# 实验九 文件操作

## 题目1 二进制文件操作

### 实验描述：

从键盘输入字符串并保存成文件。

要求：

1、输入字符串中包括：大小写字母、空格、数字等字符，以“#”作为字符串输入的结

束条件；

2、对输入的字符进行加密并保存，以二进制文件方式保存，用户可以输入保存路径，

如：C：\file.txt ；

3、打开文件时，对加密文件进行解密并显示在屏幕上。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\9\1document.cpp

 \* @Description: 该程序用于对用户输入的字符串进行加密和解密操作，并将加密后的数据保存到文件中。用户可以输入包含大小写字母、空格和数字的字符串，以'#'作为字符串输入的结束条件。程序将输入的字符进行加密，加密规则为将字母A变成E，a变成e，即将字母变成其后的第4个字母，W变A，X变成B，Y变成C，Z变成D，w变a，x变成b，y变成c，z变成d。保存加密后的数据时，以二进制文件方式保存。用户可以输入保存路径，如：C:\file.txt。在打开文件时，程序会对加密文件进行解密并显示在屏幕上。

 \*/

#include <iostream>

#include <fstream>

using namespace std;

// 加密函数

/\*\*

 \* @description: 将字符进行加密

 \* @param {char} c 待加密的字符

 \* @return {char} 加密后的字符

 \*/

char encrypt(char c)

{

    if (c >= 'A' && c <= 'Z')

    {

        return (c - 'A' + 4) % 26 + 'A';

    }

    else if (c >= 'a' && c <= 'z')

    {

        return (c - 'a' + 4) % 26 + 'a';

    }

    else

    {

        return c;

    }

}

// 解密函数

/\*\*

 \* @description: 将字符进行解密

 \* @param {char} c 待解密的字符

 \* @return {char} 解密后的字符

 \*/

char decrypt(char c)

{

    if (c >= 'A' && c <= 'Z')

    {

        return (c - 'A' + 22) % 26 + 'A';

    }

    else if (c >= 'a' && c <= 'z')

    {

        return (c - 'a' + 22) % 26 + 'a';

    }

    else

    {

        return c;

    }

}

int main()

{

    // 打开文件并写入加密数据

    ofstream fout;

    fout.open(".\\file.txt", ios::binary);

    if (!fout)

    {

        cout << "Failed to open file." << endl;

        return 1;

    }

    cout << "Please input the string you want to encrypt, end with '#':" << endl;

    char c;

    while (cin.get(c) && c != '#')

    {

        c = encrypt(c);

        fout.write((char \*)&c, sizeof(char));

    }

    fout.close();

    // 读取加密数据并解密

    ifstream fin;

    fin.open(".\\file.txt", ios::binary);

    if (!fin)

    {

        cout << "Failed to open file." << endl;

        return 1;

    }

    cout << "The decrypted string is:" << endl;

    while (fin.read((char \*)&c, sizeof(char)))

    {

        c = decrypt(c);

        cout << c;

    }

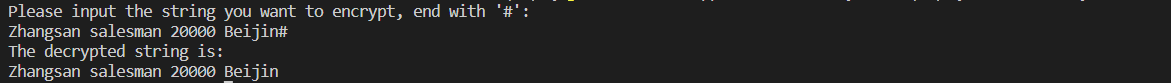
    fin.close();

    return 0;

}

### 实验结果：

程序运行结果：



存储文件内容：



### 思考与扩展：

无

## 题目2 人员信息保存

### 实验描述：

为第六次实验的题目三“人员管理”添加文件读取保存功能

1、将输入人员信息保存成文件；

3、读取已保存的人员信息文件，并显示文件中保存的人员信息；

### 实验代码：

#### Main.cpp

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\9\employee\Main.cpp

 \* @Description:

 \*/

// g++ -std=c++11 -o Main Main.cpp employee.cpp Date.cpp

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

#include <map>

#include "employee.h"

#include "Date.h"

using namespace std;

// 部门销售额统计

map<string, int> department\_map;  // 使用map容器存储部门和对应的销售额

// manager 类

class manager : virtual public employee

{

public:

    manager() {}

    manager(int id, const string &name, char gender, const Date &birth) : employee(id, name, gender, birth, "Manager", GetMonthlySalary()) {}

    /\*\*

     \* @description: 显示经理的信息

     \* @param {ostream} &fout 输出流对象

     \* @return {\*}

     \*/

    void Display(ostream &fout) override

    {

        DisplayEmployee(\*this, fout);  // 调用DisplayEmployee函数显示经理的基本信息

    }

    /\*\*

     \* @description: 获取经理的月薪

     \* @return {double} 经理的月薪

     \*/

    double GetMonthlySalary() const override

    {

        return 8000.0;  // 固定月薪为8000.0元

    }

};

// technician 类

class technician : public employee

{

public:

    technician() {}

    technician(int id, const string &name, char gender, const Date &birth, double work\_time) : work\_time\_(work\_time)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->gender\_ = gender;

        this->birth\_ = birth;

        this->position\_ = "technician";

        this->salary\_ = GetMonthlySalary();

    }

    void Display(ostream &fout) override // 重写虚函数,添加工作时间的输出

    {

        DisplayEmployee(\*this, fout);

        fout << "Work time:" << work\_time\_ << endl;

    }

    double GetMonthlySalary() const override

    {

        return work\_time\_ \* 25;

    }

private:

    double work\_time\_;

};

// salesman 类

class salesman : virtual public employee

{

public:

    salesman() {}

    salesman(int id, const string &name, char gender, const Date &birth, const string &department, double sales) : department\_(department), sales\_(sales)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->gender\_ = gender;

        this->birth\_ = birth;

        this->position\_ = "salesman";

        this->salary\_ = GetMonthlySalary();

        department\_map[department] += sales;

    }

    void Display(ostream &fout) override

    {

        DisplayEmployee(\*this, fout);

        fout << "Department:" << department\_ << endl;

        fout << "Sales:" << sales\_ << endl;

    }

    double GetMonthlySalary() const override

    {

        return sales\_ \* 0.04;

    }

protected:

    string department\_;

    double sales\_;

};

// salesmanager 类

class salesmanager : public manager, public salesman

{

public:

    salesmanager() {}

    salesmanager(int id, const string &name, char gender, const Date &birth, const string &department)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->birth\_ = birth;

        this->gender\_ = gender;

        this->department\_ = department;

        this->position\_ = "salesmanager";

        this->salary\_ = GetMonthlySalary();

    }

    void Display(ostream &fout) override

    {

        DisplayEmployee(\*this, fout);

        fout << "Department:" << department\_ << endl;

        fout << "Sales:" << department\_map[department\_] << endl;

    }

    double GetMonthlySalary() const override

    {

        return 5000 + department\_map[department\_] \* 0.05;

    }

};

/\*\*

 \* @description: 将员工信息保存到文件中

 \* @param {vector<employee \*>} &employees 员工向量的引用，包含要保存的员工信息

 \* @return {\*}

 \*/

void SaveToFile(vector<employee \*> &employees)

{

    ofstream fout(".\\employee.txt");          // 打开名为"employee.txt"的文件进行写入

    for (int i = 0; i < employees.size(); i++) // 遍历employees向量中的每个员工

    {

        employees[i]->Display(fout); // 调用员工对象的Display方法，将员工信息写入文件

        fout << "#" << endl;         // 输出"#"表示一条员工信息的结束

    }

}

/\*\*

 \* @description: 从文件中加载员工信息并存储到员工向量中

 \* @param {vector<employee \*>} &employees 员工向量的引用，用于存储加载的员工信息

 \* @return {\*}

 \*/

void LoadFromFile(vector<employee \*> &employees)

{

    ifstream fin(".\\employee.txt"); // 打开名为"employee.txt"的文件进行读取

    if (!fin.is\_open())              // 如果文件打开失败，则输出错误信息并退出

    {

        cout << "Open file failed!" << endl;

        return;

    }

    string line;                                        // 用于存储读取的每一行内容

    int id, year, month, day, salary, work\_time, sales; // 员工信息的各个字段

    string name, position, department;                  // 员工信息的各个字段

    char gender;                                        // 员工信息的各个字段

    // 逐行读取文件内容

    while (getline(fin, line))

    {

        // cout << line << endl;

        if (line.find("#") == 0) // 如果行以"#"开头，则表示员工信息的一条记录结束

        {

            if (position == "Manager") // 如果职位为Manager

            {

                employees.push\_back(new manager(id, name, gender, Date(year, month, day))); // 创建Manager对象并添加到employees向量中

            }

            else if (position == "technician") // 如果职位为technician

            {

                employees.push\_back(new technician(id, name, gender, Date(year, month, day), salary)); // 创建technician对象并添加到employees向量中

            }

            else if (position == "salesman") // 如果职位为salesman

            {

                employees.push\_back(new salesman(id, name, gender, Date(year, month, day), department, salary)); // 创建salesman对象并添加到employees向量中

            }

            else if (position == "salesmanager") // 如果职位为salesmanager

            {

                employees.push\_back(new salesmanager(id, name, gender, Date(year, month, day), department)); // 创建salesmanager对象并添加到employees向量中

            }

        }

        else if (line.find("ID") == 0) // 如果行以"ID"开头，则表示当前行为员工的ID信息

        {

            id = stoi(line.substr(3)); // 提取行中的数字部分，将其转换为整数，赋值给变量id

        }

        else if (line.find("Name") == 0) // 如果行以"Name"开头，则表示当前行为员工的姓名信息

        {

            name = line.substr(5); // 提取行中的姓名部分，赋值给变量name

        }

        else if (line.find("Gender") == 0) // 如果行以"Gender"开头，则表示当前行为员工的性别信息

        {

            gender = line[7]; // 提取行中的性别字符，赋值给变量gender

        }

        else if (line.find("Birthday") == 0) // 如果行以"Birthday"开头，则表示当前行为员工的出生日期信息

        {

            year = stoi(line.substr(9, 4));   // 提取行中的年份部分，转换为整数，赋值给变量year

            month = stoi(line.substr(14, 2)); // 提取行中的月份部分，转换为整数，赋值给变量month

            day = stoi(line.substr(17, 2));   // 提取行中的日期部分，转换为整数，赋值给变量day

        }

        else if (line.find("Position") == 0) // 如果行以"Position"开头，则表示当前行为员工的职位信息

        {

            position = line.substr(9); // 提取行中的职位部分，赋值给变量position

        }

        else if (line.find("Work time") == 0) // 如果行以"Work time"开头，则表示当前行为员工的工作时间信息

        {

            work\_time = stoi(line.substr(10)); // 提取行中的数字部分，将其转换为整数，赋值给变量work\_time

        }

        else if (line.find("Department") == 0) // 如果行以"Department"开头，则表示当前行为员工的部门信息

        {

            department = line.substr(11); // 提取行中的部门部分，赋值给变量department

        }

        else if (line.find("Sales") == 0) // 如果行以"Sales"开头，则表示当前行为销售员工的销售额信息

        {

            sales = stoi(line.substr(6)); // 提取行中的数字部分，将其转换为整数，赋值给变量sales

        }

        else if (line.find("Salary") == 0) // 如果行以"Salary"开头，则表示当前行为员工的薪水信息

        {

            salary = stoi(line.substr(7)); // 提取行中的数字部分，将其转换为整数，赋值给变量salary

        }

    }

}

int main()

{

    vector<employee \*> employees;                   // 存放所有员工的指针

    int choice, year, month, day, work\_time, sales; // 选择，年，月，日，工作时间，销售额

    int id;                                         // 员工编号

    string name;                                    // 员工姓名

    char gender;

    string department, position;

    while (true)

    {

        cout << "=====================================" << endl

             << "1.add manager" << endl

             << "2.add technician" << endl

             << "3.add salesman" << endl

             << "4.add salesmanager" << endl

             << "5.show all employees" << endl

             << "6.save to file" << endl

             << "7.load from file" << endl

             << "8.exit" << endl;

        cin >> choice; // 选择

        switch (choice)

        {

        case 1:

            cout << "Adding a manager(id,name,gender,year,month,day)" << endl;

            cin >> id >> name >> gender >> year >> month >> day;

            employees.push\_back(new manager(id, name, gender, Date(year, month, day))); // 添加一个manager对象

            break;

        case 2:

            cout << "Adding a technician(id,name,gender,year,month,day,work\_time)" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> work\_time;

            employees.push\_back(new technician(id, name, gender, Date(year, month, day), work\_time));

            break;

        case 3:

            cout << "Adding a salesman(id,name,gender,year,month,day,department,sales)" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> department >> sales;

            employees.push\_back(new salesman(id, name, gender, Date(year, month, day), department, sales));

            break;

        case 4:

            cout << "Adding a salesmanager(id,name,gender,year,month,day,department)" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> department;

            employees.push\_back(new salesmanager(id, name, gender, Date(year, month, day), department));

            break;

        case 5:

            cout << employees.size() << endl;

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

            {

                employees[i]->Display(cout);

            }

            cout << "=====================================" << endl;

            break;

        case 6:

            SaveToFile(employees);

            break;

        case 7:

            LoadFromFile(employees);

            break;

        case 8:

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

            {

                delete employees[i];

            }

            employees.clear();

            exit(0);

        default:

            cout << "Input error!" << endl;

            break;

        }

    }

    system("pause");

return 0;

#### employee.h

/\*

@Author: 王贤义

 \* @FilePath: \code\C+++C\c++\9\employee\employee.h

@Description: 员工类头文件

\*/

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <string>

#include "Date.h"

#include <iostream>

#include <fstream>

using namespace std;

class employee

{

public:

    employee() {} // 默认构造函数

    employee(int id, const string &name, char gender, const Date &birth, const string &position, double salary);

    // 构造函数，用于初始化员工对象

    /\*\*

     \* @description: 显示员工信息

     \* @param {employee} &e 员工对象引用

     \* @param {ostream} &fout 输出流对象引用

     \* @return {\*}

     \*/

    friend void DisplayEmployee(const employee &e, ostream &fout);

    // 友元函数，用于显示员工信息

    /\*\*

     \* @description: 获取员工月薪

     \* @return {double} 员工月薪

     \*/

    virtual double GetMonthlySalary() const = 0;

    // 纯虚函数，获取员工月薪

    /\*\*

     \* @description: 显示员工信息

     \* @param {ostream} &fout 输出流对象引用

     \* @return {\*}

     \*/

    virtual void Display(ostream &fout) = 0;

    // 纯虚函数，显示员工信息

protected:

    int id\_;          // 员工ID

    string name\_;     // 员工姓名

    char gender\_;     // 员工性别

    Date birth\_;      // 员工出生日期

    string position\_; // 员工职位

    double salary\_;   // 员工薪水

};

#endif

#### Employee.cpp

#include "employee.h"

using namespace std;

employee::employee(int id, const string &name, char gender, const Date &birth, const string &position, double salary) : id\_(id), name\_(name), gender\_(gender), birth\_(birth), position\_(position), salary\_(salary) {}

void DisplayEmployee(const employee &e, ostream &fout)

{

    fout << "==============================" << endl;

    fout << "ID:" << e.id\_ << endl;

    fout << "Name:" << e.name\_ << endl;

    fout << "Gender:" << e.gender\_ << endl;

    fout << "Birthday:" << e.birth\_.GetYear() << "." << e.birth\_.GetMonth() << "." << e.birth\_.GetDay() << endl;

    fout << "Position:" << e.position\_ << endl;

    fout << "Salary:" << e.salary\_ << endl;

}

#### Date.h

/\*

@Author: 王贤义

 \* @FilePath: \code\C+++C\c++\9\employee\Date.h

@Description: 日期类头文件

\*/

#ifndef DATE\_H

#define DATE\_H

class Date

{

public:

    Date() {} // 默认构造函数

    Date(int year, int month, int day);

    // 构造函数，用于初始化日期对象

    /\*\*

     \* @description: 设置年份

     \* @param {int} year 年份

     \* @return {\*}

     \*/

    void SetYear(int year);

    /\*\*

     \* @description: 设置月份

     \* @param {int} month 月份

     \* @return {\*}

     \*/

    void SetMonth(int month);

    /\*\*

     \* @description: 设置日期

     \* @param {int} day 日期

     \* @return {\*}

     \*/

    void SetDay(int day);

    /\*\*

     \* @description: 获取年份

     \* @return {int} 年份

     \*/

    int GetYear() const;

    /\*\*

     \* @description: 获取月份

     \* @return {int} 月份

     \*/

    int GetMonth() const;

    /\*\*

     \* @description: 获取日期

     \* @return {int} 日期

     \*/

    int GetDay() const;

private:

    int year\_;  // 年份

    int month\_; // 月份

    int day\_;   // 日期

};

#endif

#### Date.cpp

#include "date.h"

Date::Date(int year, int month, int day) : year\_(year), month\_(month), day\_(day) {}

void Date::SetYear(int year) { year\_ = year; }

void Date::SetMonth(int month) { month\_ = month; }

void Date::SetDay(int day) { day\_ = day; }

int Date::GetYear() const { return year\_; }

int Date::GetMonth() const { return month\_; }

int Date::GetDay() const { return day\_; }

### 实验结果：

测试数据：

添加Manager:

1

1

John

M

1990

5

15

添加Technician:

2

2

Jane

F

1995

10

20

160

添加Salesman:

3

3

Mike

M

1992

3

25

Sales

50000

添加SalesManager:

4

4

Sarah

F

1988

8

10

Sales

显示所有员工:

5

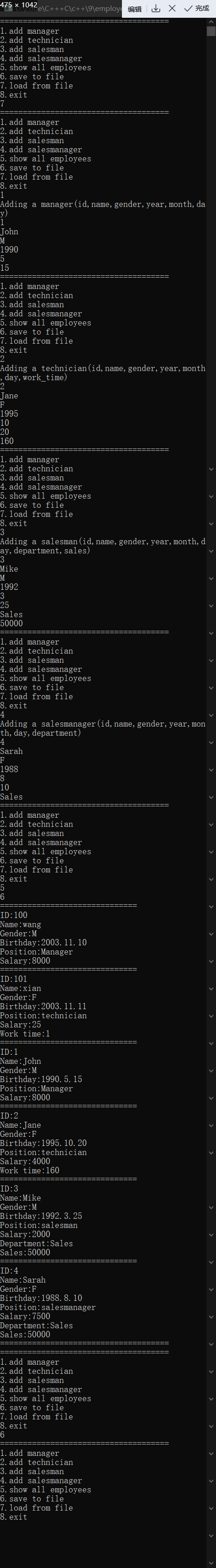
保存到文件:

6

从文件加载:

7

退出程序:

8

### 思考与扩展：

无

## 题目3 文件合并

### 实验描述：

合并两个文件成为一个文件

要求：

执行可执行程序，将两个文件合并成一个文件；

例如：combine.exe file1.txt file2.txt file3.txt

Combine.exe 为源程序生成的可执行程序，将file1.txt 和 file2.txt 中的内容合并，存放到file3.txt 这个文件中；

### 实验代码：

#### Combine.cpp

/\*

@Author: 王贤义

 \* @FilePath: \code\C+++C\c++\9\Combine\Combine.cpp

@Description: 文件合并程序

\*/

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

int main(int argc, char \*argv[])

{

    if (argc != 4)

    {

        cout << "Usage: combine.exe file1 file2 file3" << endl; // 使用说明: combine.exe是程序名，file1 file2 file3是参数

        return 1;

    }

    string file1\_name = argv[1]; // 获取文件1的名称

    string file2\_name = argv[2]; // 获取文件2的名称

    string file3\_name = argv[3]; // 获取文件3的名称

    ifstream file1(file1\_name); // 打开文件1进行读取

    ifstream file2(file2\_name); // 打开文件2进行读取

    ofstream file3(file3\_name); // 打开文件3进行写入

    if (!file1 || !file2 || !file3)

    {

        cout << "Error: Cannot open files" << endl; // 打开文件失败，输出错误信息

        return 1;

    }

    vector<string> lines; // 创建一个字符串向量，用于存储读取的文件内容

    string line;

    // 从文件1逐行读取内容，并存储到字符串向量中

    while (getline(file1, line))

    {

        lines.push\_back(line);

    }

    // 从文件2逐行读取内容，并存储到字符串向量中

    while (getline(file2, line))

    {

        lines.push\_back(line);

    }

    // 将字符串向量中的内容逐行写入文件3

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

    {

        file3 << lines[i] << endl;

    }

    cout << "Files combined successfully." << endl; // 文件合并成功

    return 0;

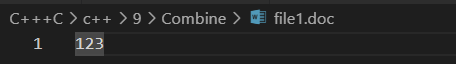
}

### 实验结果：

运行程序



file1.doc：



file3.doc:



file3.txt：

图形用户界面, 文本

描述已自动生成

### 思考与扩展：

针对不同类型的文件合并，是否需要修改源程序？

以下是一个用来合并csv（逗号分隔）文件的代码：

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

using namespace std;

// 函数：将逗号分隔的字符串拆分为字段并存储到向量中

vector<string> SplitCSV(const string& line) {

    vector<string> fields;

    stringstream ss(line);

    string field;

    while (getline(ss, field, ',')) {

        fields.push\_back(field);

    }

    return fields;

}

int main(int argc, char\* argv[]) {

    if (argc != 4) {

        cout << "Usage: combine.exe file1.csv file2.csv file3.csv" << endl;

        return 1;

    }

    string file1\_name = argv[1];

    string file2\_name = argv[2];

    string file3\_name = argv[3];

    ifstream file1(file1\_name);

    ifstream file2(file2\_name);

    ofstream file3(file3\_name);

    if (!file1 || !file2 || !file3) {

        cout << "Error: Cannot open files" << endl;

        return 1;

    }

    string line;

    vector<string> combinedLines;

    // 读取并合并文件1的内容

    while (getline(file1, line)) {

        combinedLines.push\_back(line);

    }

    // 读取并合并文件2的内容

    while (getline(file2, line)) {

        combinedLines.push\_back(line);

    }

    // 将合并的内容写入文件3

    for (const auto& combinedLine : combinedLines) {

        file3 << combinedLine << endl;

    }

    cout << "Files combined successfully." << endl;

    return 0;

}