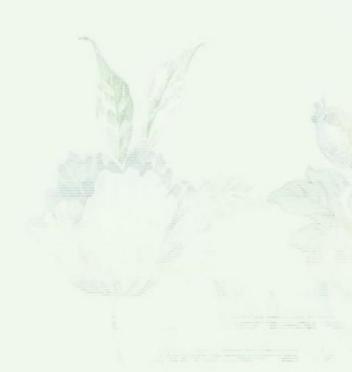
第二章 线性表





例、学生健康情况登记表如下:

姓名	学 号	性别	年龄	健康情况
王小林	790631	男	18	健康
陈红	790632	女	20	一般
刘建平	790633	男为	21	健康
张立立	790634	男	17	一般

系统分析

- (1)逻辑结构的分析
- (2) 系统操作(功能)的分析
- (3)存储结构
- (4) 系统实现



例、学生健康情况登记表如下:

姓名	学 号	性别	年龄	健康情况
王小林	790631	男	18	健康
陈红	790632	女	20	一般
刘建平	790633	男为	21	健康
张立立	790634	男	17	一般

线性表的逻辑结构及其基本操作

线性表是n(n>=0)个相同类型数据元素 a_0 , a_1 , ..., a_{n-1} 构成的有限序列。

形式化定义:

Linearlist = (D, R)

$$P : D = \{a_i \mid a_i \in D_0, i = 0, 1, \dots, n-1, n > 0\}$$

$$R = \{N\} \mid, N = \{\langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D_0, i = 1, 2, \dots, n-1\}$$

D₀为某个数据对象的集合 N为线性表长度

- 线性表的逻辑特征是:
- 〉 在非空的线性表,有且仅有一个开始结点a₁,它 没有直接前趋,而仅有一个直接后继a₂;
-)有且仅有一个终端结点a_n,它没有直接后继,而 仅有一个直接前趋a_{n-1};
- 其余的内部结点a_i(2≤i≤n-1)都有且仅有一个直接前趋a_{i-1}和一个直接后继a_{i+1}。
 线性表是一种典型的线性结构。
- · 数据的运算是定义在逻辑结构上的,而运算的 具体实现则是在存储结构上进行的。

系统操作(功能)的分析

系统操作(功能)的分析

- 1----新建学生健康表
- 2-----向学生健康表插入学生信息
- 3-----在健康表删除指定学生的信息
- 4-----为某个学生修改身体状况信息(按学号操作)
- 5-----接学生的学号排序并显示结果
- 6-----在健康表中查询学生信息(按学生学号来进行查找)
- 7-----在屏幕中输出全部学生信息
- 8-----从文件中读取所有学生健康表信息
- 9-----向文件写入所有学生健康表信息
- 10----退出

线性表抽象数据类型

数据元素: a;同属于一个数据元素类,i=1,2,.....,n n≧0。

结构关系:对所有的数据元素 a_i (i=1,2,....,n-1) 存在次序关系 < a_i , a_{i+1} >,

a₁无前驱,a_n无后继。

基本操作: 对线性表可执行以下的基本操作

Initiate(L) 构造一个空的线性表L。

Length(L) 求长度。

Empty(L) 判空表。

Full(L) 判表满。

Clear(L) 请空操作。

Get(L,i) 取元素。

Locate(L,x) 定位操作。

Prior(L,data) 求前驱。

link(L,data) 求后继。

Insert(L,i,b) 插入操作(前插)。

Delete(L,i) 删除操作。

线性表的抽象类

```
template <class T >
class LinearList {
public:
 LinearList();
                                    //构造函数
 ~LinearList();
                              //析构函数
  virtual int Size() const = 0; // 求表最大体积
  virtual int Length() const = 0;
                                   //求表长度
  virtual int Search(T x) const = 0;
                                       //搜索
  virtual int Locate(int i) const = 0;
                                       //定位
  virtual T^* getData(int i) const = 0;
                                        //取值
  virtual void setData(int i, T x) = 0;
                                        //赋值
```

```
virtual bool Insert(int i, T x) = 0;
                                         //插入
virtual bool Remove(int i, T& x) = 0;
                                         //删除
virtual bool IsEmpty() const = 0;
                                         //判表空
virtual bool IsFull() const = 0;
                                         //判表满
virtual void Sort() = 0;
                                         //排序
virtual void input() = 0;
                                         //输入
virtual void output() = 0;
                                         //输出
virtual LinearList<T>operator=
         (LinearList<T>& L) = 0; //复制
```

};

线性表在计算机中的实现

• 1.存储结构

• 2. 系统操作(功能)的实现

线性表的顺序存储结构

• 顺序存储

定义:把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。用这种方法存储的线

性表简称顺序表。

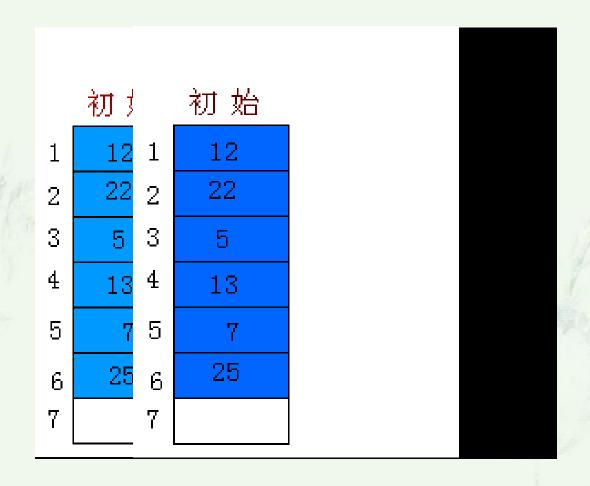
0	a_1
1	a. ₂
	•••
i-1	ai
i	a _{i+1}
n-1	an
maxlen-1	

存储结构的可行性分析

- · 系统操作(功能)实现过程的模拟
 - 插入
 - 一删除
 - 查找
 - 排序
 - ...

实现过程的模拟

• 趙除操作



实现过程的模拟

• 查找操作

```
i=1
3 12 19 7 36 5
2
r[0].key
```

实现过程的模拟

• 排序操作

```
第一步:
    将第一、第二个元素排序
    0 1 2 3 4 5 6 7 <u>8</u>
    i=2 <u>49 38</u> 65 97 76 13 27 49
```

线性表在计算机中的实现

• 1.存储结构

• 2. 系统操作(功能)的实现

顺序存储类

- 存储空间: 用数组实现
- 属性:

操作:

0	a_1
1	a ₂
i-1	ai
i	a _{i+1}
n-1	a _n
maxlen-1	



```
#define maxSize 100
                                           data[0..maxSize-1]
//const int maxSize=最大容量;
                                                   a1
Typedef int T;
typedef struct {
  T data[maxSize];//顺序表的静态存储表示
                                                  a2
 int n; //int last;
} SeqList;
                                        n
                                                             maxSize
typedef int T;
typedef struct {
 T *data;
                   //顺序表的动态存储表示
                                                  an
  int maxSize:
  int n; // int last;
} SeqList;
```

SeqList La,Lb; La.data[0]表示线性表的第一个元素, La.n 则表示线性表的当前元素个数

• 顺序表上实现的基本操作

在顺序表存储结构中,很容易实现线性表的一些操作,如线性表的构造、第i个元素的访问。

注意: C语言中的数组下标从"O"开始,因此,若La是SeqList类型的顺序表,则表中第i个元素是La.data[i-1]。

顺序表(SeqList)类的定义

```
public:
  SeqList(int sz = defaultSize);
                              //构造函数
  SeqList(SeqList<T>& L);
                              //复制构造函数
  ~SeqList() {delete[] data;}
                              //析构函数
  int Size() const {return maxSize;} // 求表最大容量
  int Length() const {return last+1;}
                                  //计算表长度
  int Search(T& x) const;
     //搜索X在表中位置。 函数返回表项序号
  int Locate(int i) const;
    //定位第i个表项。函数返回表项序号
  bool getData(int i, T & x); //取第i个元素
```

//插入

//删除

• • • • •

bool Insert(int i, T &x);

bool Remove(int i, T& x);

};

顺序表的构造函数

```
template <class T>
SeqList<T>::SeqList(int sz) {
  if (sz > 0) {
     maxSize = sz; last = -1;
     data = new T[maxSize];
                             //创建表存储数组
     if (data == 0)
                             // 动态分配失败
       { cerr << "存储分配错误! " << endl;
         exit(1); }
```

复制构造函数

```
template <class T>
SeqList<T>::SeqList (SeqList<T>& L) {
  T value;
  maxSize = L.Size(); last = L.Length()-1;
  data = new T[maxSize]; //创建存储数组
              //动态分配失败
  if (data == 0)
    {cerr << "存储分配错误! " << endl; exit(1);}
  for (int i = 1; i <= last+1; i++) //传送各个表项
  { L.getData(i,value); data[i-1] = value; }
```

查找

(1)按序号查找

(2) 按值查找

按序号查找

bool SeqList<T>:: getData(int i, T & x); // 1≦i≦n //取第i介元素

0	a ₁
1	a ₂
	•••
i-1	ai
i	a _{i+1}
n-1	an
maxlen-1	

按值查找

 a_1 a_2 i-1 a_{i} a_{i+1} n-1an maxlen-1

顺序表的按值查找算法

```
template <class T>
int SeqList<T>::search(T& x) const {
//在表中顺序搜索与给定值 X 匹配的表项.
//函数返回该表项是第几个元素, 否则函数返回()
  for (int i = 1; i <= last+1; i++)
                                 //顺序搜索
    if (data[i-1] == x) return i;
                //表项序号和表项位置差1
  return 0;
                //搜索失败
```

查找算法的分析





插入

线性表的插入运算是指在表的第i(1≦i≦n+1个位置上,插入一个新结点el,使长度为n的线性表

(a₁, ...a_{i-1}, a_i, ..., a_n) 变成长度为n+1的线性表

 $(a_1, ..., a_{i-1}, el, a_i, ..., a_n)$

单元编号(序号) 0 1	内容 a ₁ a ₂	单元编号(序号) 0 1	内容 a ₁ a ₂	
•••	•••	•••	•••	
i-1	a_{i}	i-1	el	←插入位置
i	a _{i+1}	i	$a_{ m i}$	
		•••		
n-1	an	n-1	a _{n-1}	
		n	a,	
maxlen-1		maxlen-1		
	插入前		插入后	

说明:即在顺序表的第i-1个数据元素与第i个数据元素 之间插入一个新的元素。

插入算法的步骤是:

- (1)检查线性表是否还有剩余空间可以插入元素,若已满,则进行"溢出"的错误处理;
- (2)检查i是否满足条件1≤i≤n+1,若不满足,则进行"位置不合法"的错误处理;
- (3)将线性表的第i个元素以及其后面的所有 元素均向后移动一个位置,以便腾出第i个 空位置来存放新元素;
- (4)将新元素el填入第i个空位置上;
- (5)把线性表的长度增加1。

插入算法

```
template <class T>
bool SeqList<T>::Insert (int i, T& x) {
//将新元素x插入到表中第i(1 \le i \le n+1)个表项位
//置。函数返回插入成功的信息
  if (last == maxSize-1) return false;
                                     //表满
  if (i < 1 || i > Length()+1) return false;//参数i不合理
  for (int j = last+1; j >= i; j--)
                                    //依次后移
     data[i] = data[i-1];
  data[i-1] = x; //插入(第 i 表项在data[i-1]处)
                                 //插入成功
  last++; return true;
};
```

• 分析算法的复杂度

这里的问题规模是表的长度,设它的值为N。

删除

线性表的删除运算是指将表的第i ($1 \le i \le n$)结点删除,使长度为n的线性表: (a_1 , ...a_{i-1}, a_i , a_{i+1} ..., a_n) 变成长度为n-1的线性表(a_1 , ...a_{i-1}, a_{i+1} , ..., a_n)

单元编号(序号)	内容		单元编号(序号)	内容
0	a ₁		0	a ₁
1	a. ₂		1	a ₂
	•••		•••	•••
i-1	a_{i}	←删除元素	i-1	a _{i+1}
i	a _{i+1}		i	a _{i+2}
	•••		•••	
			n=2	an
n-1	a _n		•••	
			•••	
maxlen-1			maxlen-1	
	删除前			删除后

删除算法的步骤为:

- (1)检查i 是否满足条件1≤i≤n,若不满足,则进行"位置不合法"的错误处理;
- (2)将线性表中的第i 个元素后面的所有元素均向前移动一个位置;
- (3)把线性表的长度减少1。

删除算法

```
template <class T>
bool SeqList<T>::Remove (int i, T& x) {
//从表中删除第i(1 \le i \le n) 个表项,通过引用型参
//数 X 返回被删元素。函数返回删除成功信息
   if (last == -1) return false;
   if (i < 1 || i > last+1) return false; //参数i不合理
   x = data[i-1];
   for (int j = i; j <= last; j++)
                              //依次前移。
     data[j-1] = data[j];
   last--; return true;
};
```





顺序表类的应用

P47

利用类的成员函数来解决问题

```
public:
  SeqList(int sz = defaultSize);
                               //构造函数
  SeqList(SeqList<T>& L);
                               //复制构造函数
  ~SeqList() {delete[] data;}
                               //析构函数
  int Size() const {return maxSize;} // 求表最大容量
  int Length() const {return last+1;}
                                   //计算表长度
  int Search(T& x) const;
     //搜索X在表中位置。 函数返回表项序号
  int Locate(int i) const;
     //定位第i个表项。函数返回表项序号
  bool getData(int i, T & x); //取第i个元素
  bool Insert(int i, T &x);
                                    //插入
  bool Remove(int i, T& x);
                               //删除
```

};

顺序表的应用: 集合的"并"运算



顺序表的应用: 集合的"并"运算

```
void Union (SeqList<int>& LA,
                SeqList<int>& LB ) {
  int n1 = LA.Length (), n2 = LB.Length ();
  int i, k, x;
  for (i = 1; i \le n2; i++)
    LB.getData(i,x);
                         //在LB中取一元素
    k = LA.Search(x);
                         //在LA中搜索它
     if (k == 0) // 若在LA中未找到插入它
      { LA.Insert(n1+1, x); n1++; }
                   //插入到第n个表项位置}
```

顺序表的应用: 集合的"交"运算





顺序表的应用: 集合的"交"运算

```
void Intersection (SeqList<int> & LA,
                     SeqList<int> & LB ) {
 int n1 = LA.Length ();
 int x, k, i = 1;
 while ( i \le n1 ) {
    LA.getData(i,x);
                     //在LA中取一元素
                       //在LB中搜索它
    k = LB.Search(x);
    if (k == 0)
                       //若在LB中未找到
      {LA.Remove(i, x); n1--;} //在LA中删除它
                       //未找到在A中删除它
    else i++;
```

完整系统的运行逻辑



顺序表的优点:

✓ 无须为表示节点间的逻辑关系而增加额外的存储空间

✓可以方便的随机存取表中的任一节点

顺序表的缺点

✓插入和删除运算不方便

✓由于要求占用连续的存储空间,存储分配只能预先进行

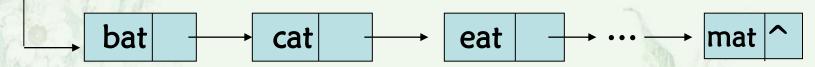
例、学生健康情况登记表如下:

姓名	学 号	性别	年龄	健康情况
王小林	790631	男	18	健康
陈红	790632	女	20	一般
刘建平	790633	男	21	健康
张立立	790634	男	17	一般

线性表的链式存储和实现

线性表:(bat, cat, eat, fat, hat, jat, lat, mat)





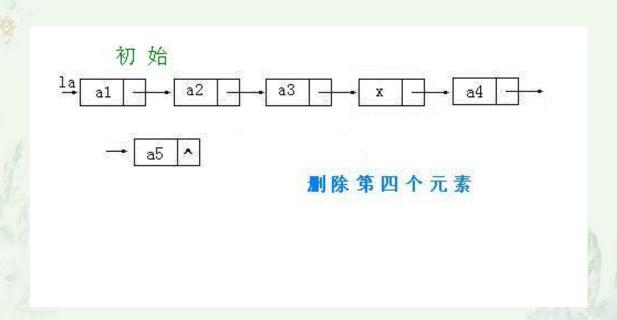
链表的可行性分析

- · 系统操作(功能)实现过程的模拟
 - 插入
 - 一删除
 - 查找
 - 排序

- ...

实现过程的模拟

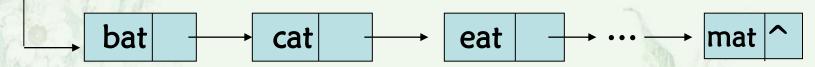
• 賴除操作



线性表的链式存储和实现

线性表:(bat, cat, eat, fat, hat, jat, lat, mat)





链式结构中结点结构的抽象

data

link

链表是指用一组任意的存储单元来依次存放线性表的结点,这组存储单元即可以是连续的,也可以是不连续的,甚至是随机分布在内存中的任意位置上的。

即链表中结点的逻辑次序和物理次序不一定相同。

C++语言实现链表结点的结构

data

link

```
struct LinkNode { //链表结点类 int data; // T data; LinkNode * link; };
```

单链表 (Singly Linked List)

- 线性结构
 - 结点之间可以连续, 可以不连续存储
 - ◆结点的逻辑顺序与物理顺序可以不一致
 - ◆ 表可扩充

$$\mathbf{first} \longrightarrow a_1 \longrightarrow a_2 \longrightarrow a_3 \longrightarrow a_4 \longrightarrow a_5 \wedge$$

单链表类的定义

- 属性: 反映链表的重要特性(即链头)
- •操作:线性表操作的抽象及综合(查找、插入、删除等)

```
class List {
```

//链表类, 直接使用链表结点类的数据和操作 private:

```
LinkNode *first; //表头指针
```

public:

. . . .

};

与链表有关的基本操作

- (1)结点指针的定义 LinkNode<T> *p;
- (2)结点存储空间的申请 new 方法

p=new LinkNode <T>();

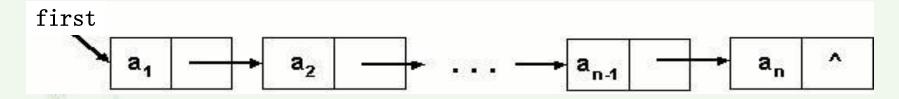
(3)空间的释放 delete方法

delete p

(4)结点中域的访问 p->data=x; p->link=q; (q是结点指针)

(5) 全指针 NULL或0 p->link=0; q=0;

单链表的建立



1、头插入建表

该方法从一个空表开始,重复读入数据,生成新结点,将读入数据存放到新结点的数据城中,然后将新结点插入到当前链表的表头上,直到读入结束为止。

```
template < class T>
List<T>:: HLinkList (int n)
  first=0;
  for (i=0;i< n;i++)
    p=new LinkNode <T>();
     cin>>p->data;
     p->link=first;
     first=p;
```

单链表的建立

2、尾插入建表

头插法建立链表虽然算法简单,但生成的链表中结点的次序和输入的顺序相反。若希望二者次序一致,可采用尾插法建表。

该方法是将新结点插入到当前链表的表尾上,为此必须增加一个尾指针r,使其始终指向当前链表的尾结点。

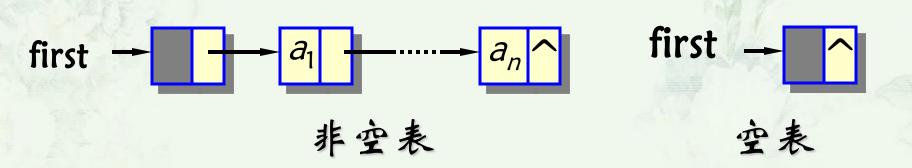
```
template < class T>
List<T>:: RLinkList (int n )
   first=0;tail=0;
   for(i=0;i< n;i++)
  { p=new LinkNode <T>();
     cin>>p->data;
     p->link=0;
     if(first = 0)
        first=p;
                              该算法的缺陷是不如头插
     else
                              入算法那么简洁
        tail->link=p;
     tail=p;
```

尾插入建链改进算法

```
template < class T>
List<T>:: RLinkList (int n )
                                  引入表头结点
 first=tail=new LinkNode <T>(); first->link=0;
 for(i=0;i<n;i++)
 \{ p=\text{new Node}<T>(); cin>>p->data; \}
    p->link=0;
    tail->link=p;
    tail=p;
      first _
                                                ٨
```

带表头结点的单链表

- 表头结点位于表的最前端,本身不带数据,仅标志表头。
- 设置表头结点的目的是
 - 统一空表与非空表的操作
 - * 简化链表操作的实现。



单链表的模板类

- · 类模板将类的数据成员和成员函数设计得更完整、更灵活。
- . 类模板更易于复用。
- · 在单链表的类模板定义中,增加了表头结点。

用模板定义的结点类

```
template <class T>
struct LinkNode {
                         //链表结点类的定义
                         //数据域
  T data;
  LinkNode<T> * link; //链指针域
  LinkNode() { link = 0; } //构造函数
  LinkNode(T item, LinkNode<T> * ptr = 0)
     { data = item; link = ptr; } //构造函数
  bool operator== (T x) { return data.key == x; }
     //重载函数. 判相等
  bool operator != (T x) { return data.key != x; }
};
```

用模板定义的单链表类

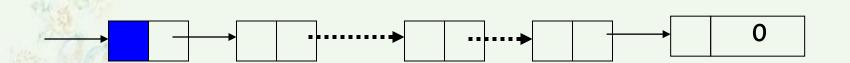
```
template <class T>
class List {
//单链表类定义, 不用继承也可实现
                                    P60
protected:
  LinkNode<T> * first; //表头指针
public:
  List() { first = new LinkNode<T>; } //构造函数
  List(T x) { first = new LinkNode < T > (x); }
  List(List<T>&L);
                          //复制构造函数
  ~List(){ }
                          //析构函数
  void makeEmpty();
                          //将链表置为空表
  int Length() const;
                          //计算链表的长度
```

```
LinkNode<T>*Search(Tx);//搜索含x元素
LinkNode<T> *Locate(int i);
                               //定位第i个元素
T *getData(int i);
                               //取出第i元素值
void setData(int i, T x);
                               //更新第i元素值
bool Insert (int i, T x);
                           //在第i元素后插入
bool Remove(int i, T& x);
                           //删除第i个元素
bool IsEmpty() const
                           //判表空否
 { return first->link == 0 ? true : false; }
LinkNode<T> *getFirst() const { return first; }
void setFirst(LinkNode<T> *f ) { first = f; }
void Sort();
                           //排序
void Print(); //输出整条链表的结点值
```

输出整条链表的结点值——遍历

void print();

功能:输出链表中的所有结点值。



输出单链表所有结点值的算法

```
template <class T>
Void List<T>:: Print ( )
  LinkNode<T>*p = first->link;
  //检测指针 p 指示第1个结点
  while (p!=0)
      //逐个结点检测
     cout < p-> data; p = p-> link;
```

求长度操作

int length();

功能:求单链表的长度。

