct LinkList *pNew;
116     pNew = applicate_memory();
117     pNew->data = m;
```

```
118     while(p != NULL)
119     {
120         if(i + 1 == x)
121         {
122             struct LinkList *q;
123             q = p->next;
124             p->next = pNew;
125             pNew->next = q;
126             break;
127         }
128         i++;
129         p = p->next;
130     }
131 }
```

7) 排序

仅针对数据域进行排序交换，同时由于单链表仅能从头遍历至尾，故插入排序不太适用。

```
154 //选择排序
155 void sort(struct LinkList *phead)
156 {
157     struct LinkList *p,*q;
158     for (p = phead->next; p != NULL; p = p->next)
159         for (q = p->next; q != NULL; q = q->next)
160         {
161             if(p->data > q->data){
162                 int temp = p->data;
163                 p->data = q->data;
164                 q->data = temp;
165             }
166         }
167 }
```

8) 删除所有重复元素

对于排序好的序列，重复元素必在逻辑顺序上相邻。因此从第一个数据依次往后遍历，比较当前位置与下一相邻位置的数据是否相等，若相等，删除当前位置数据。在此并未出现存储位置的改变，因此不用像顺序结构那样回溯一步。

```
172 //对于已排好序的线性表,删除重复元素
173 void Delete_repeat(struct LinkList *phead)
174 {
175     sort(phead);
176     struct LinkList *p = phead->next;
177     int i = 1;
178     while(p->next != NULL)
179     {
180         if(p->data == p->next->data)
181             Delete(phead, i);
182         p = p->next;
183         i++;
184     }
185 }
```

作业 2 线性结构的存储结构与应用

9) 线性表“逆置”

利用**头插法**进行单链表的逆置，同时注意最后一个节点的逆置问题。

```
187  //逆序
188  void Invert_order(struct LinkList *phead)
189  {
190      struct LinkList *p,*q;
191      p = phead->next->next;
192      q = p->next;
193      phead->next->next = NULL;
194      while(p != NULL)
195      {
196          p->next = phead->next;
197          phead->next = p;
198          p = q;
199          if(q->next != NULL)
200              q = q->next;
201          else  //遍历至最后一个节点
202          {
203              p->next = phead->next;
204              phead->next = q;
205              break;
206          }
207      }
208  }
```

10) 线性表循环右移 k 位

和头插法思路类似，循环 k 次，每次将尾节点插入到表头之后。

```
210  //循环右移k位
211  void move_k_right(struct LinkList *phead, int k)
212  {
213      struct LinkList *p, *q;
214      while(k > 0)
215      {
216          p = phead->next;
217          while(p != NULL)
218          {
219              if(p->next->next == NULL)//遍历至尾节点
220              {
221                  q = phead->next;
222                  phead->next = p->next;
223                  p->next->next = q;
224                  p->next = NULL;
225              }
226              p = p->next;
227          }
228          k--;
229      }
230  }
```

11) 线性表循环左移 k 位

依次将第一个节点移动至表尾即可

```
232  //循环左移k位
233  void move_k_left(struct LinkList *phead, int k)
234  {
235      struct LinkList *p, *q;
236      while(k > 0)
237      {
238          q = phead->next;
239          phead->next = q->next;
240          q->next = NULL;
241          p = phead->next;
```

```
        while(p != NULL)
        {
            if(p->next == NULL)//遍历至尾节点
            {
                p->next = q;
                break;
            }
            p = p->next;
        }
        k--;
    }
}
```

12) 合并两线性表

利用指针，直接将线性表 2 的指针复制到线性表 1 的尾节点上即可实现两个线性表的合并。

```
//合并两个已排好序的线性表的算法。
void combine(struct LinkList *phead1,struct LinkList *phead2)
{
    struct LinkList *p;
    p = phead1->next;
    while(p != NULL)
    {
        if(p->next == NULL)//遍历至尾节点
        {
            p->next = phead2->next;
            break;
        }
        p = p->next;
    }
    return ;
}
```

三、顺序存储结构和链式存储结构的比较

1. 两种存储方式的优缺点

顺序存储时，相邻数据元素的存放地址也相邻（**逻辑与物理统一**）。其优点为查找便捷简单，缺点是面对大量数据时对内存要求较高，插入或删除元素时不方便。

链式存储时，相邻数据元素可随意存放（**逻辑与物理不统一**），优点是插入或删除元素时很方便，使用灵活，缺点是对链表的任何操作需要从表头开始进行遍历，使用效率较低。

2. 两种存储方式的应用场景比较

顺序表适宜于做查找这样的静态操作，但在插入/删除时需要移动元素个数较多。若线性表的长度变化不大，且其主要操作是查找，则可采用顺序表。

链表宜于做插入、删除这样的动态操作，此过程并不需要移动元素，只需要修改指针。若线性表的长度变化较大，且其主要操作是插入、删除操作，则采用链表。