《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 计科大类 | 姓名 |  | 学号 |  |
| 实验布置日期 | | 2019.10.25 | | 提交  日期 | 2019.11.15 | | 成绩 |  |

课程实践实验五：线性表的实现和应用

## 问题描述及要求

（1）实现顺序表类及基本操作并进行测试。

（2）实现单链表类及基本操作并进行测试。除基本操作外，为单链表类增加逆置、按值查找、按值统计、删除等（递归及非递归）算法并进行测试。

（3）设计并实现有序单链表类，添加有序表下的按值插入、有序表合并等算法。

（4）利用单链表类，完成日常支出简单管理。

输入n项支出项目(可以选择从文件输入)，并依次输出所有支出项目。

求出这n个支出项目中的最小、最大和平均消费。

按照日期找出某一天的所有花费。

按照日期和项目找出该项目花费 。

按照项目找出该项目的所有花费。比如要求给出在shoes这一项上总共花了多少钱。

## 二、概要设计

**（1）对实验内容的理解和二次概括**

**使用线性表和链表的定义来实现基本的操作，并通过定义节点的数据结构，来实现日常支出管理。**

**（2）给出系统的功能列表**

* **顺序表类的基本操作：增，减，清除等**
* **单链表的基本操作**
* **单链表的逆置，按值查找，按值统计，删除等**
* **有序表下的按值插入，有序表合并等算法**
* **日常支出的简单管理**

**（3）程序运行的界面设计 （可以用图示等方式，如：首先出现屏幕提示，请用户选择输入配置的方式，1从键盘输入活单元坐标2……3….然后用户）**

**（4）确定总体设计思路，采用何种数据结构，设计哪些类，各类的作用 ，类方法的介绍，类之间的关系描述**

**采用顺序表和单链表的数据结构，设计一个顺序表类，一个单链表类，再针对日常支出的简单管理设计一个单链表类，每个类有基本的操作方法：size, empty, clear, full, reverse, retrieve, replace, remove, insert**

## 三、详细设计

单链表的逆置：void reverse();

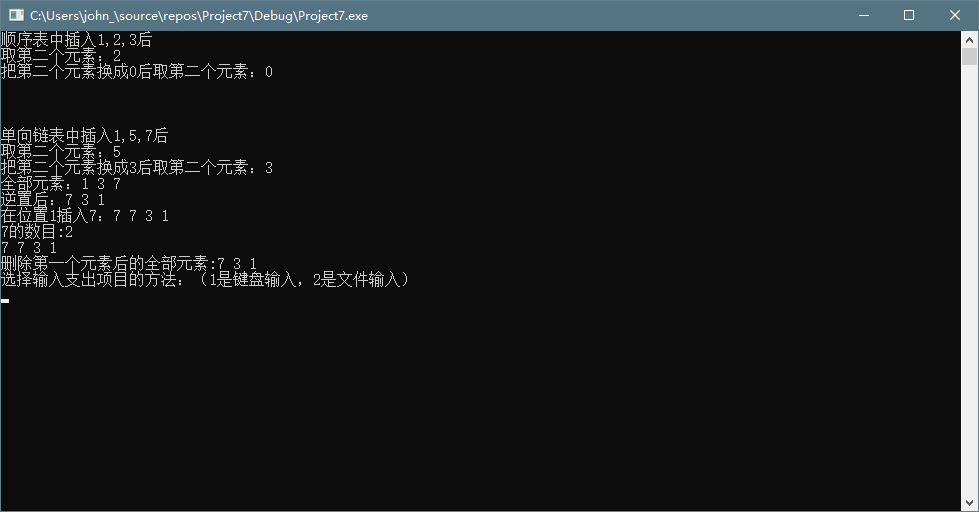
从头结点开始，使指针p指向头节点，再使第一个头结点的next指向第三个节点，第二个节点的next指向头结点，这样就完成了前两个节点的顺序交换，然后让head指向第二个节点，继续循环，依次交换。

单链表的按值查找：Error\_code list<list\_entry>::search(const list\_entry&x, int &position)

从单链表的头节点开始，依次向后，使用position记录位置，如果出现某一个节点的值与查找的值相等，则记录查找到的position位置。

## 四、实验结果

紧扣题目要求设计提供相应的测试方法和结果。可以给出具体的测试用例，每个测试用例一般可列出：

* 测试输入：无
* 测试目的：检查基本功能是否出错
* 实际输出：
* 
* 测试结论：通过

## 五、实验分析与探讨

实验中主要碰到的问题是基本功能实现的过程中怎样正确返回错误代码，怎样完善程序，以及在应用顺序表和链表的基本功能时怎样正确定义结点的数据结构，如何应变。

## 六、小结

本次实验基本完成了顺序表和链表的基本功能的实现，我熟悉了线性表的定义脚本编程方法，和对顺序表和链表的应用。

## 附录：源代码

#include<iostream>

#include<string>

#include<fstream>

#include<vector>

using namespace std;

enum Error\_code {

success, fail, rangeerror, underflow, overflow, fatal, not\_present,

duplicate\_error, entry\_inserted, entry\_found, internal\_error

};

const int max\_list = 10;

template<class List\_entry>

class List

{

public:

List()

{

count = 0;

}

int size()const;

bool full()const;

bool empty()const;

void clear();

void traverse(void(\*visit)(List\_entry& x));

Error\_code retrieve(int position, List\_entry &x);

Error\_code replace(int position, const List\_entry &x);

Error\_code remove(int position, List\_entry &x);

Error\_code insert(int position, const List\_entry &x);

protected:

int count;

List\_entry entry[max\_list];

};

template<class List\_entry>

int List<List\_entry>::size()const

{

return count;

}

template<class List\_entry>

bool List<List\_entry>::full()const

{

if (count == max\_list)

return true;

else

return false;

}

template<class List\_entry>

bool List<List\_entry>::empty()const

{

if (count == 0)

return true;

else

return false;

}

template<class List\_entry>

Error\_code List<List\_entry>::remove(int position, List\_entry &x)

{

if (position < 0 || position >= count)

return fail;

if (empty())

return underflow;

else

{

x = entry[position];

for (int i = position; count - 1 >= i; i++)

{

entry[i] = entry[i + 1];

}

return success;

}

}

template<class List\_entry>

void List<List\_entry>::clear()

{

for (int i = count - 1; i >= 0; i--)

{

List\_entry x;

remove(i, x);

}

}

template<class List\_entry>

Error\_code List<List\_entry>::retrieve(int position, List\_entry &x)

{

if (position < 0 || position >= count)

return fail;

else

x = entry[position];

return success;

}

template<class List\_entry>

Error\_code List<List\_entry>::replace(int position, const List\_entry &x)

{

if (position < 0 || position >= count)

return fail;

else

entry[position] = x;

return success;

}

template <class List\_entry>

void List<List\_entry>::traverse(void(\*visit)(List\_entry& x))

{

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

{

(\*visit)(entry[i]);

}

}

template<class List\_entry>

Error\_code List<List\_entry>::insert(int position, const List\_entry &x)

{

if (full())

return overflow;

if (position < 0 || position > count)

return fail;

for (int i = count - 1; i >= position; i--)

{

entry[i + 1] = entry[i];

}

entry[position] = x;

count++;

return success;

}

template<class Node\_entry>

struct Node

{

Node\_entry entry;

Node<Node\_entry>\*next;

Node()

{

next = NULL;

}

Node(Node\_entry item, Node \*add\_on = NULL)

{

entry = item;

next = add\_on;

}

};

template<class list\_entry>

class list

{

public:

~list()

{

delete head;

}

list();

list(const list<list\_entry>&copy);

int size() const;//长度

bool empty() const;

void clear();

void traverse(void(\*visit)(list\_entry& x));

Error\_code retrieve(int position, list\_entry& x);

Error\_code replace(int position, const list\_entry& x);

Error\_code remove(int position, list\_entry& x);

Error\_code insert(int position, const list\_entry& x);

void reverse();//逆置

int tongji(const list\_entry& x);//统计

Error\_code search(const list\_entry& x);//按值查找

protected:

int count;

Node<list\_entry>\*head;

Node<list\_entry>\*set\_position(int position)const;

};

template<class list\_entry>

Node<list\_entry>\*list<list\_entry>::set\_position(int position)const

{

Node<list\_entry>\* q = head;

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

q = q->next;

return q;

}

template<class list\_entry>

list<list\_entry>::list(const list<list\_entry>&copy)

{

count = copy.count;

head = copy.head;

}

template<class list\_entry>

list<list\_entry>::list()

{

count = 0;

head = NULL;

}

template<class list\_entry>

int list<list\_entry>::size()const

{

return count;

}

template<class list\_entry>

bool list<list\_entry>::empty() const

{

if (count == 0)

return true;

else

return false;

}

template<class list\_entry>

void list<list\_entry>::clear()

{

while (head != NULL)

{

Node<list\_entry>\*old\_en = head;

head = head->next;

delete old\_en;

}

count = 0;

}

template <class list\_entry>

void list<list\_entry>::traverse(void(\*visit)(list\_entry& x))

{

Node<list\_entry>\* q = head;

while (q)

{

(\*visit)(q->entry);

q = q->next;

}

}

template <class list\_entry>

Error\_code list<list\_entry>::retrieve(int position, list\_entry& x)

{

if (position < 0 || position >= count)

return rangeerror;

else

{

Node<list\_entry>\*a = set\_position(position);

x = a->entry;

return success;

}

}

template <class list\_entry>

Error\_code list<list\_entry>::replace(int position, const list\_entry& x)

{

if (position < 0 || position >= count)

return rangeerror;

else

{

Node<list\_entry>\*a = set\_position(position);

a->entry = x;

return success;

}

}

template <class list\_entry>

Error\_code list<list\_entry>::remove(int position, list\_entry& x)//删除；

{

if (position < 0 || position >= count)

return rangeerror;

if (position == 0)

{

x = head->entry;

Node<list\_entry>\*a = head;

head = head->next;

delete a;

count--;

return success;

}

if (position > 0)

{

Node<list\_entry>\*a = set\_position(position - 1);

Node<list\_entry>\*old\_en = a->next;

x = old\_en->entry;

Node<list\_entry>\*b = old\_en->next;

a->next = b;

delete old\_en;

count--;

return success;

}

}

template <class list\_entry>

Error\_code list<list\_entry>::insert(int position, const list\_entry& x)

{

if (position < 0 || position > count)

return rangeerror;

Node<list\_entry>\*new\_node, \*previous, \*following;

if (position == 0)

{

following = head;

new\_node = new Node<list\_entry>(x, following);

head = new\_node;

count++;

}

if (position > 0)

{

previous = set\_position(position - 1);

following = previous->next;

new\_node = new Node<list\_entry>(x, following);

previous->next = new\_node;

count++;

}

}

template<class list\_entry>

void list<list\_entry>::reverse()

{

if (count > 1)

{

Node<list\_entry>\*p = head;

while (p->next != NULL)

{

Node<list\_entry>\*a = p->next;

p->next = a->next;

a->next = head;

head = a;

}

}

}

template<class list\_entry>

int list<list\_entry>::tongji(const list\_entry& x)

{

int number = 0;

Node<list\_entry>\*p = head;

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

{

if (p->entry == x)

{

number++;

}

p = p->next;

}

return number;

}

template<class list\_entry>

Error\_code list<list\_entry>::search(const list\_entry& x)

{

Node<list\_entry>\* p = head;

int number = 0;

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

{

if (p->entry == x)

{

number++;

return success;

}

p = p->next;

}

return fail;

}

template<class list\_entry>

class sx\_list :public list<list\_entry> //有序链表

{

public:

sx\_list();

Error\_code insert(const list\_entry&x);

void operator+(const sx\_list&a);

};

template<class list\_entry>

sx\_list<list\_entry>::sx\_list()

{

list();

}

template<class list\_entry>

Error\_code sx\_list<list\_entry>::insert(const list\_entry&x)

{

int position = 0;

Node<list\_entry>\*p = head;

while (p != NULL)

{

if (p->entry < x)

{

p = p->next;

position++;

}

else

{

return insert(position, x);

}

}

return insert(count, x);

}

template<class list\_entry>

void sx\_list<list\_entry>::operator+(const sx\_list&a)

{

Node<list\_entry>\*p = a.head;

while (p != NULL)

{

insert(p->entry);

p = p->next;

}

}

void c(int &a)

{

cout << a << " ";

}

struct node

{

string way;

string day;

double m;

node \*next;

node()

{

next = NULL;

}

node(string &w, string &d, double &m1, node \*add\_on = NULL)

{

way = w;

day = d;

m = m1;

next = add\_on;

}

};

class list1

{

private:

node \*head;

int count;

node \*set\_position(int position);

public:

void p1(int position);//对应下标的输出

double p2(string d);//对应日期

double p3(string xm);//输出对应项目的总数据

double p4(string d, string xm);//日期和项目

list1();

list1(const list1&copy);

Error\_code insert(int position, string&d, string&xm, double x);

double max();

double min();

double aver();

};

list1::list1()

{

count = 0;

head = NULL;

}

list1::list1(const list1&copy)

{

if (copy.count != 0)

{

count = copy.count;

node\* original, \*p;

original = copy.head;

string d = original->day;

string xm = original->way;

double x = original->m;

head = p = new node(d, xm, x);

while (original->next != NULL)

{

original = original->next;

d = original->day;

xm = original->way;

x = original->m;

p->next = new node(d, xm, x);

}

}

count = 0;

head = NULL;

}

double list1::max()

{

double max;

node\*p = head;

max = p->m;

while (p != NULL)

{

double a = p->m;

if (a > max)

max = a;

p = p->next;

}

return max;

}

double list1::min()

{

double min;

node \*p = head;

min = p->m;

while (p != NULL)

{

double a = p->m;

if (a < min)

min = a;

p = p->next;

}

return min;

}

node\* list1::set\_position(int position)

{

node\*p = head;

int i = 0;

while (i < position)

{

p = p->next;

i++;

}

count++;

return p;

}

double list1::aver()

{

vector<string> s;

node\*p = head;

while (p != NULL)

{

bool flag = true;

for (int i = 0; i < s.size(); i++)

{

if (p->way == s[i])

{

flag = false;

break;

}

}

if (flag == true)

s.push\_back(p->way);

p = p->next;

}

double all = 0;

int h = s.size();

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

{

all += p3(s[i]);

}

return all / h;

}

double list1::p3(string xm)

{

double x = 0;

node\*p = head;

while (p != NULL)

{

if (p->way == xm)

{

x += p->m;

}

p = p->next;

}

return x;

}

Error\_code list1::insert(int position, string&d, string&xm, double x)

{

if (position < 0 || position>count)

return rangeerror;

node \*new\_node, \*previous, \*following;

if (position == 0)

{

following = head;

new\_node = new node(d, xm, x, following);

head = new\_node;

}

else

{

previous = set\_position(position - 1);

following = previous->next;

new\_node = new node(d, xm, x, following);

previous->next = new\_node;

}

count++;

return success;

}

double list1::p2(string d)

{

double x = 0;

node\*p = head;

while (p != NULL)

{

if (p->day == d)

{

x += p->m;

}

p = p->next;

}

return x;

}

void list1::p1(int position)

{

node\*p = set\_position(position);

cout << p->day << " " << p->way << " " << p->m << endl;

}

double list1::p4(string d, string xm)

{

double x = 0;

node\*p = head;

while (p != NULL)

{

if (p->day == d && p->way == xm)

{

x += p->m;

}

p = p->next;

}

return x;

}

int main()

{

List<int> A;

cout << "顺序表中插入1,2,3后" << endl;

A.insert(0, 1);

A.insert(1, 2);

A.insert(2, 3);

int a=0;

A.retrieve(1, a);

cout << "取第二个元素：";

cout << a << endl;

A.replace(1, 0);

A.retrieve(1, a);

cout << "把第二个元素换成0后取第二个元素：";

cout << a << endl;

cout << endl;

A.remove(0, a);

cout << endl;

cout << endl;

list<int>B;

cout << "单向链表中插入1,5,7后" << endl;

B.insert(0, 1);

B.insert(1, 5);

B.insert(2, 7);

B.retrieve(1, a);

cout << "取第二个元素：";

cout << a << endl;

B.replace(1, 3);

B.retrieve(1, a);

cout << "把第二个元素换成3后取第二个元素：";

cout << a << endl;

cout << "全部元素：";

B.traverse(c);

cout << endl;

B.reverse();

cout << "逆置后：";

B.traverse(c);

cout << endl;

cout << "在位置1插入7：";

B.insert(1, 7);

B.traverse(c);

cout << endl;

cout << "7的数目:" << B.tongji(7) << endl;

B.search(3);

B.traverse(c);

cout << endl;

B.remove(0, a);

cout << "删除第一个元素后的全部元素:";

B.traverse(c);

cout << endl;

list1 C;

cout << "选择输入支出项目的方法：（1是键盘输入，2是文件输入）" << endl;

string b;

getline(cin, b);

if (b == "1")

{

cout << "请输入日期：（结束输入#）" << endl;

string date;

getline(cin, date);

int i = 0;

while (date != "#")

{

cout << "请输入项目名称：" << endl;

string xm;

getline(cin, xm);

cout << "请输入金额：" << endl;

string money;

getline(cin, money);

double m = atof(money.c\_str());

C.insert(i, date, xm, m);

i++;

cout << "请输入日期：（结束输入#）" << endl;

getline(cin, date);

}

}

if (b == "2")

{

cout << "请输入文件名：(文件中格式应该是日期，项目名称，金额)" << endl;

string file;

getline(cin, file);

ifstream fin(file);

string date;

string xm;

double m;

int i = 0;

while (fin >> date >> xm >> m)

{

C.insert(i, date, xm, m);

i++;

}

fin.close();

}

cout << "最小消费金额：";

cout << C.min() << endl;

cout << "最大消费金额：";

cout << C.max() << endl;

cout << "平均消费金额：";

cout << C.aver() << endl;

string d;

cout << "请输入想查找的日期中的消费：";

getline(cin, d);

cout << "消费金额为：" << C.p2(d) << endl;

string xm;

cout << "请输入想查找的项目中的消费：";

getline(cin, xm);

cout << "消费金额为：" << C.p3(xm) << endl;

cout << "请输入想查找的日期和项目中的消费：";

getline(cin, d);

getline(cin, xm);

cout << "消费金额为：" << C.p4(d, xm) << endl;

system("pause");

}