# 实验四 结构体与链表

## 题目1 日期计算

### 实验描述：

定义一个结构体变量（包括年、月、日），编程序，要求输入年月日，程序能计算并输出该日在本年中第几天。注意闰年问题。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\1date.cpp

 \* @Description: 用结构体存储日期，计算某天是一年中的第几天

 \*/

#include <iostream>

using namespace std;

struct Date

{

    int year;

    int month;

    int day;

};

/\*\*

 \* @description: 判断是否为闰年

 \* @param {int} year 年份

 \* @return {\*} 是否为闰年

 \*/

bool isLeapYear(int year) // 判断是否为闰年

{

    return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;

}

/\*\*

 \* @description: 计算某年某月的天数

 \* @param {int} year

 \* @param {int} month

 \* @return {\*} 天数

 \*/

int daysOfMonth(int year, int month) // 计算某年某月的天数

{

    switch (month)

    {

    case 2:

        return isLeapYear(year) ? 29 : 28;

    case 4:

    case 6:

    case 9:

    case 11:

        return 30;

    default:

        return 31;

    }

}

int main()

{

    Date date;

    cout << "Input year, month, day: ";

    cin >> date.year >> date.month >> date.day;

    // 判断输入的日期是否合法

    if (date.year < 0 || date.month < 1 || date.month > 12 || date.day < 1 || date.day > daysOfMonth(date.year, date.month))

    {

        cout << "Invalid date" << endl;

        return 0;

    }

    int days = 0;

    for (int i = 1; i < date.month; i++)

    {

        days += daysOfMonth(date.year, i); // 计算前几个月的天数

    }

    days += date.day; // 加上当月的天数

    cout << date.year << "." << date.month << "." << date.day

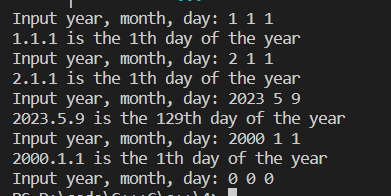
         << "is the " << days << "th day of the year" << endl;

    delete &date;

    return 0;

}

### 实验结果：



### 思考与扩展：

1. 修改程序，在程序中用指针方法访问结构体变量的成员。

以下是修改后的程序

// 通过函数指针调用函数

int result = (\*sum)(2, 3);

#include <iostream>

using namespace std;

struct Date

{

    int year;

    int month;

    int day;

};

bool isLeapYear(int year)

{

    return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;

}

int daysOfMonth(int year, int month)

{

    switch (month)

    {

    case 2:

        return isLeapYear(year) ? 29 : 28;

    case 4:

    case 6:

    case 9:

    case 11:

        return 30;

    default:

        return 31;

    }

}

int main()

{

    Date\* date = new Date; // 创建结构体变量的指针

    while (true)

    {

        cout << "Input year, month, day: ";

        cin >> date->year >> date->month >> date->day; // 使用指针方法访问结构体变量的成员

        if (date->year == 0 && date->month == 0 && date->day == 0)

        {

            break;

        }

        if (date->year < 0 || date->month < 1 || date->month > 12 || date->day < 1 || date->day > daysOfMonth(date->year, date->month))

        {

            cout << "Invalid date" << endl;

            continue;

        }

        int days = 0;

        for (int i = 1; i < date->month; i++)

        {

            days += daysOfMonth(date->year, i);

        }

        days += date->day;

        cout << date->year << "." << date->month << "." << date->day

             << " is the " << days << "th day of the year" << endl;

    }

    delete date; // 释放动态分配的内存

    return 0;

}

1. 结构体变量占用的存储单元字节数是多少，能不能用sizeof（）求结构体变量字节

数？

struct Date

{

    int year;

    int month;

    int day;

};

Date结构体在内存单元中占12个字节，在内存单元中可以对齐。可以用sizeof()求结构体变量字节数。

## 题目2 学生成绩管理

### 实验描述：

有 5 个学生，每个学生的数据包括学号、班级、姓名、三门课成绩。从键盘输入5 个学生数据，要求打印出每个学生三门课的平均成绩，以及平均分最高分学生数据（包括学号、班级、姓名、三门课成绩，平均分）。

要求：

1、定义学生结构体。

2、用一个函数实现5 个学生数据的输入，用另一个函数负责求每个学生三门课程的平均成绩，再用一个函数求出平均分最高的学生并输出该学生的数据。要求平均分和平均分最高的学生数据都在主函数种输出。

3、用多文件程序结构实现。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\2score\score.cpp

 \* @Description: 分析学生成绩总函数，包括输入学生信息、计算平均分、查找最高分等

 \*/

//g++ -o score score.cpp student.cpp

#include <iostream>

#include "student.h"

using namespace std;

int main()

{

    int number = 5;

    Student stu[20];

    // cout << "Please enter student number: ";

    // cin >> number;

    input\_students(stu, number);

    calculate\_average\_score(stu, number);

    int max\_index = find\_highest\_average\_score(stu, number);

    cout << "Average scores:" << endl;

    for (int i = 0; i < number; i++)

    {

        cout << "Student " << i << ": " << stu[i].aver\_score << endl;

    }

    cout << "The student with the highest average score:" << endl;

    cout << "Num: " << stu[max\_index].num << endl;

    cout << "Name: " << stu[max\_index].name << endl;

    cout << "Class: " << stu[max\_index].classname << endl;

    cout << "Scores: " << stu[max\_index].score[0] << " " << stu[max\_index].score[1] << " " << stu[max\_index].score[2] << endl;

    cout << "Average score: " << stu[max\_index].aver\_score << endl;

    system("pause");

    return 0;

}

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\2score\student.h

 \* @Description: 头文件，包括Student结构体定义和各功能函数声明

 \*/

// student.h

#ifndef STUDENT\_H

#define STUDENT\_H

typedef struct Student

{

    int num;

    char name[20];

    char classname[20];

    float score[3];

    float aver\_score;

} Student;

void input\_students(Student stu[], int number);

void calculate\_average\_score(Student stu[], int number);

int find\_highest\_average\_score(Student stu[], int number);

#endif

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\2score\student.cpp

 \* @Description: 功能函数实现，包括输入信息函数，计算平均分函数，查找最高分函数

 \*/

#include <iostream>

#include "student.h"

using namespace std;

/\*\*

 \* @description: 输入学生信息

 \* @param {Student} stu 要输入学生的结构体数组

 \* @param {int} number 要输入的学生数量

 \* @return {\*}

 \*/

void input\_students(Student stu[], int number)

{

    for (int i = 0; i < number; i++)

    {

        cout << "Please enter NO." << i << " student info: num name class score1 score2 score3" << endl;

        cin >> stu[i].num >> stu[i].name >> stu[i].classname;

        cin >> stu[i].score[0] >> stu[i].score[1] >> stu[i].score[2];

    }

}

/\*\*

 \* @description: 计算每个学生各科的平均分，并存入到结构体数组中

 \* @param {Student} stu 学生结构体数组

 \* @param {int} number  学生数量

 \* @return {\*}

 \*/

void calculate\_average\_score(Student stu[], int number)

{

    for (int i = 0; i < number; i++)

    {

        stu[i].aver\_score = (stu[i].score[0] + stu[i].score[1] + stu[i].score[2]) / 3.0;

    }

}

/\*\*

 \* @description: 顺序查找寻找平均分最高的学生

 \* @param {Student} stu 学生结构体数组

 \* @param {int} number  学生数量

 \* @return {\*}  返回平均分最高的学生的下标

 \*/

int find\_highest\_average\_score(Student stu[], int number)

{

    int max\_index = 0;

    float max\_score = stu[0].aver\_score;

    for (int i = 1; i < number; i++)

    {

        if (stu[i].aver\_score > max\_score)

        {

            max\_index = i;

            max\_score = stu[i].aver\_score;

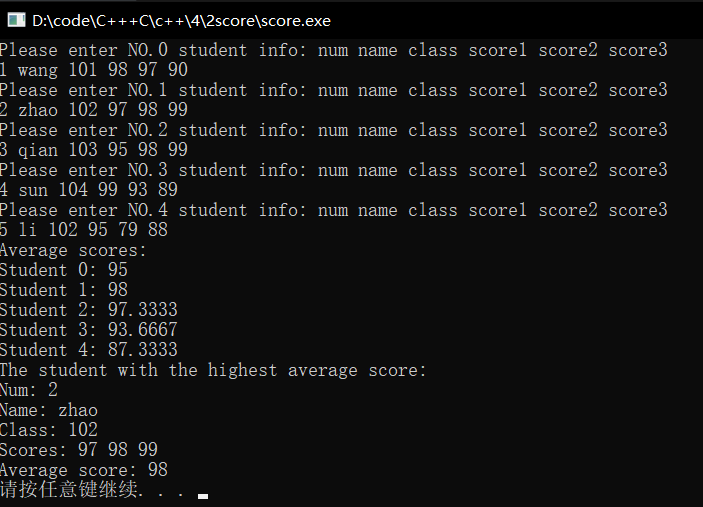
        }

    }

    return max\_index;

}

### 实验结果：



### 思考与扩展：

1、如果这样定义结构体

typedef struct Student

{

int num； //学号

char name[20]； //姓名注意字节长度

char classname[20]； //班级

float score[3]； //三门课程成绩

float aver\_score; //平均分

} STU，\* pSTU；

说明如下三条语句的含义：

student stu1;

STU stu2;

pSTU pstu;

* student stu1;声明一个student结构体，名称为stu1
* STU stu2;声明一个student结构体，名称为stu2
* pSTU pstu;声明一个student结构体指针，名称为pstu

2、结构体中姓名和班级字段如果定义为 string 类型，程序该如何改写？

程序只需要进行如下改写

#include <string>

typedef struct Student

{

    int num;

    std::string name;

    std::string classname;

    float score[3];

    float aver\_score;

} Student;

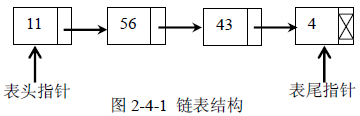
## 题目3 结构体链表

### 实验描述：

设计一个单向链表。从标准输入读取若干整数，建立链表，每次读入的数放入链表结尾。

当用户输入 0 时，结束链表的建立工作。然后从前往后依次输出链表节点中的内容。链表

的结构类似于下图：



每个节点包含两个值，一个是真正存放的整数值，另一个为指向链表中下一个节点的指针。链表中最后一个节点不指向任何节点，所以指针为空（NULL）。表头指针和表尾指针分别指向链表的头节点和尾节点。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\3link.cpp

 \* @Description: 设计一个单向链表,要求实现链表的建立、遍历和释放节点空间的功能。

 \*/

#include <iostream>

using namespace std;

// 定义链表节点类型

typedef struct node

{

    int elem;

    node \*next;

} Node;

// 定义链表类型

typedef struct list

{

    Node \*head; // 链表头指针

    Node \*tail; // 链表尾指针

} List;

int main()

{

    List mylist = {NULL, NULL}; // 初始化链表为空

    int input = -1;

    cout << "Please input integers to build the link(0 TO END):" << endl;

    while (input != 0)

    {

        cin >> input;

        if (input != 0)

        {

            Node \*pNode = new Node; // 动态分配节点空间

            if (pNode == NULL)      // 判断分配是否成功

            {

                cout << "Memory insufficient!" << endl;

                break;

            }

            pNode->elem = input;     // 节点存放值

            pNode->next = NULL;      // 节点指针赋值为空

            if (mylist.head == NULL) // 如果是第一个节点，链表为空

            {

                mylist.head = mylist.tail = pNode; // 第一个节点既是头也是尾。

            }

            else

            {

                mylist.tail->next = pNode; // 尾节点的指针指向新节点，即将新节点加到链表中

                mylist.tail = pNode;       // 新节点加入到链表尾后，让尾指针指向该节点。

            }

        }

    }

    // 顺序遍历链表并释放节点空间

    Node \*pNode = mylist.head; // 让一个指针pNode 指向链表头

    cout << "Link elements:";

    while (pNode != NULL)

    {

        cout << pNode->elem << " "; // 输出节点存储的数据

        Node \*pNext = pNode->next;  // 保存下一个节点的指针

        delete pNode;               // 释放当前节点的空间

        pNode = pNext;              // 让pNode 指向下一个节点。

    }

    return 0;

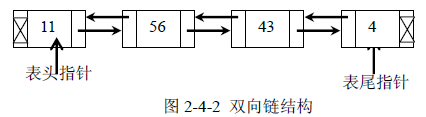
}

### 实验结果：



### 思考与扩展：

如果是双向链表，程序要做那些改动？双向链表的示意图如下：



修改节点结构体的定义：

typedef struct node

{

    int elem;

    node \*prev; // 新增指向前一个节点的指针

    node \*next;

} Node;

修改链表结构体的定义：

typedef struct list

{

    Node \*head;

    Node \*tail;

} List;

修改链表节点的插入逻辑：

if (mylist.head == NULL)

{

    mylist.head = mylist.tail = pNode;

    pNode->prev = NULL; // 新增代码，将头节点的prev指针置为NULL

}

else

{

    pNode->prev = mylist.tail;  // 将新节点的prev指针指向当前尾节点

    mylist.tail->next = pNode;  // 尾节点的next指针指向新节点

    mylist.tail = pNode;        // 将尾指针指向新节点

}

修改节点遍历和释放逻辑：

Node \*pNode = mylist.head;

cout << "Link elements:";

while (pNode != NULL)

{

    cout << pNode->elem << " ";

    Node \*pNext = pNode->next;

    delete pNode;

    pNode = pNext;

}

## 题目4 通信录

### 实验描述：

设一个通信录由以下几项数据信息构成：

数据项 类型

姓名 字符串

地址 字符串

邮政编码 字符串

电话号码 字符串

电子邮件 字符串

实现一个通讯录程序。实现功能：存取、显示、删除记录、查询记录等功能。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\4\4tongxin.cpp

 \* @Description: 实现一个通讯录程序。实现功能：存取、显示、删除记录、查询记录等功能。

 \*/

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Define a struct to represent a contact

struct Contact

{

    string name;

    string address;

    string postal\_code;

    string phone\_number;

    string email;

};

// Define a vector to store all contact information

vector<Contact> contacts;

// Add contact

/\*\*

 \* @description: 添加一个联系人，包括姓名、地址、邮编、电话、邮箱，将其存入到vector中

 \* @return {\*}

 \*/

void add\_contact()

{

    Contact c;

    cout << "Please enter the contact name:";

    getline(cin, c.name);

    cout << "Please enter the contact address:";

    getline(cin, c.address);

    cout << "Please enter the postal code of the contact person:";

    getline(cin, c.postal\_code);

    cout << "Please enter the contact phone number:";

    getline(cin, c.phone\_number);

    cout << "Please enter the contact email:";

    getline(cin, c.email);

    contacts.push\_back(c);

    cout << "Contact added" << endl;

}

// Delete contact

/\*\*

 \* @description: 删除一个联系人，根据姓名查找，找到后删除

 \* @return {\*}

 \*/

void delete\_contact()

{

    string name;

    cout << "Please enter the name of the contact you want to delete:";

    getline(cin, name);

    for (int i = 0; i < contacts.size(); i++) // 遍历联系人向量

    {

        if (contacts[i].name == name)

        {

            contacts.erase(contacts.begin() + i);

            cout << "Contact deleted" << endl;

            return;

        }

    }

    cout << "The contact was not found" << endl;

}

// Search contact

/\*\*

 \* @description: 查找一个联系人，根据姓名查找，找到后输出联系人信息

 \* @return {\*}

 \*/

void search\_contact()

{

    string name;

    cout << "Please enter the name of the contact you want to query:";

    getline(cin, name);

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

    {

        if (contacts[i].name == name)

        {

            cout << "Name:" << contacts[i].name << endl;

            cout << "Address:" << contacts[i].address << endl;

            cout << "Postal Code:" << contacts[i].postal\_code << endl;

            cout << "Telephone Number:" << contacts[i].phone\_number << endl;

            cout << "E-mail:" << contacts[i].email << endl;

            return;

        }

    }

    cout << "The contact was not found" << endl;

}

// Display all contacts

/\*\*

 \* @description: 输出所有联系人信息

 \* @return {\*}

 \*/

void show\_contacts()

{

    if (contacts.size() == 0)

    {

        cout << "Don't have any contacts" << endl;

        return;

    }

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

    {

        cout << "Contact person " << i + 1 << ":" << endl;

        cout << "Name:" << contacts[i].name << endl;

        cout << "Address:" << contacts[i].address << endl;

        cout << "Postal Code:" << contacts[i].postal\_code << endl;

        cout << "Telephone Number:" << contacts[i].phone\_number << endl;

        cout << "E-mail:" << contacts[i].email << endl;

    }

}

int main()

{

    int choice;

    while (true) // 选择菜单页面

    {

        cout << "Please select an operation:" << endl;

        cout << "1. Add contact" << endl;

        cout << "2. Delete contact" << endl;

        cout << "3. Search contact" << endl;

        cout << "4. Show all contacts" << endl;

        cout << "0. Exit program" << endl;

        cin >> choice;

        cin.ignore();

        switch (choice)

        {

        case 1:

            add\_contact();

            break;

        case 2:

            delete\_contact();

            break;

        case 3:

            search\_contact();

            break;

        case 4:

            show\_contacts();

            break;

        case 0:

            return 0;

        default:

            cout << "Invilad input" << endl;

            break;

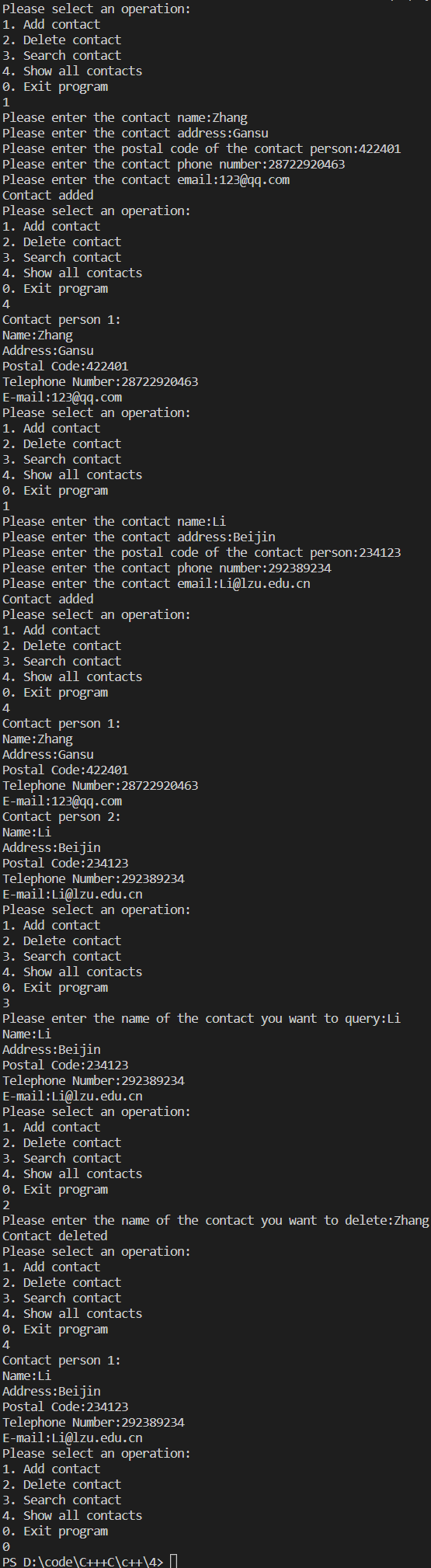
        }

    }

    return 0;

}

### 实验结果：



### 思考与扩展：

无