第 3 章　抽象数据类型的实现

3.1 实验概要

实验项目名称：　抽象数据类型的实现
实验项目性质：　设计性实验
所属课程名称：　数据结构
实验计划学时：　6

3.2 实验目的

对某组具体的抽象数据类型，运用课程所学的知识和方法，设计合理的数据结构，并在此基础上实现该抽象数据类型的全部基本操作。通过本设计性实验，检验所学知识和能力，发现学习中存在的问题。 进而达到熟练地运用本课程中的基础知识及技术的目的。

3.3 预习与参考

1．确定要实现的抽象数据类型，并对基本操作做适当的选取和增加；
2．选择存储结构，并写出相应的类型定义；
3．设计各基本操作的实现算法，并表达为函数形式；
4．设计测试方案，编写主函数；
5．将上述 4 步的结果写成预习报告。

3.4 实验要求和设计指标

以教材中讨论的各种抽象数据类型为对象，利用 C 语言的数据类型表示和实现其中某个抽象数据类型。可选的抽象数据类型如下表所列：

编号	抽象数据类型	基本难度	存储结构
1	栈和队列	1.0	顺序　和　链接
2	线性表	1.0	顺序　和　链接
3	哈希表	1.1	任选
4	二叉树	1.2	任选
5	堆	1.2	任选
6	二叉排序树	1.2	任选
7	平衡二叉树	1.3	任选
8	树	1.2	任选
9	并查集	1.2	任选
10	B 树	1.4	任选
11	有向图	1.3	任选
12	无向图	1.3	任选
13	有向带权图	1.3	任选

注：如果基本操作数量较多，可选择实现其中一个基本操作子集。

实验要求如下：

1. 首先了解设计的任务，然后根据自己的基础和能力从中选择一题。一般来说，选择题目应以在规定的时间内能完成，并能得到应有的锻炼为原则。 若学生对教材以外的相关题目较感兴趣，希望选作实验的题目时，应征得指导教师的认可，并写出明确的抽象数据类型定义及说明。

2. 实验前要作好充分准备，包括：理解实验要求，掌握辅助工具的使用，了解该抽象数据类型的定义及意义，以及其基本操作的算法并设计合理的存储结构。

3. 实验时严肃认真，要严格按照要求独立进行设计，不能随意更改。注意观察并记录各种错误现象，纠正错误，使程序满足预定的要求，实验记录应作为实验报告的一部分。

4. 实验后要及时总结，写出实验报告，并附所打印的问题解答、程序清单，所输入的数据及相应的运行结果。

3.5 实验仪器设备和材料

计算机学院实验中心。

编程环境：AnyviewCL 可视化编程环境、TC++、C++Builder、VC++或 Java。

3.6 调试及结果测试

调试内容应包括：调试过程中遇到的问题是如何解决的以及对实验的讨论与分析；基本操作的时间复杂度和空间复杂度的分析和改进设想。列出对每一个基本操作的测试结果，包括输入和输出，测试数据应完整和严格。

3.7 考核形式

考核形式以实验过程和实验报告相结合的方式进行。在实验完成后，应当场运行和答辩，由指导教师验收，只有在验收合格后才能算实验部分的结束。实验报告作为整个设计性实验评分的书面依据。设计性实验的成绩评定以选定题目的难易度、完成情况和实验报告为依据综合评分。从总体来说，所实现的抽象数据类型应该全部符合要求，类型定义，各基本操作的算法以及存储结构清晰；各模快测试运行正确；程序的结构合理；设计报告符合规范。

3.8 实验报告要求

实验结束后要写出实验报告，以作为整个设计性实验评分的书面依据和存档材料。实验报告是反映学生实验效果的最主要的依据，也是学生正确地表达问题、综合问题和发现问题的能力的基本培养手段，因而是非常重要的内容。本设计性实验的报告要包括以下几项内容：

（1）设计任务、要求及所用软件环境或工具；

（2）抽象数据类型定义以及各基本操作的简要描述；

（3）所选择的存储结构描述及在此存储结构上各基本操作的实现；

（4）程序清单（计算机打印），输入的数据及各基本操作的测试结果；

（5）实验总结和体会。

实验报告以规定格式的电子文档书写、打印并装订，排版及图表要清楚、工整。

3.9 思考题

对设计性实验进行总结和讨论，包括本实验的优、缺点，数据存储结构的特点，与其它存储结构之间的比较等。通过总结，可以对抽象数据类型有更全面、深入的认识，这是设计性实验不可缺少的重要内容。这部分内容应作为实验报告中的一个组成部分。

3.10 示例

1．题目

采用字符类型为元素类型和无头结点单链表为存储结构，实现抽象数据类型List。

ADT List{

数据对象：D＝{ a_i | a_i∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1＝{ <a_{i-1}, a_i>|a_{i-1}, a_i∈D, i=2,...,n }

基本操作：

SetEmpty(**&**L)

操作结果：构造一个空的线性表L。

Destroy(**&**L)

初始条件：线性表L已存在。

操作结果：销毁线性表L。

Length(L)

初始条件：线性表L已存在。

操作结果：返回L中元素个数。

Get(L, i, **&**e)

初始条件：线性表L已存在，1≤i≤LengthList(L)。

操作结果：用e返回L中第i个元素的值。

Locate(L, e, compare())

初始条件：线性表L已存在，compare()是元素判定函数。

操作结果：返回L中第1个与e满足关系compare()的元素的位序。若这样的元素不存在，则返回值为0。

Insert(**&**L, i, e)

初始条件：线性表L已存在，1≤i≤LengthList(L)+1。

操作结果：在L的第i个元素之前插入新的元素e，L的长度加1。

Delete(**&**L, i, **&**e)

初始条件：线性表L已存在且非空，1≤i≤LengthList(L)。

操作结果：删除L的第i个元素，并用e返回其值，L的长度减1。

Display(L)

初始条件：线性表L已存在。

操作结果：依次输出L的每个元素。

} ADT List

2．存储结构定义

公用头文件DS0.h:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <values.h>

#define TRUE   1
#define FALSE  0
#define OK     1
```

```
#define ERROR  0
#define IBFEASIBLE  -1
#define OVERFLOW   -2

#define MAXLEN  20
#define MAXSIZE 20

typedef int Status;
typedef char ElemType; /* 元素类型为字符类型*/
```

（1） 顺序存储结构

```
#define LIST_INIT_SIZE 20 /*线性表存储空间的初始分配量*/
#define LISTINCREMENT  10  /*线性表存储空间的分配增量*/
typedef struct{
    ElemType *elem; /*存储空间基址*/
    int   length;    /*当前长度*/
    int   listsize; /*当前分配的存储容量*/
} SqList;
```

（2） 无头结点单链表存储结构

```
typedef struct LNode {
   ElemType data;
   struct LNode *next;
} LNode, *LList;     /* 不带头结点单链表类型*/
```

（3） 带头结点单链表存储结构

```
typedef struct LNode {  /* 结点类型 */
   ElemType data;
   struct LNode *next;
} LNode, *Link, *Position;

typedef struct LinkList { /* 链表类型 */
   Link head,tail;  /* 分别指向线性链表中的头结点和最后一个结点 */
   int len;         /* 指示线性链表中数据元素的个数 */
} LinkList;
```

3．算法设计

（1） 顺序存储结构

```
Status SetEmpty(SqList &L) { /*构造一个空的顺序线性表 */

L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
  if(!L.elem)
    return OVERFLOW; /* 存储分配失败 */
```

```
   L.length=0; /* 空表长度为 0 */
   L.listsize=LIST_INIT_SIZE; /* 初始存储容量 */
   return OK;
 }
Status Destroy (SqList &L) { /*销毁顺序线性表 L */
   free(L.elem);
   L.elem=NULL;
   L.length=0;
   L.listsize=0;
   return OK;
}
int Length(SqList L) { /* 求表长*/
   return L.length;
}
Status Get(SqList &L, int i, ElemType &e) { /* 获取第 i 元
素 */
   if(i<1||i>L.length)
      return ERROR;
   e=*(L.elem+i-1);
   return OK;
}
int Locate(SqList L, ElemType x) { /* 确定 x 在表中的位序 */
   ElemType *p;
   int i=1; /* i 的初值为第 1 个元素的位序 */
   p=L.elem; /* p 的初值为第 1 个元素的存储位置 */
   while(i<=L.length && *p++!=x)
      ++i;
   if(i<=L.length)
      return i;
   else
      return 0;
}

Status Insert(SqList &L, int i, ElemType e) {
   /* 操作结果：在 L 中第 i 个位置之前插入新的数据元素 e，L 的长度加
1 */
   ElemType *newbase,*q,*p;
   if(i<1||i>L.length+1) /* i 值不合法 */
      return ERROR;
   if(L.length>= L.listsize) { /* 当前存储空间已满,增加分配
*/
      newbase=(ElemType *) realloc(L.elem,

(L.listsize+LISTINCREMENT)*sizeof(ElemType));
```

```
      if(!newbase) return OVERFLOW; /* 存储分配失败 */
      L.elem=newbase; /* 新基址 */
      L.listsize+=LISTINCREMENT; /* 增加存储容量 */
    }
    q=L.elem+i-1; /* q为插入位置 */
    for(p=L.elem+L.length-1; p>=q; --p)
      *(p+1)=*p;  /* 插入位置及之后的元素右移 */
    *q=e; /* 插入e */
    ++L.length; /* 表长增1 */
    return OK;
}
Status Delete(SqList &L, int i, ElemType &e) {
    /* 初始条件：顺序线性表L已存在，1≤i≤ListLength(L) */
    /* 操作结果：删除L的第i个数据元素，并用e返回其值，L的长度
减1 */
    ElemType *p,*q;
    if(i<1||i> L.length) /* i值不合法 */
      return ERROR;
    p= L.elem+i-1; /* p为被删除元素的位置 */
    e=*p; /* 被删除元素的值赋给e */
    q= L.elem+L.length-1; /* 表尾元素的位置 */
    for(++p; p<=q; ++p) /* 被删除元素之后的元素左移 */
      *(p-1)=*p;
    L.length--; /* 表长减1 */
    return OK;
}
Status Display(SqList L) { /* 依次显示表中元素 */
    ElemType *p;
    int i;
    p=L.elem;
    printf("( ");
    for(i=1; i<=L.length; i++)
      printf("%c",*p++);
    printf(" )\n");
    return OK;
}
```

（2） 无头结点单链表

```
void SetEmpty(LList &L) { /* 置无头结点的空单链表*/
    L=NULL;
}
Status Destroy (LList &L) { /* 销毁链表*/
    LList q=L;
    while(L) {
      L=L->next;
```

```
      free(q);
      q=L;
   }
   return OK;
}
int Length(LList L) { /* 求表长*/
   int n=0;
   while(L!=NULL) {
      n++;
      L=L->next;
   }
   return n;
}
Status Get(LList L, int i, ElemType &e) { /* 获取第 i 元素
*/
   int j=1;
   while (j<i && L!=NULL) {
      L=L->next;
      j++;
   }
   if(L!=NULL) { e=L->data;  return OK; }
   else return ERROR; /* 位置参数 i 不正确 */
}
int Locate(LList L, ElemType x) { /* 确定 x 在表中的位序 */
   int n=1;
   while (L!=NULL && L->data!=x) {
      L=L->next;
      n++;
   }
   if (L==NULL) return 0;
   else return n;
}
Status Insert(LList &L, int i, ElemType e) { /* 插入第 i
元素*/
   int j=1;
   LList s,q;
   s=(LList)malloc(sizeof(LNode));
   s->data=e;
   q=L;
   if (i==1) { s->next=q;  L=s;  return OK;}
   else {
      while(j<i-1 && q->next!=NULL) {
         q=q->next;
         j++;
```

```
      }
      if (j==i-1) {
         s->next=q->next;
         q->next=s;
         return OK;
      }
      else return ERROR;  /* 位置参数 i 不正确 */
   }
}
Status Delete(LList &L, int i, ElemType &e) { /* 删除第 i 元素*/
   int j=1;
   LList q=L,t;
   if (i==1) {
      e=q->data;
      L=q->next;
      free(q);
      return OK;
   }
   else {
      while (j<i-1 && q->next!=NULL) {
         q=q->next;
         j++;
      }
      if (q->next!=NULL && j==i-1) {
         t=q->next;
         q->next=t->next;
         e=t->data;
         free(t);
         return OK;
      }
      else return ERROR;  /* 位置参数 i 不正确*/
   }
}
void Display(LList L) { /* 依次显示表中元素 */
   printf("单链表显示: ");
   if (L==NULL)
      printf("链表为空!");
   else if (L->next==NULL)
      printf("%c\n", L->data);
   else {
      while(L->next!=NULL) {
         printf("%c->", L->data);
         L=L->next;
```

```
        }
        printf("%c", L->data);
    }
    printf("\n");
}
```

（3） 带头结点单链表

```
Status SetEmpty(LinkList &L) { /* 置带头结点的空单链表*/
    Link p;
    p=(Link)malloc(sizeof(LNode)); /* 生成头结点 */
    if(p) {
        p->next=NULL;
        L.head=L.tail=p;
        L.len=0;
        return OK;
    }
    else
        return ERROR;
}
Status Destroy(LinkList &L) { /* 销毁线性链表 L，L 不再存在 */
    Link p,q;
    if(L.head!=L.tail) { /* 不是空表 */
        p=q= L.head->next;
        L.head->next=NULL;
        while(p!=L.tail) {
            p=q->next;
            free(q);
            q=p;
        }
        free(q);
    }
    free(L.head);
    L.head=L.tail=NULL;
    L.len=0;
    return OK;
}
int Length(LinkList L) { /* 返回线性链表 L 中元素个数 */
    return L.len;
}
Status Get(LinkList L, int i, ElemType &e) { /* 获取第 i
元素 */
    /* i=0 为头结点 */
    Link p;
    int j;
    if(i<1||i>L.len)
```

```
      return ERROR;
   else {
      p=L.head;
      for(j=1;j<=i;j++)
         p=p->next;
      e=p->data;
      return OK;
   }
}
int Locate(LinkList L, ElemType x) { /* 确定 x 在表中的位序
*/
   int i=0;
   Link p=L.head;
   do {
      p=p->next;
      i++;
   } while(p && p->data!=x); /* 没到表尾且没找到满足关系的元
素 */
   if (!p)
      return 0;
   else
      return i;
}
Status Insert(LinkList &L, int i, ElemType e) { /* 插入
第 i 元素*/
   int j=0;
   Link s,q;
   s=(Link)malloc(sizeof(LNode));
   s->data=e;
   q=L.head;
   while(j<i-1 && q->next!=NULL) {
      q=q->next;
      j++;
   }
   if (j==i-1) {
      s->next=q->next;
      q->next=s;
      if (L.tail==q) L.tail=s;
      L.len++;
      return OK;
   }
   else return ERROR;  /* 位置参数 i 不正确 */
}
Status Delete(LinkList &L, int i, ElemType &e) {
```

```
  /* 删除第 i 元素*/
  int j=0;
  Link q=L.head,t;
  while (j<i-1 && q->next!=NULL) {
    q=q->next;
    j++;
  }
  if (q->next!=NULL && j==i-1) {
    t=q->next;
    q->next=t->next;
    e=t->data;
    if(L.tail==t) L.tail=q;
    free(t);
    L.len--;
    return OK;
  }
  else return ERROR;  /* 位置参数 i 不正确*/
}
void Display(LinkList L) { /* 依次显示表中元素 */
  Link p;
  printf("单链表显示: ");
  if (L.head==L.tail)
    printf("链表为空!");
  else {
    p=L.head->next;
    while(p->next!=NULL) {
      printf("%c->", p->data);
      p=p->next;
    }
    printf("%c", p->data);
  }
  printf("\n");
}
```

4. 测试

（1） 顺序存储结构

```
SqList head;
void main() { /* 主函数*/
  char e,c;
  int i,n,select,x1,x2,x3,x4,m,g;
  SetEmpty(head);
  n=random(8); /* 随机产生表长 */
  for (i=1; i<=n; i++) { /* 将数据插入到顺序表中 */
    c='A'+random(26);
```

```
      Insert(head,i,c);
   }
   do {
      Display(head);
      printf("select 1 求长度 Length()\n");
      printf("select 2 取结点 Get()\n");
      printf("select 3 求值查找 Locate()\n");
      printf("select 4 删除结点 Delete()\n");
      printf("input your select: ");
      scanf("%d",&select);
      switch(select) {
         case 1: x1=Length(head);
                 printf("顺序表的长度:%d ",x1);
                 break;
         case 2: printf("请输入要取的结点序号: ");
                 scanf("%d",&m);
                 if(Get(head,m,x2)) printf("%c\n",x2);
                 else printf("错误\n");
                 break;
         case 3: printf("请输入要查找的数据元素: ");
                 scanf("\n%c",&e);
                 x3=Locate(head,e);
                 printf("%d\n",x3);
                 break;
         case 4: printf("请输入要删除的元素序号: ");
                 scanf("%d",&g);
                 if(Delete(head,g,x4))
printf("%c\n",x4);
                 else printf("错误\n");
                 break;
      }
   } while (select>0 && select <5);
}
```

（2） 无头结点单链表

```
LList head;
void main() { /* 主函数*/
   char e,c;
   int i,n,select,x1,x2,x3,x4,m,g;
   SetEmpty(head);
   n=random(8); /* 随机产生表长 */
   for (i=1; i<=n; i++) { /* 将数据插入到顺序表中 */
      c='A'+random(26);
      Insert(head,i,c);
   }
```

```
    do {
       Display(head);
       printf("select 1 求长度 Length()\n");
       printf("select 2 取结点 Get()\n");
       printf("select 3 求值查找 Locate()\n");
       printf("select 4 删除结点 Delete()\n");
       printf("input your select: ");
       scanf("%d",&select);
       switch(select) {
          case 1: x1=Length(head);
                  printf("顺序表的长度:%d ",x1);
                  break;
          case 2: printf("请输入要取的结点序号: ");
                  scanf("%d",&m);
                  if(Get(head,m,x2)) printf("%c\n",x2);
                  else printf("错误\n");
                  break;
          case 3: printf("请输入要查找的数据元素: ");
                  scanf("\n%c",&e);
                  x3=Locate(head,e);
                  printf("%d\n",x3);
                  break;
          case 4: printf("请输入要删除的元素序号: ");
                  scanf("%d",&g);
                  if(Delete(head,g,x4))
printf("%c\n",x4);
                  else printf("错误\n");
                  break;
       }
    } while (select>0 && select <5);
}
```

（3） 带头结点单链表

```
LinkList head;
void main() { /* 主函数*/
   char e,c;
   int i,n,select,x1,x2,x3,x4,m,g;
   SetEmpty(head);
   n=random(8); /* 随机产生表长 */
   for (i=1; i<=n; i++) { /* 将数据插入到顺序表中 */
      c='A'+random(26);
      Insert(head,i,c);
   }
   do {
      Display(head);
```

```
        printf("select 1 求长度 Length()\n");
        printf("select 2 取结点 Get()\n");
        printf("select 3 求值查找 Locate()\n");
        printf("select 4 删除结点 Delete()\n");
        printf("input your select: ");
        scanf("%d",&select);
        switch(select) {
           case 1: x1=Length(head);
                   printf("顺序表的长度:%d ",x1);
                   break;
           case 2: printf("请输入要取的结点序号: ");
                   scanf("%d",&m);
                   if(Get(head,m,x2)) printf("%c\n",x2);
                   else printf("错误\n");
                   break;
           case 3: printf("请输入要查找的数据元素: ");
                   scanf("\n%c",&e);
                   x3=Locate(head,e);
                   printf("%d\n",x3);
                   break;
           case 4: printf("请输入要删除的元素序号: ");
                   scanf("%d",&g);
                   if(Delete(head,g,x4))
printf("%c\n",x4);
                   else printf("错误\n");
                   break;
        }
      } while (select>0 && select<5);
}
```

5．三种存储结构的比较

	存储结构	顺序映象	无头结点单链表	带头结点单链表
基本操作时间复杂度	SetEmpty(&L)	O(1)	O(1)	O(1)
	Destroy(&L)	O(1)	O(n)	O(n)
	Length(L)	O(1)	O(n)	O(1)
	Get(L, i, &e)	O(1)	O(n)	O(n)
	Locate(L,e,compare())	O(n)	O(n)	O(n)
	Insert(&L, i, e)	O(n)	O(n)	O(n)
	Delete(&L, i, &e)	O(n)	O(n)	O(n)
	Display(L)	O(n)	O(n)	O(n)

优缺点分析	优 点	可以随机存取	插入删除时不需要移动元素	1．对空表不需要额外进行判断处理； 2．求表长度方便
	缺 点	插入删除时需要移动元素	1．对空表需要额外进行判断处理； 2．求表长不方便	不能随机存取

6．思考与小结

（1）无头结点单链表的插入和删除操作的实现，要区分该表是否为空，并编写相应的代码。

（2）在算法设计时，要注意判断有关参数值的合法性。

（3）三种存储结构的主函数相同。设计主函数及测试运行时，要考虑测试的完备性。

7．预习报告和实验报告

（1）预习报告：包括 1-4 步的初稿。

（2）实验报告：在预习报告的基础上，增加在实验中，对算法修改核调试的收获体会，以及思考和小结的内容。