

**数据结构实验报告（二）**

实验课题：单链表的基本操作

学 院：理学院

专 业：数学系

组 员：190110304王君涛

190110312战祥贤

190110318王 鹏

190110302王 琪

190110331陈 立

组 长：王君涛

2020年10月7日

**链表操作**

所有功能：

**具体实现（实现过程以姓名表为实例）：**

1. **创建链表并初始化**

**（首先定义结构体作为链表的节点包含指针域与数据域）**

1. **前插法建立链表初始化：**

具体代码：

void createList\_front(List\*& L,int n)

{

L = new List;

L->next = NULL;

cout<<”请输入元素：”;

for (int i = 0; i < n; i++)

{

List\* p = new List;

cin >> p->name;

p->next = L->next;

L->next = p;

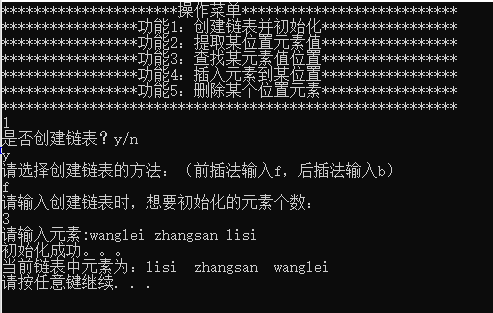
}

cout << "初始化成功。。。" << endl;

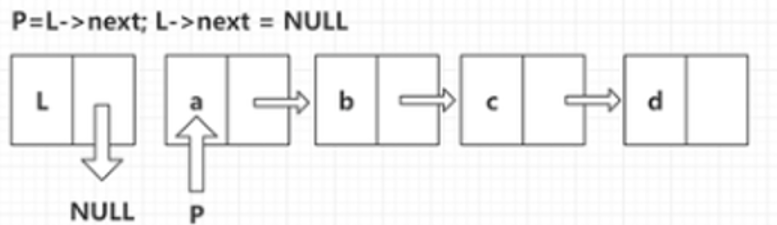
}

说明：（1）首先创建一个节点作为头节点（l=new list）将l->next指向空至此一个空链表创建成功。（2）然后开始插入这里采用前插：（p作为中间指针为节点的插入服务）将p指向新创建的节点（List\* p = new List;）输入新节点数据域的值（cin >> p->name;）新插入的节点的指针域指向上一个插入的节点。（p->next = L->next;插入第一个元素时指空）将头节点的指针指向新创建的节点（L->next = p;）

运行结果



图示：



1. 后插法建立有头节点的链表

具体代码：

void createList\_back(List\*& L, int n)

{

L = new List;

List\* R = new List;

L->next = NULL;

R= L;

cout<<”请输入元素：”;

for (int i = 0; i < n; i++)

{

List\* p = new List;

cin >> p->name;

R->next = p;

p->next = NULL;

R = p;

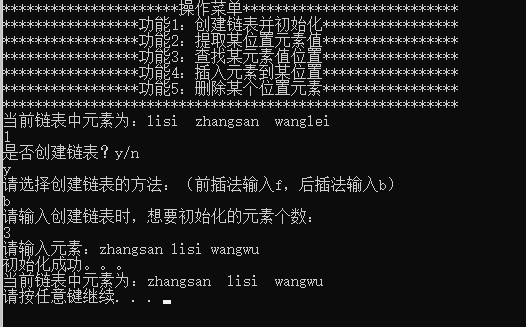
}

cout << "初始化成功。。。" << endl;

}

说明：（1）首先创建一个节点作为头节点（l=new list）将l->next指向空，至此创建空链表成功。（2）然后这里采用后：要先定义一个指针R（List\* R = new List;）让R指向表尾。（p作为中间指针为节点的插入服务）将p指向新创建的节点（List\* p = new List;）输入新节点数据域的值（cin >> p->name;）将上一个节点的指针指向新创建的节点。（R->next = p;）新插入的节点在队尾要指向空（p->next = NULL;）最后要保持R永远指向队尾（R = p;）

运行结果



通过一个函数将以上两种方法包装

void createList(List\*& L)//将前插法与后插法合并，创建链表

{

char j,k;

cout << "是否创建链表？y/n" << endl;

cin >> j;

if (j == 'y')

{

int n;

cout << "请选择创建链表的方法：（前插法输入f，后插法输入b）" << endl;

cin >> k;

if (k == 'f')

{

cout << "请输入创建链表时，想要初始化的元素个数：" << endl;

cin >> n;

createList\_front(L, n);

}

else if (k == 'b')

{

cout << "请输入创建链表时，想要初始化的元素个数：" << endl;

cin >> n;

createList\_back(L, n);

}

}

else

{

cout << "程序退出。" << endl;

}

}

与顺序表的对比：

这两种时间复杂度都为o（n）链表创建时可以根据所需的表长申请存储空间，不必像顺序表一样提前申请造成空间的浪费。

1. **提取某位置元素值**

具体代码：

string getElem(List\*& L, int n)//得到第n个位置上的元素值

string getElem(List\*& L, int n)

{

List\* p = new List;

int i;

p = L;

for (i = 0; i < n; i++)

{

p = p->next;

}

if (!p || i > n)

{

cout << "运行错误" << endl;

return "error\_opreate";

}

cout << "第" << n << "个位置的元素为：" << p->name << endl;

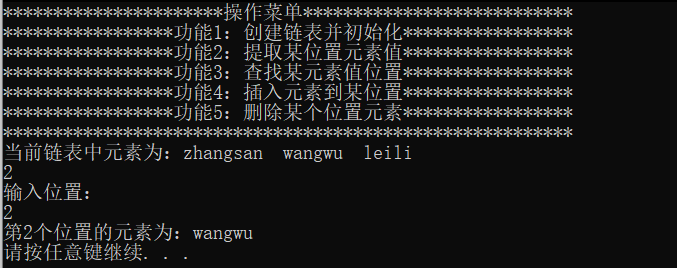
return p->name;

}

说明：（p作为中间指针为节点的查找服务）将p指向新节点L

从键盘输入查找元素的顺序位置通过for循环将指针p后移直到输入的位置为止。通过if (!p || i > n)判断是否合理。合理则将该位置的数据返回主函数输出。

运行结果



与顺序表对比

顺序表由于物理位置的连续性只需要O（1）的时间复杂度便可以直接找到要查找的元素位置

1. **查找某元素值位置**

**具体代码：**

void locateElem(List\*& L,string e) {

List\* p = new List;

p = L->next;

int num=1;

while (p)

{

if (p->name == e)

{

cout << "第" << num << "个位置上的元素值等于" << e << "，地址值为：" << p << endl;;

}

num++;

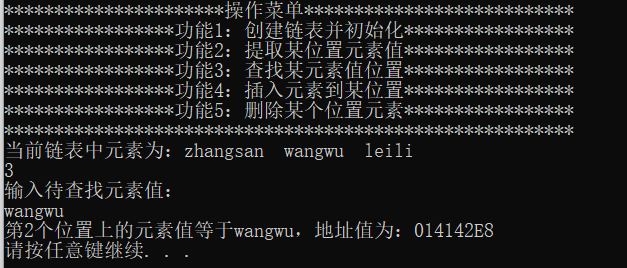
p = p->next;

}

}

说明：（p作为中间指针为节点的查找服务）从键盘输入你要找的元素的具体值e。p指向头节点通过while（p）循环找到元素为值e的节点位置。

运行结果



1. **插入元素到某位置**

具体代码：

void insertElem(List\*& L, int i, string e) {

List\* p = L;

int j = 0;

while (p && (j < i - 1))

{

p = p->next;

j++;

}

if (!p || j > i - 1)

{

cout << "错误！" << endl;

return;

}

List\* s = new List;

s->name = e;

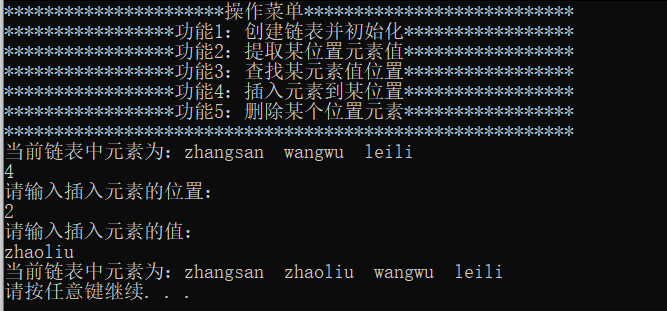
s->next = p->next;

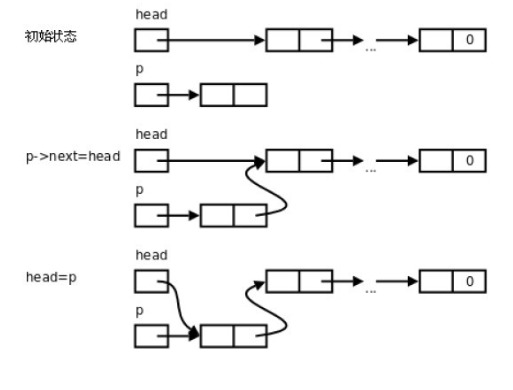
p->next = s;

}

说明：（p作为中间指针为节点的插入服务）p指向头节点，（List\* p = L;）从键盘输入要插入的元素值e以及要插入的位置i通过while循环（while (p && (j < i - 1)){p = p->next;j++;}））找到要插入的位置并用p标记。创造一个节点s将输入的值e赋给s的数据域（s->name = e;）。s的指针通过p指向下一个节点（s->next = p->next;）.。s的前一个结点通过p指向s（ p->next = s;）至此完成节点的插入。

运行结果





1. 删除某个位置元素

具体代码：

void deleteElem(List\*& L,int n)

{

List\* p = new List;

p = L;

for (int i = 0; i < n-1; i++)

{

p = p->next;

if (!p)

{

cout << "error\_operate"<<endl;

return;

}

}

if (!p->next)

{

cout << "error\_operate"<<endl;

return;

}

List\* q = new List;

q = p->next;

p->next = q->next;

cout << "successfully deleted" << endl;

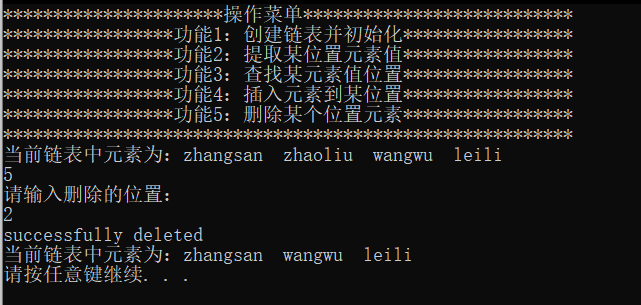
delete q;

}

说明：

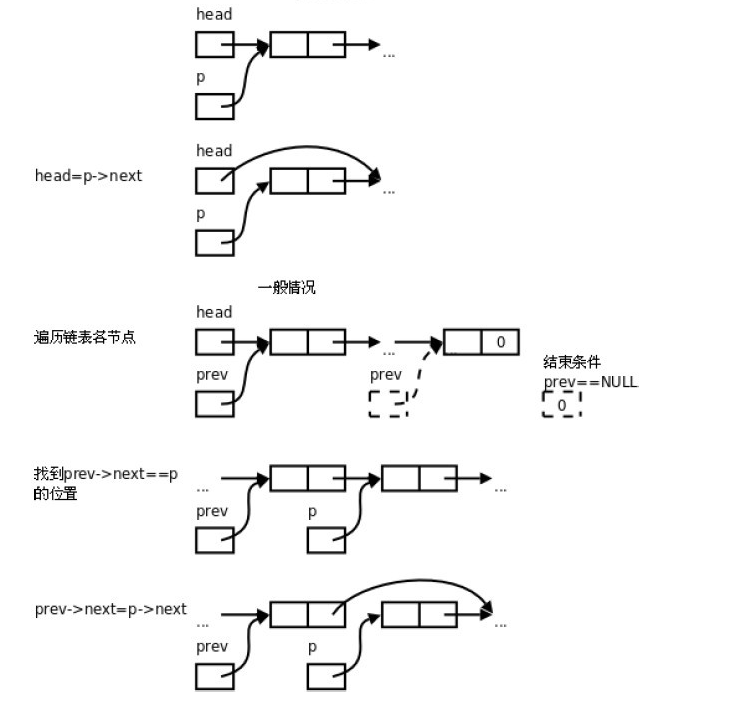
（p作为中间指针为节点的删除服务）从键盘输入要删除的节点n。P指向头节点，通过for循环找到要删除的前一个节点（for (int i = 0; i < n-1; i++)）。将q指向要删除的节点（q = p->next;）。将p的指针指向p的后一个节点（p->next = q->next;） 释放q的空间(delete q;）至此完成节点的删除。

运行结果



与顺序表相比

链式表不能直接找到元素的地址而要通过依次查找因此要更费时。



* **主程序**
* #include<iostream>
* using namespace std;
* #include<string>
* struct List
* {
* List\* next;
* string name;
* };
* //所有的操作函数如下：
* /\*
* void createList\_front(Lis\*& L,int n)//用前插法建立有头节点的链表，并初始化n个元素
* void createList\_back(List\*& L, int n)//用后插法建立有头节点的链表，并初始化n个元素
* void createList(List\*& L)//将前插法与后插法合并，创建链表
* string getElem(List\*& L, int n)//得到第n个位置上的元素值
* void locateElem(List\*& L,string e)//查找出链表中所有元素值为e的元素位置
* void insertElem(List\*& L, int i, string e)//在链表的第i个位置插入新元素，值为e
* void showList(List\*& L)//显示当前链表
* void deleteElem(List\*& L,int n)//删除第n个位置上的元素
* void showMenu()//菜单
* \*/
* void createList\_front(List\*& L,int n)//用前插法建立有头节点的链表
* {
* L = new List;
* L->next = NULL;
* for (int i = 0; i < n; i++)
* {
* List\* p = new List;
* cin >> p->name;
* p->next = L->next;
* L->next = p;
* }
* cout << "初始化成功。。。" << endl;
* }
* void createList\_back(List\*& L, int n)//用后插法建立有头节点的链表
* {
* L = new List;
* List\* R = new List;
* L->next = NULL;
* R= L;
* for (int i = 0; i < n; i++)
* {
* List\* p = new List;
* cin >> p->name;
* R->next = p;
* p->next = NULL;
* R = p;
* }
* cout << "初始化成功。。。" << endl;
* }
* void createList(List\*& L)//将前插法与后插法合并，创建链表
* {
* char j,k;
* cout << "是否创建链表？y/n" << endl;
* cin >> j;
* if (j == 'y')
* {
* int n;
* cout << "请选择创建链表的方法：（前插法输入f，后插法输入b）" << endl;
* cin >> k;
* if (k == 'f')
* {
* cout << "请输入创建链表时，想要初始化的元素个数：" << endl;
* cin >> n;
* cout << "请输入元素：" << endl;
* createList\_front(L, n);
* }
* else if (k == 'b')
* {
* cout << "请输入创建链表时，想要初始化的元素个数：" << endl;
* cin >> n;
* cout << "请输入元素：" << endl;
* createList\_back(L, n);
* }
* }
* else
* {
* cout << "程序退出。" << endl;
* }
* }
* string getElem(List\*& L, int n)
* {
* List\* p = new List;
* int i;
* p = L;
* for (i = 0; i < n; i++)
* {
* p = p->next;
* }
* if (!p || i > n)
* {
* cout << "运行错误" << endl;
* return "error\_opreate";
* }
* cout << "第" << n << "个位置的元素为：" << p->name << endl;
* return p->name;
* }
* void locateElem(List\*& L,string e)
* {
* List\* p = new List;
* p = L->next;
* int num=1;
* while (p)
* {
* if (p->name == e)
* {
* cout << "第" << num << "个位置上的元素值等于" << e << "，地址值为：" << p << endl;;
* }
* num++;
* p = p->next;
* }
* }
* void insertElem(List\*& L, int i, string e)
* {
* List\* p = L;
* int j = 0;
* while (p && (j < i - 1))
* {
* p = p->next;
* j++;
* }
* if (!p || j > i - 1)
* {
* cout << "错误！" << endl;
* return;
* }
* List\* s = new List;
* s->name = e;
* s->next = p->next;
* p->next = s;
* }
* void showList(List\*& L)//显示当前链表
* {
* List\* p = new List;
* p = L->next;
* cout << "当前链表中元素为：";
* while (p)
* {
* cout << p->name << " ";
* p = p->next;
* }
* cout << endl;
* }
* void deleteElem(List\*& L,int n)
* {
* List\* p = new List;
* p = L;
* for (int i = 0; i < n-1; i++)
* {
* p = p->next;
* if (!p)
* {
* cout << "error\_operate"<<endl;
* return;
* }
* }
* if (!p->next)
* {
* cout << "error\_operate"<<endl;
* return;
* }
* List\* q = new List;
* q = p->next;
* p->next = q->next;
* cout << "successfully deleted" << endl;
* delete q;
* }
* void showMenu()
* {
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*操作菜单\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能1：创建链表并初始化\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能2：提取某位置元素值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能3：查找某元素值位置\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能4：插入元素到某位置\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能5：删除某个位置元素\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能0：退出链表结束程序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
* }
* int main()
* {
* List\* L;
* int a, b, c;
* string e,g;
* ///////////////////////////////
* int Choice = 0;
* bool flag=false;
* while (true)
* {
* system("cls");
* showMenu();
* if(flag==true)
* showList(L);
* cout << "请选择功能：" << endl;
* while (true)
* {
* cin >> Choice;
* if (Choice >= 0 && Choice <= 5)
* {
* break;
* }
* else
* {
* cout << "输入错误，程序退出" << endl;
* Choice = 0;
* return 0;
* }
* }
* switch (Choice)
* {
* case 1://功能1：创建链表并初始化
* createList(L);
* showList(L);
* flag = true;
* system("pause");
* break;
* case 2://功能2：得到某位置上的元素值
* cout << "输入位置：" << endl;
* cin >> a;
* getElem(L, a);
* system("pause");
* break;
* case 3://功能3：查找某元素值在链表中位置
* cout << "输入待查找元素值：" << endl;
* cin >> e;
* locateElem(L, e);
* system("pause");
* break;
* case 4://功能4：在链表某位置插入元素
* cout << "请输入插入元素的位置：" << endl;
* cin >> b;
* cout << "请输入插入元素的值：" << endl;
* cin >> g;
* insertElem(L, b, g);
* showList(L);
* system("pause");
* break;
* case 5://功能5：删除链表中某位置元素
* cout << "请输入删除的位置：" << endl;
* cin >> c;
* deleteElem(L, c);
* showList(L);
* system("pause");
* break;
* case 0://功能6：退出系统
* cout << "欢迎下次使用！" << endl;
* system("pause");
* return 0;
* break;
* }
* }
* system("pause");
* return 0;
* }

**总结：**

（一）顺序表存储时必须预先分配空间，为了不至于使空间溢出，分配时尽量使空间大于所需空间这就造成空间的浪费，因此当线性表的长度变化较大时难以预估储存规模时，尽量采用链式表。

从存储密度上如果不考虑顺序表的空闲区，顺序表的存储密度为1，单链表的存储密度为0.5。

（二）随机存储时顺序表的存储效率要高于单链表。基于此若线性表的主要操作是和元素位置紧密相连的宜采用顺序表。

插入删除时单链表只需要修改指针即可，而顺序表要进行移动操作，因此对于频繁进行插入删除操作时，采用链表更加合适。