第 1 章 基 础 实 验

CHAPTER 1

1.1 线性表

本节要点

(1) 顺序表和链表的概念。

(2) 顺序表和链表的存储结构。

(3) 顺序表和链表的操作实现。

学习目标

(1) 理解顺序表和链表之间的区别和联系。

(2) 掌握顺序存储结构和链式存储结构的数据类型定义方法。

(3) 掌握顺序存储结构和链式存储结构的各种操作的实现。

(4) 掌握如何使用线性表来解决相关的应用问题。

基本知识点

线性表按照存储结构不同分为顺序表和单链表。顺序表用一组地址连续的存储单元依次存放线性表中的数据元素,单链表是指用一组任意的存储单元存放线性表的元素,地址可以连续也可以不连续;顺序表可以随机存取,但是插入或者删除的时候需要移动大量元素,单链表只能顺序存取,但插入和删除时,不需要大量移动数据,表的容量不受限制。

实验 1　顺序表基本功能实现

1. 实验目的

(1) 掌握顺序表的存储结构。
(2) 验证顺序表及其基本操作的实现。
(3) 理解算法与程序的关系,能够将顺序表算法转换为对应的程序。

2. 实验内容

(1) 初始化顺序表。
(2) 在顺序表的第 i 个位置插入元素。
(3) 删除顺序表的第 i 个元素。
(4) 输出顺序表。
(5) 判断顺序表是否为空。
(6) 判断顺序表是否为满。
(7) 求顺序表第 i 个元素的值。
(8) 查找值为 x 的元素。

3. 算法设计

用结构体来描述顺序表。在结构体的定义中,包含顺序表的长度、数组的定义以及表的最大容量 3 个属性。有时最大容量可以采用常量定义的形式。

结构体的定义如下:

```
typedef int dataType;
#define MAXSIZE 10000;
typedef struct{
dataType *elem;
 int length;
}
SqList;
```

应该实现顺序表的初始化、插入、删除、输出、判空、判满、求值及查找等操作。

```
void InitList(SqList &L);                    //初始化顺序表
void Insert(SqList &L,int i,dataType x );    //在顺序表第 i 个位置插入元素
void Delete(SqList &L,int i );               //删除顺序表的第 i 个元素
void Print(SqList &L);                       //输出顺序表
```

```
int Empty((SqList &L);                          //判断顺序表是否为空
int Full(SqList &L);                            //判断顺序表是否为满
int GetData(SqList &L,int i);                   //求顺序表第 i 个元素的值
int Locate(SqList &L,dataType x);               //查找值为 x 的元素
```

4. 程序实现

程序主函数的实现代码如下,各函数的实现代码参考主教材。

```
#include <iostream>
#include <string.h>
using namespace std;
#define MAXSIZE 100
typedef struct{
dataType *elem;
int length;
}
SqList;
int main(int argc, char *argv[]) {
 SqList L;
 int a;
 int x;
printf("1 初始化顺序表\n");
printf("2. 在顺序表的第 i 个位置插入元素\n");
printf("3. 删除顺序表的第 i 个元素\n");
printf("4. 输出顺序表\n");
printf("5. 判断顺序表是否为空 \n");
printf("6. 判断顺序表是否为满 \n");
printf("7. 求顺序表第 i 个元素的值 \n");
printf("8. 查找值为 x 的元素 \n");
 while(1)
 {
    printf("输入你想要进行的操作序号: ");
    scanf("%d",&a);
    switch(a)
    {
        case 1: InitList (&L);
case 2: Insert(SqList &L, i, x );break;
case 3: Delete(SqList &L, i );break;
case 4: Print(SqList &L);break;
case 5: Empty((SqList &L););break;
case 6: Full(SqList &L);break;
case 7: GetData(SqList &L, i); break;
case 8: Locate(SqList &L, x); break;
       default: printf("输入错误,重新输入:");break;
```

```
    }
    printf("是否要继续操作(Y/N): ");
    getchar();
    scanf(" %c",&x);
    if(x!= 'Y') break;
}
return 0;
}
```

实验 2　链表基本功能的实现

1. 实验目的

(1) 掌握线性表的链式存储结构。

(2) 验证链表基本操作的实现。

(3) 学会使用链表来解决各种实际问题。

2. 实验内容

(1) 初始化链表。

(2) 建立单链表。

(3) 在链表的第 i 个位置插入元素。

(4) 删除链表的第 i 个元素。

(5) 输出链表。

(6) 判断链表是否为空。

(7) 求链表中第 i 个元素的值。

(8) 查找值为 x 的元素。

(9) 清空链表。

3. 算法设计

在链式存储中,结点的结构如下:

```
Typedef int dataType;
typedef struct Lnode
{
 dataType data;
 struct Lnode * next;
}Lnode, * Linklist;
```

为简单起见，本实验假定表的数据元素为 int 型。

定义一个指针 head，指向表头结点，实现链表的初始化、插入、删除、判空、求值、定位查找、按值查找、输出和清空链表等操作。

```
void InitList(SqList &L);                          //初始化单链表
void Creatlist(SqList &L,int n);                   //建立单链表
void Insert(SqList &L,int i,dataType x );          //在单链表第 i 个位置插入元素
void Delete(SqList &L,int i );                     //删除单链表的第 i 个元素
void Print(SqList &L);                             //输出单链表
int Empty((SqList &L);                             //判断链表是否为空
int GetData(SqList &L,int i)                       //求链表中第 i 个元素的值
int Locate(SqList &L,dataType x);                  //在链表中查找值为 x 的元素
void Clearlist(SqList &L);                         //清空链表
```

4. 程序实现

程序主函数的实现代码如下，各函数的实现代码参考主教材。

```
#include <iostream>
#include <string.h>
using namespace std;
Typedef int dataType;
 typedef struct Lnode
  {
  dataType data;
  struct Lnode * next;
}Lnode, * Linklist;

int main(int argc, char * argv[]) {
Linklist L;
int i;
int x;
int a;
printf("1.初始化单链表\n");
printf("2. 建立单链表\n");
printf("3. 在单链表第 i 个位置插入元素\n");
printf("4. 删除单链表的第 i 个元素\n");
printf("5. 输出单链表 \n");
printf("6. 判断链表是否为空 \n");
printf("7. 求链表中第 i 个元素的值 \n");
printf("8. 在链表中查找值为 x 的元素 \n");
printf("9. 清空链表 \n");
```

```
while(1)
{
    printf("输入你想要进行的操作序号: ");
    scanf("%d",&a);
    switch(a)
    {
        case 1: InitList (&L);
        case 2: Creatlist(SqList &L, n);
        case 2: Insert(SqList &L, i, x );break;
        case 3: Delete(SqList &L, i );break;
        case 4: Print(SqList &L);break;
        case 5: Print(SqList &L);break;
        case 6: Empty((SqList &L););break;
        case 7: GetData(SqList &L, i); break;
        case 8: Locate(SqList &L, x); break;
        case 9: Clearlist(SqList &L);break;
        default: printf("输入错误,重新输入:");break;
    }
    printf("是否要继续操作(Y/N): ");
    getchar();
    scanf("%c",&x);
    if(x!= 'Y') break;
}
return 0;
}
```

1.2 栈和队列

本节要点

(1) 栈和队列的概念。

(2) 栈和队列的存储结构。

(3) 栈和队列的操作实现。

学习目标

(1) 理解栈和队列的概念、联系和差别。

(2) 掌握栈和队列的顺序存储结构和链式存储结构的数据类型定义方法。

(3) 掌握顺序存储结构和链式存储结构的各种操作的实现。

(4) 掌握如何使用栈和队列来解决相关的应用问题。

在线性结构中,栈和队列非常重要。尽管栈和队列都是特殊的线性表,但它们的操作都是受限的。具体来说,栈的操作是“后进先出”,而队列的操作则是“先进先出”。因此,它们都是操作受限的线性表。从抽象数据类型的角度看,它们处理各自元素的方法有很大差别。

实验 3　栈和队列的基本功能实现(1)
——栈的顺序表示和实现

1. 实验目的

(1) 掌握顺序栈的存储结构。

(2) 验证顺序栈及其基本操作的实现。

(3) 理解算法与程序的关系,能够将顺序栈算法转换为对应的程序。

2. 实验内容

(1) 初始化顺序栈。

(2) 取出栈顶元素。

(3) 出栈。

(4) 入栈。

3. 算法设计

用结构体来描述顺序栈,结构体的定义中包含栈顶指针、栈底指针和顺序栈的长度。结构体的定义如下:

```
typedef char SElemType;
typedef struct
{
   SElemType * base;                    //栈底指针
   SElemType * top;                     //栈顶指针
   int stacksize;                       //栈可用的最大容量
} SqStack;
```

实现顺序表的初始化、取栈顶元素、入栈、出栈等操作。

(1) InitStack(SqStack &S)。

(2) GetTop(SqStack S, SElemType &e)。

(3) Push(SqStack &S, SElemType e)。

(4) Pop(SqStack &S, SElemType &e)。

4. 程序实现

程序主函数代码如下：

```
//顺序栈的实现
#include<iostream>
#include<fstream>
using namespace std;
//顺序栈的定义
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE 100
typedef int Status;
typedef char SElemType;
typedef struct{
 SElemType *base;                          //栈底指针
 SElemType *top;                           //栈顶指针
int stacksize;                             //栈可用的最大容量
 }SqStack;
//主函数
int main()
{
   SqStack s;
   int choose, flag=0;
   SElemType j,t;
   cout<<"1. 初始化\n";
   cout<<"2. 入栈\n";
   cout<<"3. 读取栈顶元素\n";
   cout<<"4. 出栈\n";
   cout<<"0. 退出\n";
   choose = -1;
   while(choose!=0)
   {
      cout<<"请选择: ";
      cin>>choose;
      switch(choose)
      {
```

```
case 1:
   if(InitStack(s))
   {
      flag = 1;
      cout <<"成功对栈初始化\n\n";
   }
   else
      cout <<"初始化栈失败\n\n";
   break;
case 2:
{
   fstream file;
   file.open("SqStack.txt");
   if(!file)
   {
      cout <<"错误!未找到文件!"<< endl << endl;
      exit(ERROR);
   }
   if(flag)
   {
      flag = 1;
      cout <<"进栈元素依次为: \n";
      while(!file.eof())
      {
         file >> j;
         if(file.fail())
            break;
         else
         {
            Push(s,j);
            cout << j <<" ";
         }
     }
     cout << endl << endl;
}
else
    cout <<"栈未建立,请重新选择\n\n";
file.close();
}
break;
case 3:
   if(flag!= -1&&flag!= 0)
      cout <<"栈顶元素为: \n"<< GetTop(s)<< endl << endl;
   else
```

```
            cout <<"栈中无元素,请重新输入\n"<< endl;
         break;
      case 4:
         cout <<"依次出栈的元素为: \n";
         while(Pop(s,t))
            cout << t <<" ";
         flag = -1;
         cout << endl << endl;
         break;
      }
   }
   return 0;
}
```

实验 4　栈和队列的基本功能实现(2)
——栈的链式表示和实现

1. 实验目的

(1) 掌握链栈的存储结构。

(2) 验证链栈及其基本操作的实现。

(3) 理解算法与程序的关系,能够将链栈算法转换为对应的程序。

2. 实验内容

(1) 初始化链栈。

(2) 取出栈顶元素。

(3) 出栈。

(4) 入栈。

3. 算法设计

用结构体来描述链栈,结构体的定义中包含数据域和指针域。结构体的定义如下:

```
typedef char SElemType;
typedef struct StackNode
{
   SElemType data;
   struct StackNode * next;
} StackNode, * LinkStack;
```

实现链栈的初始化、取栈顶元素、入栈、出栈等操作。

(1) InitStack(SqStack &S)。

(2) GetTop(SqStack S, SElemType &e)。

(3) Push(SqStack &S, SElemType e)。

(4) Pop(SqStack &S, SElemType &e)。

4. 程序实现

程序主函数代码如下：

```
//链栈的实现
#include<iostream>
#include<fstream>
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE 100
typedef int Status;
typedef char SElemType;
typedef struct StackNode
{
   SElemType data;
   struct StackNode *next;
} StackNode, *LinkStack;
…
//主函数
int main()
{
   LinkStack s;
   int choose, flag=0;
   SElemType j,t;
   cout<<"1. 初始化\n";
   cout<<"2. 入栈\n";
   cout<<"3. 读取栈顶元素\n";
   cout<<"4. 出栈\n";
   cout<<"0. 退出\n";
   choose = -1;
   while(choose!=0)
   {
      cout<<"请选择: ";
      cin>>choose;
      switch(choose)
```

```
{
case 1:
   if(InitStack(s))
   {
      flag = 1;
      cout <<"成功对栈初始化\n\n";
   }
   else
      cout <<"初始化栈失败\n\n";
   break;
case 2:
{
   fstream file;
   file.open("SqStack.txt");
   if(!file)
   {
      cout <<"错误!未找到文件!"<< endl << endl;
      exit(ERROR);
   }
   if(flag)
   {
      flag = 1;
      cout <<"进栈元素依次为: \n";
      while(!file.eof())
      {
         file >> j;
         if(file.fail())
            break;
         else
         {
            Push(s,j);
            cout << j <<" ";
         }
      }
      cout << endl << endl;
   }
   else
      cout <<"栈未建立,请重新选择\n\n";
   file.close();
}
break;
case 3:
   if(flag!= -1&&flag!= 0)
      cout <<"栈顶元素为: \n"<< GetTop(s)<< endl << endl;
```

```
            else
                cout <<"栈中无元素,请重新输入\n"<< endl;
            break;
        case 4:
            cout <<"依次出栈的元素为: \n";
            while(Pop(s,t))
                cout << t <<" ";
            flag = -1;
            cout << endl << endl;
            break;
        }
    }
    return 0;
}
```

实验 5　栈和队列的基本功能实现(3)
——队列的链式表示和存储

1. 实验目的

(1) 掌握链队的存储结构。

(2) 验证链队及其基本操作的实现。

(3) 理解算法与程序的关系,能够将链队算法转换为对应的程序。

2. 实验内容

(1) 初始化链队。

(2) 入队。

(3) 出队。

(4) 取出队头元素。

3. 算法设计

用结构体来描述链队。第一个结构体定义了链队中的结点元素,包含数据域和指针域;第二个结构体定义了整个链队,包含队头指针和队尾指针。结构体的定义如下:

```
typedef char QElemType;
typedef char SElemType;
//链队的链式存储
```

```
typedef struct QNode
{
   QElemType data;
   struct QNode * next;
}
QNode, * QueuePtr;
typedef struct
{
   QueuePtr front;
   QueuePtr rear;
}
LinkQueue;
```

应该实现链队的初始化、入队、出队、取队头元素等操作。

```
InitQueue(LinkQueue &Q);
EnQueue(LinkQueue &Q, QElemType e);
DeQueue(LinkQueue &Q, QElemType &e);
GetHead(LinkQueue Q);
```

4. 程序实现

程序主函数实现代码如下：

```
//链队的实现
#include < iostream >
#include < fstream >
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW - 2
typedef int Status;
typedef char QElemType;
typedef char SElemType;
//链队的链式存储
typedef struct QNode
{
   QElemType data;
   struct QNode * next;
} QNode, * QueuePtr;
typedef struct
{
   QueuePtr front;
   QueuePtr rear;
} LinkQueue;
```

```
…
//主函数
int main()
{
   int choose, flag = 0;
   LinkQueue Q;
   QElemType j,e;
   cout <<"1. 初始化\n";
   cout <<"2. 入队\n";
   cout <<"3. 读取队头元素\n";
   cout <<"4. 出队\n";
   cout <<"0. 退出\n";
   choose = -1;
   while(choose!= 0)
   {
      cout <<"请选择: ";
      cin >> choose;
      switch(choose)
      {
      case 1:
         if(InitQueue(Q))
         {
            flag = 1;
            cout <<"成功对队列初始化\n\n";
         }
         else
            cout <<"初始化队列失败\n\n";
         break;
      case 2:
      {
         fstream file;
         file.open("QNode.txt");
         if(!file)
         {
            cout <<"错误!未找到文件!\n\n";
            exit(ERROR);
         }
         if(flag)
         {
            flag = 1;
            cout <<"入队元素依次为: \n";
            while(!file.eof())
            {
               file >> j;
```

```
                if(file.fail())
                    break;
                else
                {
                    EnQueue(Q,j);
                    cout << j <<" ";
                }
            }
            cout << endl << endl;
        }
        else
            cout <<"队列未建立,请重新选择\n\n";
        file.close();
    }
    break;
    case 3:
        if(flag!= -1&&flag!= 0)
            cout <<"队头元素为: \n"<< GetHead(Q)<< endl << endl;
        else
            cout <<"队列中无元素,请重新选择\n"<< endl;
        break;
    case 4:
        cout <<"依次弹出队列元素为: \n";
        while(DeQueue(Q,e))
        {
            flag = -1;
            cout << e <<" ";
        }
        cout << endl << endl;
        break;
    }
  }
  return 0;
}
```

实验 6　栈和队列的基本功能实现(4)
——队列的顺序表示和实现

1. 实验目的

(1) 掌握顺序队列的存储结构。

(2) 验证顺序队列及其基本操作的实现。

（3）理解算法与程序的关系，能够将顺序队列算法转换为对应的程序。

2. 实验内容

（1）初始化顺序队列。

（2）入队。

（3）出队。

（4）取出队头元素。

3. 算法设计

用结构体来描述顺序队列。结构体中包含数组首地址的指针、队头游标和队尾游标。结构体的定义如下：

```
#define MAXQSIZE 100
typedef struct{
 QElemType * base;
 int front;
 int rear;
 }SqQueue;
```

实现顺序队列的初始化、取队头元素、入队、出队等操作。

```
InitStack(SqStack &S);
GetTop(SqStack S, SElemType &e);
Push(SqStack &S, SElemType e);
Pop(SqStack &S, SElemType &e);
```

4. 程序实现

程序主函数实现代码如下：

```
//循环队列
#include<iostream>
#include<fstream>
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXQSIZE 100
typedef char QElemType;
typedef int Status;
typedef char SElemType;
typedef struct
```

```
{
    QElemType * base;
    int front;
    int rear;
}
SqQueue;
…
//主函数
int main()
{
    int choose,flag = 0;
    SqQueue Q;
    QElemType e,j;
    cout <<"1.初始化\n";
    cout <<"2.入队\n";
    cout <<"3.读队头元素\n";
    cout <<"4.出队\n";
    cout <<"0.退出\n";
    choose = - 1;
    while(choose!= 0)
    {
        cout <<"请选择: ";
        cin >> choose;
        switch(choose)
        {
        case 1:
            if(InitQueue(Q))
            {
                flag = 1;
                cout <<"初始化队列成功\n\n";
            }
            else
                cout <<" 初始化队列失败\n\n";
            break;
        case 2:
        {
            fstream file;
            file.open("QNode.txt");
            if(!file)
            {
                cout <<"错误!未找到文件!\n\n\n";
                exit(ERROR);
            }
            if (flag)
            {
                flag = 1;
```

```
            cout << "入队的元素依次为: \n";
            while (!file.eof())
            {
               file >> j;
               if (file.fail())
                  break;
               else
               {
                  EnQueue(Q, j);
                  cout << j << " ";
               }
            }
            cout << endl << endl;
         }
         else
            cout <<"队列未建立,请重新选择\n\n";
         file.close();
      }
      break;
      case 3:
         if(flag!= -1&&flag!= 0)
            cout <<"队头元素为: \n"<< GetHead(Q)<< endl << endl;
         else
            cout <<"队列中无元素,请重新选择: \n";
         break;
      case 4:
         cout <<"依次弹出队列元素为: \n";
         while(DeQueue(Q,e))
         {
            flag = -1;
            cout << e <<" ";
         }
         cout << endl << endl;
         break;
      }
   }
   return 0;
}
```

实验 7　栈的应用——数制转换

1. 实验目的

(1) 掌握栈的顺序存储结构,帮助学生掌握栈操作的程序设计方法。

(2) 重点掌握如何使用栈来解决相关的应用问题。

2. 实验内容

(1) 用顺序栈的基本操作,实现数值的十进制和二进制转换。

(2) (选做)用顺序栈的基本操作,实现数值的十进制和十六进制转换。

(3) (选做)用链栈的基本操作,实现数值的十进制和二进制转换。

3. 算法设计

顺序栈的定义:

```
typedef int SElemType;
typedef struct
{
   SElemType * base;                          //栈底指针
   SElemType * top;                           //栈顶指针
   int stacksize;                             //栈可用的最大容量
} SqStack;
```

用到的栈基本操作有栈的初始化、出栈、入栈。

```
InitStack(SqStack &S);
Pop(SqStack &S, SElemType &e);
Push(SqStack &S, SElemType e);
```

4. 程序实现

程序主函数实现代码如下:

```
//顺序栈的实现
#include<iostream>
#include<fstream>
using namespace std;
//顺序栈的定义
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE 100
typedef int Status;
typedef int SElemType;
typedef struct
{
   SElemType * base;                          //栈底指针
```