WEEK5 Link\_Table

郁思敏 10153901225

**Problem:**

* 实现一个链表类，链表的某个元素是一个结构体，包含一个学生的id、姓名与成绩。需要实现下述功能：
  + 1. 插入一个学生的信息。
  + 2. 按照id删除一个学生的信息。
  + 3. 对链表进行反转
  + 4. 对一个链表按照学生成绩排序，成绩高的在链表头部
  + 5. 将另一个输入的有序链表合并为一个。

\* 内存空间分配需使用new实现。

**Solution:**

**1.Linked\_table.h**

#ifndef LINKED\_TABLE\_H\_INCLUDED

#define LINKED\_TABLE\_H\_INCLUDED

#include <string>

using namespace std;

typedef struct Student

{

int id;

char name[10];

float score;

struct Student \*prior;

struct Student \*next;

Student()

{

prior = NULL;

next = NULL;

}

} STUDENT;

class Link\_table

{

public:

Link\_table(); //initiate class private members.

STUDENT \*Initiate\_table(); //Create a new linked\_table.

void Insert(int location); //insert a student's info by location.

void Delete(int delete\_id); //delete a student's info by its id.

void Inverse(); //print linked\_table from back to forth.

void Sort(); //sort the table according to students' scores.

void Merge\_table(STUDENT \*head2); //merge two sorted table to one table.

void Print\_table(); //print current linked\_table.

int ShowCount(); //print the number of the students.

void ClearTable(); //free space.

~Link\_table(); //destruct. free head pointer.

private:

STUDENT \*head; //the head of the table, not including student's info.

int count; //the number of the students.

};

#endif // LINKED\_TABLE\_H\_INCLUDED

**2.Linked\_table.cpp**

#include <iostream>

#include <cstring>

#include "Linked\_table.h"

using namespace std;

void Swap(STUDENT \*a, STUDENT \*b);

Link\_table::Link\_table() //initiate class private members.

{

count = 0;

head = new STUDENT;

}

STUDENT \*Link\_table::Initiate\_table() //Create a new linked\_table.

{

int n, i;

cout << "please input the number of your linked\_table:" << endl;

cin >> n;

STUDENT \*pri = head;

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

{

STUDENT \*cur = new STUDENT;

cout << "please enter id, name and score:" << endl;

cin >> cur->id >> cur->name >> cur->score;

cur->next = NULL;

cur->prior = pri;

pri->next = cur;

pri = cur;

count++;

}

return head;

}

void Link\_table::Print\_table() //print current linked\_table.

{

STUDENT \*pt = head->next;

cout << "print the table:" << endl;

while (pt!=NULL)

{

cout << pt->id << " " << pt->name << " " << pt->score << endl;

pt = pt->next;

}

}

void Link\_table::Insert(int location) //insert a student's info by location.

{

if(location<=count)

{

int i;

STUDENT \*in = new STUDENT;

STUDENT \*pt = head->next;

for (i=1; i<location; i++)

{

pt = pt->next;

}

cout << "please input the info of the insert student:" << endl;

cin >> in->id >> in->name >> in->score;

if (pt->next!=NULL)

pt->next->prior = in;

in->next = pt->next;

pt->next = in;

in->prior = pt;

count++;

}

else

cout << "input error!" <<endl;

}

void Link\_table::Delete(int delete\_id) //delete a student's info by its id.

{

bool flag = false;

STUDENT \*pt = head->next;

while(pt!=NULL)

{

if (pt->id == delete\_id)

{

if (pt->next != NULL)

{

pt->prior->next = pt->next;

pt->next->prior = pt->prior;

}

else

{

pt->prior->next = NULL;

}

delete pt;

count--;

flag = true;

cout << "delete node whose id=" << delete\_id << endl;

break;

}

else

{

pt = pt->next;

}

}

if (!flag)

cout << "There is no such a id." << endl;

}

void Link\_table::Inverse() //print linked\_table from back to forth.

{

cout << "inversed table:" << endl;

STUDENT \*pt = head;

while(pt->next!=NULL)

{

pt = pt->next;

}

while(pt->prior!=NULL)

{

cout << pt->id << " " << pt->name << " " << pt->score << endl;

pt = pt->prior;

}

}

int Link\_table::ShowCount() //print the number of the students.

{

cout << "sum student:" << count << endl;

return count;

}

void Link\_table::Sort() //sort the table according to students' scores.

{

int i, j;

STUDENT \*pt1, \*pt2;

pt1 = head->next;

for (i=1; i<count; i++)

{

pt2 = pt1->next;

for(j=i+1; j<=count; j++)

{

if(pt1->score < pt2->score)

{

Swap(pt1, pt2);

}

pt2 = pt2->next;

}

pt1 = pt1->next;

}

}

void Swap(STUDENT \*a, STUDENT \*b)

{

int id;

char name[10];

float score;

id = a->id;

strcpy(name,a->name);

score = a->score;

a->id = b->id;

b->id = id;

strcpy(a->name, b->name);

strcpy(b->name, name);

a->score = b->score;

b->score = score;

}

void Link\_table::Merge\_table(STUDENT \*head2) //merge two sorted table to one table.

{

STUDENT \*pt1, \*pt2, \*cur;

pt1 = head->next;

pt2 = head2->next;

cur = head;

while(pt1!=NULL && pt2!=NULL)

{

if(pt1->score >= pt2->score)

{

cur->next = pt1;

pt1->prior = cur;

pt1 = pt1->next;

}

else

{

cur->next = pt2;

pt2->prior = cur;

pt2 = pt2->next;

}

cur = cur->next;

}

while(pt1!=NULL)

{

cur->next = pt1;

pt1->prior = cur;

pt1 = pt1->next;

cur = cur->next;

}

while(pt2!=NULL)

{

cur->next = pt2;

pt2->prior = cur;

pt2 = pt2->next;

cur = cur->next;

}

}

void Link\_table::ClearTable() //free memory.

{

STUDENT \*cur, \*pt = head->next;

while(pt!=NULL)

{

cur = pt->next;

delete pt;

pt = cur;

}

}

Link\_table::~Link\_table() //free head pointer.

{

delete head;

head = NULL;

}

**3.main.cpp**

#include <iostream>

#include "Linked\_table.h"

using namespace std;

int main()

{

int id;

int location;

Link\_table table1;

Link\_table table2;

STUDENT \*head2;

table1.Initiate\_table(); //create table1

cout << "please input the location after where you want to insert a student's info:" <<endl;

cin >> location;

table1.Insert(location); //1.insert student's info by location.

table1.Print\_table(); //show table after insertion.

cout << "please input the student's id which you want to delete:n<=" << endl;

cin >> id;

table1.Delete(id); //2.delete a student's info according to its id.

table1.Print\_table(); //show table after deletion.

table1.Inverse(); //3.inverse table1

table1.Sort(); //4.Sort table1 by students' score.

cout << "table1 after sort:" << endl;

table1.Print\_table(); //show table1 after sort.

head2 = table2.Initiate\_table(); //create table2

table2.Sort(); //4.Sort table2 by students' score.

cout << "table2 after sort:" << endl;

table2.Print\_table(); //show table2 after sort.

table1.Merge\_table(head2); //5.merge table1 and table2.

table1.Print\_table(); //show table1 after merge.

table1.ClearTable(); //free memory. Because table2 has already linked to table1, so there is no need to free table2 any more.

return 0;

}