# 类与对象

1. 从下面的文字描述中发现其中的一个主要概念，然后使用一个类来描述这个概念，使用对象描述概念包含的个体，最后，使用两种及以上的方法表示抽象出的类及对象，并画出对象的逻辑结构。

张三是一个大学生，学习计算科学与技术专业，李四与张三同一个年级，但学习软件工程专业，他们都是去年入学的，张三今年18岁，李四比张三大一岁，喜欢唱歌，张三是男的，比较高。

**答：**反复阅读文字描述，搞清楚哪些名词代表一类事物，哪些名词代表具体的一个事物，哪些动语或短语描述事物的行为，哪些词语或短语刻画哪些事物，然后再根据这些词语的含义抽象出类及其属性和行为、对象及其属性值。

文字描述中主要涉及“大学生”这个概念，可使用类“大学生”来描述这个概念， “张三”和“李四”是两个大学生，可使用两个对象分别描述“张三”和“李四”两个个体。具体分析如下：

张三(个体名词，对象)是一个大学生（抽象名词，主要概念，类），学习（动词，行为）计算科学与技术（个体名词，专业的属性值）专业（抽象名词，属性），李四(个体名词，对象)与张三同一个年级（抽象名词，属性），但学习软件工程（个体名词，专业的属性值）专业，他们都是去年（年级的属性值）入学的，张三今年18岁（年龄的属性值），李四比张三大一岁（年龄的属性值），喜欢（动词，行为）唱歌（动名词，爱好的属性值），张三是男的（性别的属性值），比较高（形容词，属性身高）。

可采用列举法和类图描述抽象出的类“大学生”。

大学生{姓名，专业，年级，年龄，爱好，性别，身高；学习（专业），喜欢（爱好)}



可采用列举法和对象图描述抽象出的对象“张三”和“李四”。

张三(”张三” , ”计算科学与技术”, 2022, 18,无, “男”,无):大学生

李四(”李四” ,“软件工程”, 2022, 19,”唱歌”,无,无):大学生



特别说明，对文字描述的理解不同，抽象出的类及对象也可能不同，只要能准确表示自己理解的含义并没有逻辑错误就可以。

1. 例1.2的程序中创建了Person的两个对象p1和p2，请画出这两个对象的物理结构，即内存中的结构。

**答：**p1是Person的全局对象，在全局数据区中分配存贮空间，p2是Person的局部对象，在栈区中分配存贮空间，除了存贮的数据不同外，其物理结构完全相同。p1和p2两个对象的物理结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Person p1 | | | | |
| char name[20] | bool sex | int age | float height | float weight |
| "张三" | 1 | 18 | 180.5 | 65.3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Person p2 | | | | |
| char name[20] | bool sex | int age | float height | float weight |
| "李四" | 0 | 16 | 165 | 46.5 |

1. 例1.3的程序中使用普通函数给对象赋值，请使用时序图描述创建对象和调用函数的过程。

**答：**我们用如附图1.1的时序图描述例1.3中创建对象和调用函数的过程。在例1.3中，首先定义了一个名为Person的类，该类表示一个人，具有属性（姓名、性别、年龄、身高、体重）和行为（打印信息、吃东西、睡觉）。

语句Person p1定义了类Person的一个全局对象p1，在执行main()函数的代码前，先在全局数据区为全局对象p1分配内存，然后按照p1. Person ()的语义调用默认构造函数。

在setValue函数中，通过传入指向Person对象的指针p和其他属性的值，函数使用strncpy函数来安全地将姓名复制到p->name中，然后将其他属性值分别赋值给p指向的Person对象的成员变量。



附图 1.1

在main函数中，首先是语句setValue(&p1,"张三", 1, 18, 180.5, 65.3)调用setValue函数设置了p1对象的属性，然后语句p1.print()调用了Person的成员函数print来打印p1对象的属性值。

接着在main函数中语句Person p2创建了一个Person类的位于栈中的局部对象p2，实际上p2的内存分配应该是在main函数第一行代码执行前完成，此时仅是按照p2. Person ()的语义调用默认构造函数。接着语句setValue(&p2,"李四", 0, 16, 165, 46.5)调用setValue函数设置了p2对象的属性，然后语句p2.print()再调用了Person的成员函数print来打印p2对象的属性值。

1. 例1.5的程序中创建和删除了全局对象、局部对象和静态对象，请用文字简述其创建和删除的过程，并画出第二次调用fn()函数中返回前的内存状态，包括全局数据区和栈区中的对象，以及对象中的数据。

**答：**先读懂图1.23中的时序图，然后用文字描述例1.5中对象的创建和删除过程，即依照时序逐条描述消息的语义，最后，使用内存图描述第二次调用fn()函数中返回前的内存状态。

例1.5中，语句Point g(0, 0) 定义了类Point的一个全局对象g，fn()函数中的语句static Point sf(1,1) 定义了一个静态对象sf，语句Point f(2,2) 定义了一个局部对象f。main()函数中，调用了fn()函数两次，通过这两次调用可以观察到创建对象的过程，以理解定义各类对象的语义。创建和删除对象的过程，如附图1.2所示。

语句Point g(0, 0) 定义了类Point的一个全局对象g，在执行main()函数的代码前，先在全局数据区为全局对象g分配内存，然后按照g.Point (0, 0)的语义调用构造函数。

语句static Point sf(1,1) 定义了类Point的一个静态对象sf，也是在执行main()函数的代码前，在全局数据区为其分配内存，但不会调用构造函数。

第一次调用fn()函数中，先按照sf.Point (1,1)的语义调用Point的构造函数，初化数静态对象sf，然后执行语句Point f(2,2)，创建局部对象f。在退出fn()函数前，删除局部对象f，具体步骤是，先按照f.~Point()的语义调用析构函数，然后再回收其内存。

第二次调用fn()函数中，因静态局部对象sf已经分配内存并初始化，不再调用Point的构造函数，而是直接执行语句Point f(2,2)创建局部对象f。在退出fn()函数前，也要删除局部对象f。

在退出main()函数前，先删除静态对象sf，然后删除全局对象g。

下面是第二次调用 `fn()` 函数返回前的内存状态描述：

**全局数据区**：在全局数据区中，`g` 对象和`sf` 对象仍然存在，不会在 `fn()` 结束时被销毁。`f` 对象在 `fn()` 结束时被销毁。

总之，在第二次调用 `fn()` 结束后，`sf` 对象仍然存在，但 `f` 对象已被销毁。

**内存状态图**如下附图1.3所示。



附图 1.2





附图1.3

1. 例1.6的程序中使用语句Point \* phA = new Point[5]在堆中创建了一个对象数组，请画出这个对象数组的物理结构。

**答：**在堆中创建的对象数组是连续存储的，下面是这个对象数组的物理结构示意图如附图1.4所示。

在这个示意图1.4中，phA 是指向对象数组第一个元素的指针。数组中的每个元素都是一个 Point 对象，它们按顺序依次存储在内存中。这些对象在堆中是连续分配的，因此可以通过偏移量和指针算术来访问数组中的每个元素。在程序结束时，应该使用 delete[] 来释放整个堆对象数组，以避免内存泄漏。



附图1.4

1. 使用C++语言描述下面的类Desk，并编写出构造函数的代码。如果还有其他熟悉的面向对象，可使用另一种程序设计语言描述类Desk，写出其代码。



图1.29 类Desk

**答：**

#include <iostream>

using namespace std;

class Desk {

private:

double length;

double weight;

double high;

double width;

public:

// 构造函数

Desk(double len, double wei, double h, double wid) : length(len), weight(wei), high(h), width(wid) {

cout << "构造函数被调用" << endl;

}

// 析构函数

~Desk() {

cout << "析构函数被调用" << endl;

}

// 其他可能的方法和操作

// 示例方法：计算体积

double calculateVolume() {

return length \* weight \* high;

}

};

int main() {

// 创建Desk对象的示例

Desk myDesk(1.5, 1.0, 0.8, 1.2);

// 计算并输出体积

cout << "Desk的体积是：" << myDesk.calculateVolume() << endl;

return 0;

}

1. 请根据如下代码，画出类Name的类图。

class Name

{

Name(const char \*myp){

m\_len = strlen(myp);

m\_p = (char\*)malloc(m\_len + 1);

strcpy(m\_p, myp);

}

Name(const Name&obj1){

m\_len = obj1.m\_len;

m\_p = (char\*)malloc(m\_len + 1);

strcpy(m\_p, obj1.m\_p);

}

~Name(){

if (m\_p != NULL){

free(m\_p);

m\_p = NULL;

m\_len = 0;

}

}

char\*m\_p;

int m\_len;

};

**答：**



1. 在网上查阅资料，了解程序设计思想的发展历史，各个发展阶段所使用的主要设计方法以及主流设计工具和程序设计语言，总结面向对象程序设计的特点和主要应用场景。

读者自行完成。

# 封装与职责

1. 2.1节中的类Person，将其所有属性设置为private，不允许从其对象的外部访问对象的成员变量。为了从对象外部访问属性age的值，专门设计了setAge(int newAge) 和getAge()两个成员函数，用于从对象外部存取成员age中的值。增加两个成员函数后的类Person如图2.32所示。



图2.32通过成员函数存取成员的值

请参照上述方法，为类的其他属性设计存取属性值的成员函数，并编程实现。

**答：**

使用PD设计存取属性值的成员函数的方法如下：

在Person类图上双击鼠标，点击【Attritubes】卡片，选择所有属性，如下附图2.1所示：

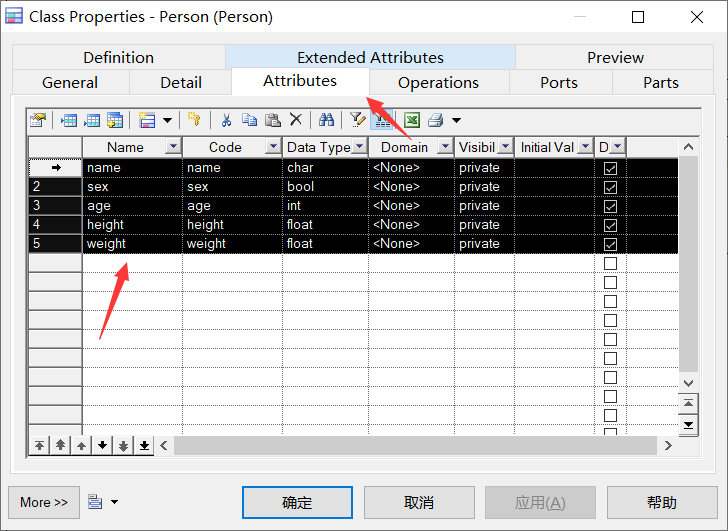
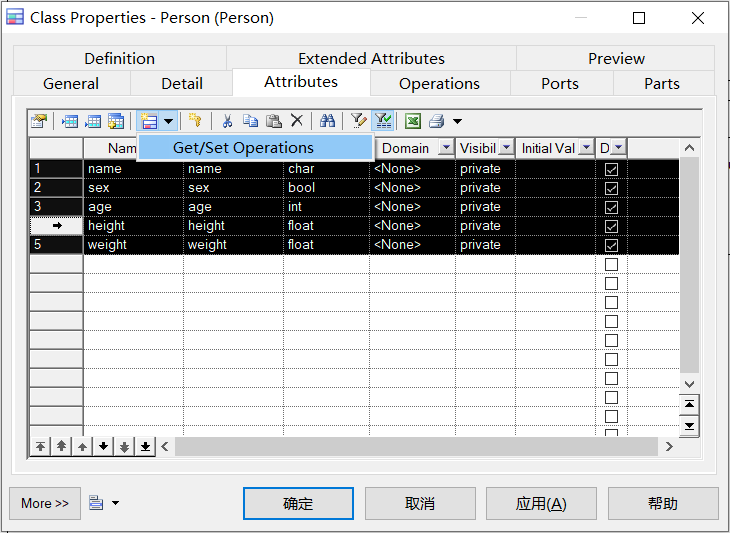


图2.1

点击【add】按钮，选择菜单【Get/Set Operations】即可自动生成存取属性值的成员函数。如附图2.2所示。最终类Person的类图如附图2.3所示。



附图2.2



附图2.3 Person类图

最后通过菜单【Language】\【Generate C++ Code...】生成代码如下（仅相关片段）：

char\* Person::getName(void)

{

return name;

}

void Person::setName(char\* newName)

{

name = newName;

}

bool Person::getSex(void)

{

return sex;

}

void Person::setSex(bool newSex)

{

sex = newSex;

}

int Person::getAge(void)

{

return age;

}

void Person::setAge(int newAge)

{

age = newAge;

}

float Person::getHeight(void)

{

return height;

}

void Person::setHeight(float newHeight)

{

height = newHeight;

}

float Person::getWeight(void)

{

return weight;

}

void Person::setWeight(float newWeight)

{

weight = newWeight;

}

1. 按照如图2.7所示的类Point，编写其声明代码和实现代码，并编写一个main()函数验证直角坐标和极坐标相互转换的功能。

答：



// Point.h

class Point

{

public:

Point();

Point(double dX, double dY);

Point(double dA, double dR, int flag);

double xOffset();

double yOffset();

double angle();

double radius();

~Point();

private:

double x;

double y;

double getX(double dA, double dR);

double getY(double dA, double dR);

double getA(double dX, double dY);

double getR(double dX, double dY);

};

//Point.cpp

#include "Point.h"

#include <math.h>

#define \_USE\_MATH\_DEFINES

#include <iostream>

Point::Point() {

}

Point::Point(double dX, double dY) {

x = dX;

y = dY;

}

Point::Point(double dA, double dR, int flag) {

x = getX(dA, dR);

y = getY(dA, dR);

}

double Point::xOffset() {

return x;

}

double Point::yOffset() {

return y;

}

double Point::angle() {

return getA(x, y);

}

double Point::radius() {

return getR(x, y);

}

Point::~Point() {

}

double Point::getX(double dA, double dR) {

return dR \* cos(dA \* M\_PI / 180);

}

double Point::getY(double dA, double dR) {

return dR \* sin(dA \* M\_PI / 180);

}

double Point::getA(double dX, double dY) {

return (180 / M\_PI)\* atan2(dY, dX);

}

double Point::getR(double dX, double dY) {

return sqrt(dX \* dX + dY \* dY);

}

// app

#include "Point.h"

#include <iostream>

using namespace std;

int main()

{

double x, y;

cout << "输入 x 和 y : \n";

cin >> x >> y;

Point p1(x, y);

cout << "极坐标：Point(" << p1.angle() << "," << p1.radius() << ")" << endl;

cout << "直角坐标：Point(" << p1.xOffset() << "," << p1.yOffset() << ")" << endl;

double a, r;

cout << "输入 a 和 r : \n";

cin >> a >> r;

Point p2(a, r, 1);

cout << "极坐标：Point(" << p2.angle() << "," << p2.radius() << ")" << endl;

cout << "直角坐标：Point(" << p2.xOffset() << "," << p2.yOffset() << ")" << endl;

}

1. 2.2节的例2.5程序中，语句Point b = a使用对象a来创建对象b，请根据如图2.9所示的创建过程画出栈区中数据的变化过程，即根据执行步骤画出内存图，包括栈区中变量、对象的成员及其值。

**答：**



根据这段代码的执行步骤，我们可以绘制出内存图，其中包括栈区中的变量和对象成员以及它们的值。下面是内存图的描述：

开始执行main()时，先为局部对象a、b分配内存，此时成员变量尚未初始化:



执行Point a(1.2, 2.3);对象a被初始化为 (1.2, 2.3)。



执行Point b = a;对象b通过a进行初始化时，调用了拷贝构造函数，将a的值复制给了b。因此，b也包含了相同的数据。



1. 2.3节的图2.23中使用时序图描述了语句fn(d2).print()执行过程，请根据这个执行过程画出内存状态的变化过程，即根据执行步骤画出内存图，包括栈区中变量、对象的成员及其值。

**答：**



对于 `fn(d2).print()` 的执行过程，按照以下步骤来描述内存状态的变化过程：

1. 调用`fn(d2)`之前，main的栈帧中已为局部对象d1和d2分配了内存空间，并通过构造函数初始化d1，通过拷贝构造函数初始化d2：
2. 调用`fn(d2)`：

- 在栈上先后为临时返回对象R和形参对象d分配内存空间，并通过拷贝构造d2去初始化d。

1. `fn` 函数执行：

- 返回d，即以d为参数进行拷贝构造，把d的值复制给R。

1. 析构形参对象d：

- fn()返回后，析构形参对象d。

1. 调用`print()`：

- 调用R的 `print()` 函数。

- 该函数打印出对象的日期信息。

1. 析构R：

- 当表达式执行完毕后，会调用析构函数来释放R的内存空间。

注意，不同的编译器或者同一编译器的不同版本，栈帧中对象的内存分配时间和相对位置是有差别的。





1. 2.3节的示例中没有实现日期相加的逻辑，请按照日期相加的逻辑完善Tdate的add(int id)成员函数代码，并增加一个实现日期减法的成员函数，以扩展类Tdate的功能。

答：

// Tdate.h

class Tdate {

public:

// ……省略教材已有代码

bool isLeapYear(int year) const;

Tdate add(int id);

int daysBetweenDates(const Tdate& secondTdate);

private:

int day;

int month;

int year;

int dayOfYear(const Tdate& Tdate);

int daysInMonth(int year, int month);

};

// Tdate.cpp

#include <iostream>

#include"Tdate.h"

using namespace std;

// ……省略教材已有代码

bool Tdate::isLeapYear(int year) const {

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

}

Tdate Tdate::add(int numDays) {

while (numDays > 0) {

int daysInCurrentMonth = daysInMonth(year, month);

if (day + numDays > daysInCurrentMonth) {

numDays -= (daysInCurrentMonth - day + 1);

day = 1;

if (month == 12) {

month = 1;

year++;

}

else {

month++;

}

}

else {

day += numDays;

numDays = 0;

}

}

Tdate rt(day, month, year);

return rt;

}

// 计算两个日期之间的天数差

int Tdate::daysBetweenDates(const Tdate& secondTdate) {

int totalDays1 = dayOfYear(\*this);

int totalDays2 = dayOfYear(secondTdate);

int yearDiff = secondTdate.year - year;

int dayDiff = totalDays2 - totalDays1;

for (int i = 1; i < yearDiff; ++i) {

dayDiff += isLeapYear(year + i) ? 366 : 365;

}

return dayDiff;

}

// 计算给定日期是该年的第几天

int Tdate::dayOfYear(const Tdate& d) {

int year = d.year, month = d.month, day = d.day;

int daysInMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

if (isLeapYear(d.year)) {

daysInMonth[2] = 29;

}

int dayCount = 0;

for (int i = 1; i < month; ++i) {

dayCount += daysInMonth[i];

}

dayCount += day;

return dayCount;

}

// 计算指定月份的天数

int Tdate::daysInMonth(int year, int month) {

if (month == 2) {

return isLeapYear(year) ? 29 : 28;

}

else if (month == 4 || month == 6 || month == 9 || month == 11) {

return 30;

}

else {

return 31;

}

}

// app

#include <iostream>

#include "TDate.h"

using namespace std;

int main()

{

Tdate d1(2, 10, 2023);

Tdate d2(2, 4, 2023);

cout << d1.daysBetweenDates(d2) << endl;

d1.add(2).print();

}

1. 在2.4节的例2.15示例中，使用了比较复杂的表达式，请分析其中的如下两个表达式的运算序列，并结合程序简述其作用。

employeeArray[i]&&(employeeArray[i]->getCode() == code)

t.getLine(i)->print()

**答：**



这个表达式包含了逻辑与运算符 && 和成员访问运算符 ->。

首先，根据 C++ 运算符优先级规则，成员访问运算符 -> 的优先级比逻辑与运算符 && 的优先级更高。

根据左结合性，&& 运算符是左结合的，这意味着它会首先结合左边的表达式。

因此，根据以上规则，表达式的运算顺序如下：

1. 首先，employeeArray 数组中的第 i 个元素会被访问。
2. 接下来，getCode() 方法会被调用以获取 employeeArray[i] 指针指向对象的 code 属性。
3. 最后，这个 code 属性的值会与变量 code 进行比较。
4. 最终，逻辑与操作符 && 将根据这两个表达式的结果返回一个布尔值。

请注意，在实际执行时，由于 && 是短路运算符，如果 employeeArray[i] 的值为假（即空指针），则后面的表达式不会被执行。这是因为整个表达式只有在两个表达式都为真的情况下才会返回真。



这个表达式包含了成员访问运算符 ->。

根据 C++ 运算符优先级规则，成员访问运算符 -> 的优先级较高。因此，表达式的运算顺序如下：

1. t 对象中的 getLine 方法将被调用，并传入参数 i。
2. getLine 方法将返回某个对象的指针，该对象具有能够调用 print() 方法的结构。
3. print() 方法将被调用，该方法应用于由 getLine(i) 返回的对象。

请注意，这种情况下，必须确保 getLine(i) 返回的是一个可以调用 print() 方法的对象的指针。否则，该表达式会引发运行时错误。

7. 在2.4节的示例中增加查找员工信息的功能，要求根据员工编号查找一个员工信息，如果找到就返回该员工的信息，如果没有找到也要返回员工的信息，其中的员工编号为0。请按照划分职责的一般原则，先确定将查找员工信息的职责赋予哪个类，然后为该类设计相应的成员函数，最后编程实现。

注：书上【例2.15】已实现。

# 关联与连接

1. 饭厅中拟放置一张饭桌和8个凳子，并抽象出了饭厅、饭桌和凳子三个类及其关系，如图3.38所示。请使用集合和映射描述其中的类及其关系。



图3.38饭厅、饭桌和凳子及其关系

**答：**



1. 结合实际情况以完善图3.38中类图，增加描述描述饭厅、饭桌和凳子特性的属性，给每个类赋予适当的职责并增加相应的成员函数，最后编程实现。

**答：**



#include <iostream>

class Desk {

private:

int deskNumber;

public:

// 构造函数

Desk(int number) : deskNumber(number) {}

// 成员函数

void display() const {

std::cout << "Desk " << deskNumber << std::endl;

}

};

class Stool {

private:

int stoolNumber;

public:

// 构造函数

Stool(int number) : stoolNumber(number) {}

// 成员函数

void display() const {

std::cout << "Stool " << stoolNumber << std::endl;

}

};

class DiningRoom {

private:

Desk\* desk;

Stool\* stoolArray[8];

public:

// 构造函数

DiningRoom(Desk\* table, Stool\* stools[]) : desk(table) {

for (int i = 0; i < 8; ++i) {

stoolArray[i] = stools[i];

}

}

// 成员函数

void display() const {

std::cout << "Dining Room Contents:" << std::endl;

desk->display();

for (int i = 0; i < 8; ++i) {

stoolArray[i]->display();

}

}

};

int main() {

// 创建桌子和凳子对象

Desk\* myDesk = new Desk(1);

Stool\* myStools[8];

for (int i = 0; i < 8; ++i) {

myStools[i] = new Stool(i + 1);

}

// 创建饭厅对象并传递桌子和凳子数组

DiningRoom\* myDiningRoom = new DiningRoom(myDesk, myStools);

// 显示饭厅内容

myDiningRoom->display();

// 释放内存

delete myDesk;

for (int i = 0; i < 8; ++i) {

delete myStools[i];

}

delete myDiningRoom;

return 0;

}

**输出结果：**

|  |
| --- |
| Dining Room Contents:  Desk 1  Stool 1  Stool 2  Stool 3  Stool 4  Stool 5  Stool 6  Stool 7  Stool 8 |

1. 3.2节中使用指针表示关联，为了提高代码的安全性，请使用引用来表示关联，并改写其中的例程代码。

**答：**修改【例3.1和例3.2】如下：

// Person.h

class Person

{

public:

// ... 其他函数不变 ...

// 删除Person()

Person(const char \* pName, Sport& sport); // 修改构造函数参数为引用

void like(Sport &likeSport); // 修改参数为引用

Sport& likedSport();// 修改返回类型为引用

// ... 其他函数不变 ...

private:

char name[20];

Sport& sport; // 将指针改为引用

};

// Person.cpp

Person::Person(const char \* pName, Sport& sport) : sport(sport) {

strncpy(name, pName, sizeof(name));

name[sizeof(name)-1] = '\0';

}

void Person::like(Sport &likeSport) {

sport = likeSport; // 使用引用赋值

}

Sport& Person::likedSport() {

return sport;

}

// ... 其他函数不变 ...;

当将 Person 类的构造函数修改为 Person(const char \* pName, Sport& sport) 时，需

要在 app.cpp 中相应地修改对象的创建和构造函数的调用。以下是修改后的 app.cpp：

#include <iostream>

#include "Sport.h"

#include "Person.h"

using namespace std;

int main() {

Sport s1("足球");

Sport s2("篮球");

Sport s3("乒乓球");

// 修改Person的创建方式

Person p1("张三", s1); // 传递Sport对象的引用

p1.print();

cout << "最喜欢";

p1.likedSport().print();

Person p2("李四", s2); // 传递Sport对象的引用

p2.print();

cout << "最喜欢";

p2.likedSport().print();

Person p3("王五", s2); // 传递Sport对象的引用

p3.print();

cout << "最喜欢";

p3.likedSport().print();

return 0;

}

1. 按照如图3.39的类图编写程序。要求使用动态数组存储一个人的姓名，类Person的对象负责管理其中存储的姓名，并编程实现。

其中，char为基本数据类型，charArray为动态字符数组，代码中可以不声明类charArray。



图3.39使用动态数组存储姓名

**答：**

// Tdate.h

#ifndef \_TDATE\_H\_

#define \_TDATE\_H\_

class Tdate {

public:

Tdate();

Tdate(int d, int m, int y);

Tdate(const Tdate &oldTdate);

void print() const;

~Tdate();

Tdate& operator=(const Tdate &v);

private:

int day;

int month;

int year;

};

#endif}

// Tdate .cpp

#include <iostream>

#include"Tdate.h"

using namespace std;

Tdate::Tdate() {

}

Tdate::Tdate(int d, int m, int y) {

day = d;

month = m;

year = y;

cout << "构造：" << "-> ";

print();

}

Tdate::~Tdate() {

cout << "析构：" << "-> ";

print();

}

Tdate::Tdate(const Tdate &oldTdate) {

cout << "拷贝构造：" << "-> ";

oldTdate.print();

memcpy(this, &oldTdate, sizeof(Tdate));

}

void Tdate::print() const {

cout << month << "/" << day << "/" << year << endl;

}

Tdate& Tdate::operator=(const Tdate &v){

cout << "赋值：" << "-> ";

v.print();

memcpy(this, &v, sizeof(Tdate));

return \*this;

}

// Person.cpp

#define \_CRT\_SECURE\_NO\_DEPRECATE

#include <iostream>

#include <string.h>

#include "Tdate.h"

#include <iostream>

using namespace std;

class Person {

public:

Person() {

}

Person(const char n[], Tdate d) {

int len = strlen(n);

name = new char[len + 1];

strncpy(name, n, len + 1);

name[len] = '\0';

birthday = new Tdate(d); //创建一个新的Tdate对象

}

~Person() {

delete name;

delete birthday; //删除连接的Tdate对象

}

void print() {

cout << "Person:" << name << ",";

birthday->print();

}

Tdate getBirthday() const {

return Tdate(\*birthday); //返回一个新对象而没有返回指针birthday，以保证安全

}

Person(const Person& oldPerson) {

memcpy(this, &oldPerson, sizeof(Person)); // 将oldPerson的内存中的数据复制内存

birthday = new Tdate(oldPerson.getBirthday()); //创建一个新的Tdate对象

}

private:

Tdate\* birthday;

char\* name = nullptr;

};

int main() {

Tdate d1(1, 2, 2000);

Tdate d2(1, 2, 2021);

cout << "\*\*\*\*Person对象\*\*\*\*" << endl;

Person p1("Tom", d1);

p1.print();

}

1. 例3.12中创建了类Person的对象p1和p2，请使用对象图描述对象p1和p2的结构及其数据，并使用映射表示对象p1与其数据之间的关系。

答：



1. 例3.12中使用表达式fn(p1).print()检测类Person对象参与计算的能力，请使用时序图描述执行表达式fn(p1).print()的过程。

答：



执行语句fn(p1).print()时，先执行函数调用fn(p1)，返回一个类Person的无名对象R，然后再执行函数调用R.print()，语句执行结束后，再删除无名对象R。返回的无名对象没有名称，为了叙述方便，用一个符号R来表示这个无名对象。语句fn(p1).print()中参数和返回值的传递过程如图所示。

图中比较详细地描述了函数调用fn (p1)的过程。总共分为三个步骤，第一步：先按照Person R的语义创建对象R，用于存储返回值，此时只分配内存，不调用构造函数，然后按照Person p(p1)的语义创建了形参对象p，并将实参对象p1的值传递给形参对象p。第二步：执行函数体中语句return p，按照R.Person(p)的语义调用拷贝构造函数，将返回值存储到对象R中，此时完成了返回函数值。第三步：退出fn()函数返回到main()函数，其中，删除了形参对象p。

1. 例3.13中使用链表管理类Student的对象，请使用时序图描述程序的执行过程。

**答：**main()函数中，总共创建了三个Student对象，构成一个链表，其中，语句new Student("Jenny")在堆中创建了一个无名对象。三个Student对象构成的链表，如下图所示。



语句Student\* pS2 = Student::findname("Jenny")在链表中按照姓名查找， Student::findname("Jenny")返回链表中的无名对象。语句delete pS1删除找到的无名对象，删除对象后的链表如下图所示。



最终程序执行过程的时序图如下图所示。



1. 每学期选修课程时需要先认真阅读本专业的培养方案，分析需要学习的课程以及相关规定，然后再根据自己的具体情况从开设的课程中选择适当的课程，以保证毕业时能够修完规定的课程，取得要求的学分。选课和一般步骤为：第一、根据所学的专业找到培养方案，根据自己的情况从本专业的培养方案中明确所执行的培养方案，确定需要学习的所有课程。第二，根据学习课程的进度从下学期开设课程中选择修读的课程。选课中涉及的专业、培养方案、课程等主要因素及其关系如图3.40所示。



图3.40 选课中涉及主要因素及其关系

1）按照集合和映射等数学思维分析所在学校的本专业培养方案，修改完善如图3.40所示的类图，然后选择关联的表示方式以及属性的数据类型，最终设计出包含类、关联以及属性的类图，以描述选课场景。

2）分析所在学校的选课系统，使用时序图分解其中的核心功能，确定每个类的主要职责，最终设计出成员函数。

3）按照前面的设计编写程序，并调试通过。

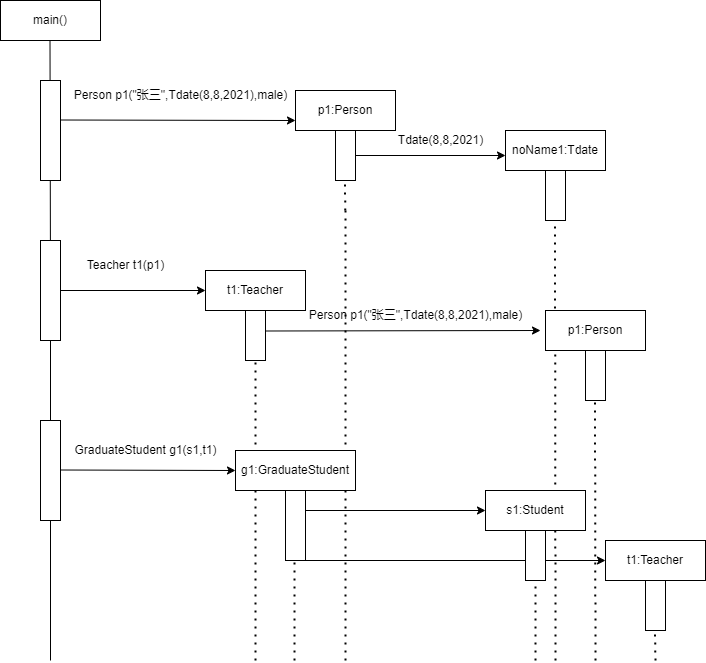
为了便于理解，在设计时可采用中文命名类及属性和方法，在编写代码前再转换为英文。

**答：略。**

# 继承与多态

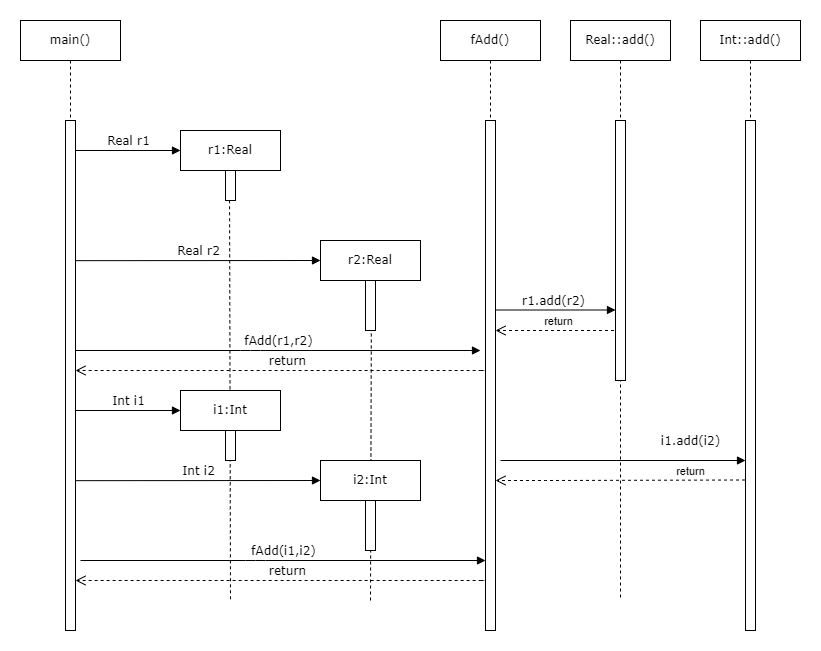
1. 例4.5中创建了p1、t1和g1三个对象，请使用时序图分别描述创建这三个对象的过程。

**答：**时序图如下图所示：



1. 创建 Person 类的对象 p1。main() 调用 Person 类的构造函数。
2. Person 构造函数接收参数 "张三"、Tdate(8, 8, 2021) 和 male。Tdate 类的构造函数被调用以初始化 Person 对象的日期属性。
3. 创建 Teacher 类的对象 t1。main() 调用 Teacher 类的构造函数。Teacher 构造函数接收 Person 类对象 p1 作为参数。Teacher 类的构造函数可能调用 Person 类的复制构造函数来初始化其基类部分。Teacher 对象 t1 完成初始化。
4. 创建 GraduateStudent 类的对象 g1。main() 调用 GraduateStudent 类的构造函数。GraduateStudent 构造函数接收 Student 类对象 s1 和 Teacher 类对象 t1 作为参数。GraduateStudent 类的构造函数首先调用 Student 类的复制构造函数来初始化其 Student 基类部分。接着，GraduateStudent 类可能会使用 Teacher 类对象 t1 来初始化与之相关的属性或成员。GraduateStudent 对象 g1 完成初始化。
5. 例4.7中按照多态调用add()成员函数，请使用时序图描述程序的执行过程，并简述动态联编的原理。

**答：**时序图如下图所示：



首先创建Real 对象 r1,r2 ,调用fAdd(r1,r2) 运行时判断调用Real::add()方法。然后创建Int 对象 i1,i2 ,调用fAdd(i1,i2) 运行时判断调用Int::add()方法。

动态联编，是指被调函数入口地址是在运行时、而不是在编译时决定的。C++ 利用动态联编来完成虚函数调用。C++ 标准并没有规定如何实现动态联编，但大多数的 C++ 编译器都是通过虚指针（vptr）和虚函数表（vtable）来实现动态联编。

基本的思路是：

1. 为每一个包含虚函数的类建立一个虚函数表，虚函数表的每一个表项存放的是个虚函数在内存中的入口地址；
2. 在该类的每个对象中设置一个指向虚函数表的指针，在调用虚函数时，先采用虚指针找到虚函数表，确定虚函数的入口地址在表中的位置，获取入口地址完成调用。
3. 例4.8中类Real 的add()成员函数使用指针以及强制转换指向的数据类型等方法分别实现了整数和实数的加法运算，请使用文字或图形描述其中三个表达式的运算序列，并画出执行语句t = fAdd(r1, r2)过程中的内存变化图。

p1 = (float\*)& digits;

p2 = (float\*)&n;

rt = \*p1 + \*p2;

**答：**首先，逐步分析这三个表达式的运算序列：

1. pl = (float\*)&digits;

①`&digits`：取 `digits` 的地址，得到一个指向 `digits` 的指针。

②`(float\*)`：将上一步得到的指针强制转换为 `float\*` 类型，表示将该指针解释为指向 `float` 类型的数据。

③`pl`：将上一步得到的结果赋给 `pl`，即 `pl` 成为一个指向 `digits` 所在内存位置的 `float` 指针。

1. p2 = (float)&n;

①`&n`：取 `n` 的地址，得到一个指向 `n` 的指针。

②`(float)`：将上一步得到的地址强制转换为 `float` 类型的值。

③`p2`：将上一步得到的结果赋给 `p2`，即 `p2` 成为一个指向 `n` 所在内存位置的 `float` 值。

1. rt = \*p1 + \*p2;

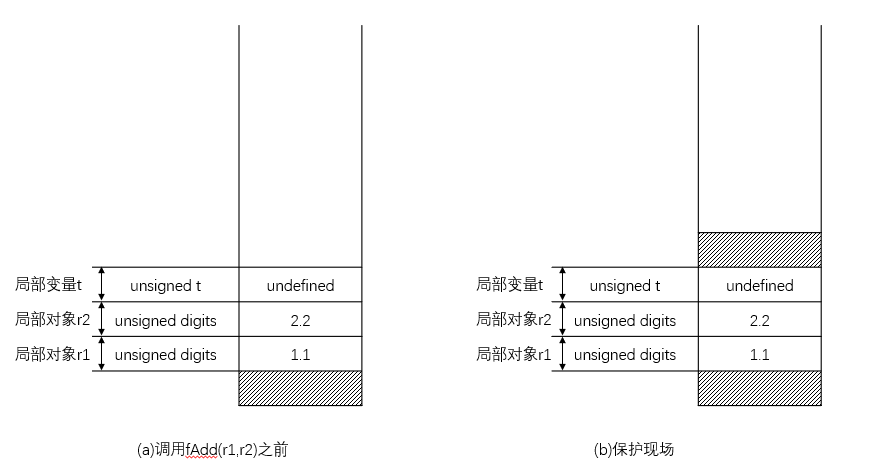
①`\*p1`：解引用指针 `p1`，获取其指向的内存位置的值。

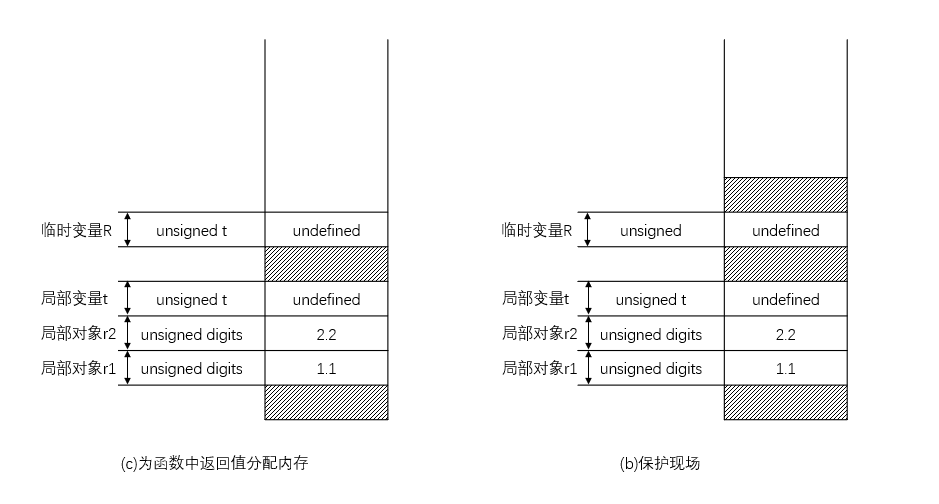
②`\*p2`：解引用指针 `p2`，获取其指向的内存位置的值。

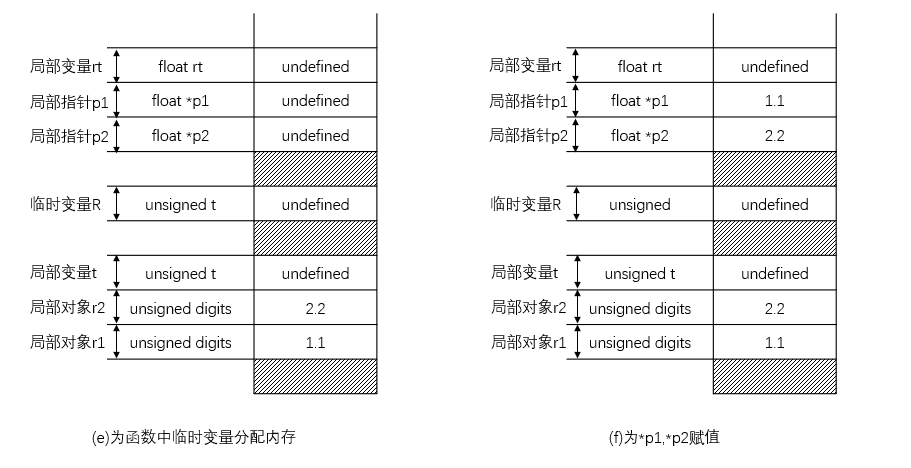
③`\*p1 + \*p2`：将两个解引用得到的值相加。

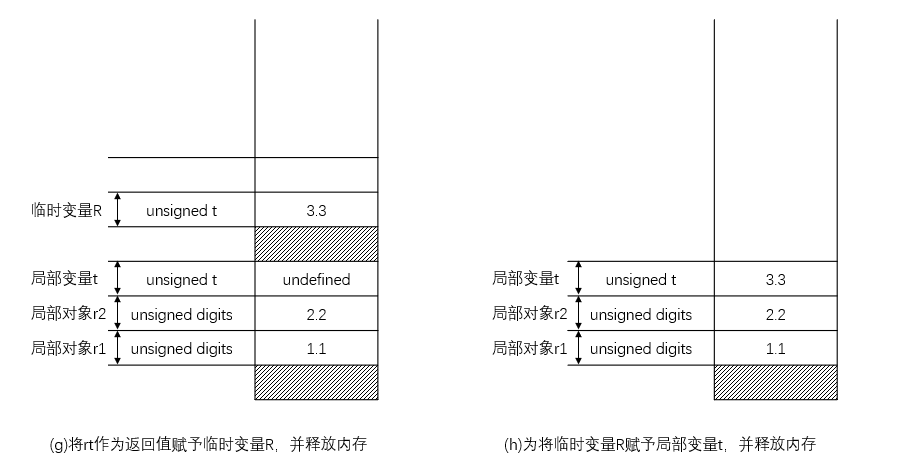
④`rt`：将上一步得到的结果赋给 `rt`，即 `rt` 成为 `p1` 和 `p2` 指向的内存位置的值的和。

然后，分析t=fAdd(r1,r2)的内存变化图如下图所示：









1. 例4.9中可计算圆和矩形的面积，请按照多态思维对程序的功能进行扩充，增加计算梯形和三角形的功能。

**答：** 我们为梯形和三角形创建新的类，这些类将继承自基类 Shape。每个新类将重写 area 方法，以计算其特定形状的面积。

首先，我们定义一个 Trapezoid 类来计算梯形的面积。梯形的面积计算公式为 (a + b) / 2 \* h，其中 a 和 b 是梯形的两个平行边的长度，h 是高。

接着，我们定义一个 Triangle 类来计算三角形的面积。对于一般三角形，我们可以使用海伦公式（Heron's formula）来计算面积。海伦公式使用三角形的三边长度 a、b 和 c：面积 = √[s(s-a)(s-b)(s-c)]，其中 s 是半周长 (a + b + c) / 2。

#include<iostream>

using namespace std;

#include<math.h>

class Shape

{

public:

Shape(double x,double y):xCoord(x),yCoord(y){}

virtual double area() const =0;

protected:

double xCoord,yCoord;

};

class Circle:public Shape

{

public:

Circle(double x,double y,double r):Shape(x,y),radius(r){}

virtual double area() const {

return 3.14\*radius\*radius;

}

protected:

double radius;

};

class Rectangle:public Shape

{

public:

Rectangle(double x1,double y1,double x2,double y2):Shape(x1,y1),x2Coord(x2),y2Coord(y2){}

virtual double area() const;

protected:

double x2Coord,y2Coord;

};

double Rectangle ::area() const

{

return fabs((xCoord - x2Coord)\*(yCoord - y2Coord));

}

**//梯形**

class Trapezoid: public Shape {

public:

Trapezoid(double x1, double y1, double a, double b, double h): Shape(x1, y1), a(a), b(b), h(h) {}

virtual double area() const override {

return (a + b) \* h / 2;

}

protected:

double a, b, h; // a 和 b 是平行边，h 是高

};

**//三角形**

class Triangle: public Shape {

public:

Triangle(double x, double y, double a, double b, double c): Shape(x, y), a(a), b(b), c(c) {}

virtual double area() const override {

double s = (a + b + c) / 2;

return sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

protected:

double a, b, c; // a、b、c 是三边的长度

};

void compute(const Shape& sp){

cout<<sp.area()<<endl;

}

int main(){

Circle c(2.0,5.0,4.0);

compute(c);

Rectangle r(2.0,4.0,1.0,2.0);

compute(r);

**//梯形**

Trapezoid t(0, 0, 3.0, 4.0, 2.0);

compute(t);

**//三角形**

Triangle tr(0, 0, 3.0, 4.0, 5.0);

compute(tr);

}

1. 参照如图4.16所示的类图编写计算学生学费的程序，并上机调试通过。

**答：**程序如下：

#include <iostream>

using namespace std;

class Person

{

protected:

string name;

string birthday;

bool sex;

};

class Teacher :public Person

{

public:

Teacher(string name, string birthday, bool sex) {

this->name = name;

this->birthday = birthday;

this->sex = sex;

}

Teacher(const Teacher& oldTeacher) {

name = oldTeacher.name;

birthday = oldTeacher.birthday;

sex = oldTeacher.sex;

}

};

class Student:public Person

{

public:

Student(string name, string birthday, bool sex) {

this->name = name;

this->birthday = birthday;

this->sex = sex;

}

Student(const Student& oldStudent) {

name = oldStudent.name;

birthday = oldStudent.birthday;

sex = oldStudent.sex;

}

void print() {}

virtual double calcTuition()=0;

};

class Undergraduate :public Student

{

protected:

float credit;

public:

Undergraduate(string name, string birthday, bool sex,float credit):Student(name, birthday, sex) {

this->credit = credit;

}

Undergraduate(const Undergraduate& oldUndergraduate) :Student(oldUndergraduate.name, oldUndergraduate.birthday, oldUndergraduate.sex) {

credit = oldUndergraduate.credit;

}

void print() {

cout << "本科生 "<<name<<" 的学费是：" << calcTuition() <<" 元" <<endl;

}

double calcTuition() {

return credit \* 100;

}

};

class GraduateStudent :public Undergraduate

{

protected:

Teacher\* supervisor;

public:

GraduateStudent(string name, string birthday, bool sex, Teacher\* supervisor) :Undergraduate(name,birthday,sex,0) {

this->supervisor = supervisor;

}

GraduateStudent(const GraduateStudent& oldGraduateStudent) :Undergraduate(oldGraduateStudent) {

supervisor = oldGraduateStudent.supervisor;

}

void print() {

cout << "研究生 " << name << " 的学费是：" << calcTuition() << " 元" << endl;

}

double calcTuition() {

return 10000.0;

}

};

int main()

{

Undergraduate undergraduate("蔡徐坤", "1998.08.02", false, 2.5);

Teacher supervisor("张艺兴", "1991.10.07", true);

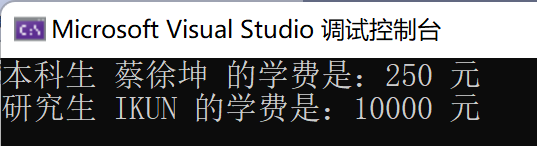
GraduateStudent graduateStudent("IKUN", "1998.08.02", false, &supervisor);

undergraduate.print();

graduateStudent.print();

}

最终运行结果如下图所示：



1. 参照如图4.29所示的类图改写例4.11所示的程序，并上机调试通过。

**答：**程序如下：

//Linkage.h

class Linkage {

public:

Linkage() = default;

~Linkage();

int insert();

protected:

static int count;

Linkage\* pNext;

static Linkage\* pFirst;

};

//linkage.cpp

#include "Linkage.h"

int Linkage::count = 0;

Linkage \*Linkage::pFirst = nullptr;

int Linkage::insert() {

count++;

pNext = pFirst; //每新建一个结点(对象)就将其挂在链首

pFirst = this;

}

Linkage::~Linkage() {

if(pFirst == this) { //如果要删除链首结点,则只要链首指针指向下一个

pFirst = pNext;

return;

}

for (Linkage \*pS = pFirst; pS; pS = pS->pNext) {

if (pS->pNext == this) {

pS->pNext = pNext; //pNext 即his->pNext

return;

}

}

}

//Account.h

#include "Linkage.h"

class Account:public Linkage{

public:

float Deposit(float amount);

virtual float Withdrawal(float amount) = 0;

unsigned accountNo(void);

float acntBalance(void);

Account(unsigned accNo, float balan);

~Account();

protected:

unsigned acntNumber;

float balance;

};

//Account.cpp

#include "Account.h"

#include <iostream>

using namespace std;

float Account::Deposit(float amount) {

balance += amount;

return balance;

}

unsigned Account::accountNo(void) {

return acntNumber;

}

float Account::acntBalance(void) {

return balance;

}

Account::Account(unsigned accNo, float balan) {

acntNumber = accNo;

balance = balan;

insert();

cout << "账户数：" << count << "，增加" << this->acntNumber << endl;

}

Account::~Account() {

count--;

cout << "账户数：" << count << "，删除" << this->acntNumber << endl;

}

//Saving.h

#include "Account.h"

class Saving:public Account {

public:

virtual float Withdrawal(float amount);

Saving(unsigned accNo, float balan);

};

//Saving.cpp

#include "Saving.h"

#include <iostream>

using namespace std;

float Saving::Withdrawal(float amount) {

if(balance < amount)

cout << "Insufficient funds : balance" << balance

<< ", withdrawal " << amount << endl;

else

balance -= amount;

return balance;

}

Saving::Saving(unsigned accNo, float balan): Account(accNo, balan){}

//Checking.h

#include "Account.h"

enum Remit{

remitByPost,

remitByCable,

other

};

class Checking:public Account {

public:

virtual float Withdrawal(float amount);

void SetRemit(Remit re);

Checking(unsigned accNo, float balan);

protected:

Remit remittance;

};

//Checking.cpp

#include "Checking.h"

#include "Account.h"

#include <iostream>

using namespace std;

float Checking::Withdrawal(float amount) {

float temp = amount;

if (remittance == remitByPost)

temp = amount + 30;

else if (remittance == remitByCable)

temp = amount + 60;

if(balance < temp)

cout << "Insuficient funds : balance " << balance

<< ", withdrawal " << temp << endl;

else

balance -= temp;

return balance;

}

void Checking::SetRemit(Remit re) {

remittance = re;

}

Checking::Checking(unsigned accNo, float balan) : Account(accNo, balan) {

remittance = other;

}

//app.cpp

#include "Checking.h"

#include "Saving.h"

#include <iostream>

using namespace std;

int main() {

cout << "\*\*\*\*\*\*\*\*\*\*构造 Saving 对象\*\*\*\*\*\*\*\*\*\*" << endl;

Saving s1(100, 1000), s2(101, 1000);

cout << "\*\*\*\*\*\*\*\*\*\*构造 Checking 象\*\*\*\*\*\*\*\*\*\*" << endl;

Checking c1(200, 1000);

Checking \*p = new Checking(201, 1000);

cout << "\*\*\*\*\*\*\*\*\*\*Saving存取款\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "存100后的余额:" << s1.Deposit(100) << endl;

cout << "取200后的余额:" << s1.Withdrawal(200) << endl;

cout << "\*\*\*\*\*\*\*\*\*\*Checking 存取款\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "存500后的余额:" << p->Deposit(500) << endl;

cout << "取300后的余额:" << p->Withdrawal(300) << endl;

cout << "\*\*\*\*\*\*\*\*\*\*删除堆中的对象\*\*\*\*\*\*\*\*\*\*" << endl;

delete p;

cout << "\*\*\*\*\*\*\*\*\*\*退出 main()函数\*\*\*\*\*\*\*\*\*\*" << endl;

return 0;

}

# 设计与实现

1. 数组实现Josephus游戏

Josephus游戏示例中，选择了链表存储圆圈中的小孩。如果将链表改为对象数组，选择对象数组存储圆圈中的小孩，请重新描述在对象数组上玩Josephus游戏的场景，并使用类图设计出程序的静态模型，最后编程实现。

**答：**

1. 构成圆圈：通过对象数组 (下标+1) mod numOfBoys解决。
2. 小孩离开:在Boy中成员in表示该小孩是否在圆圈中，true表示在圆圈中，false表示离开圆圈。
3. 以下是分析设计过程。



数组上玩Josephus游戏的场景



类与关联



对象的职责与协作

下面是对代码的修改，将链表改为对象数组的实现：

// Boy.h

#ifndef BOY\_H\_

#define BOY\_H\_

class Boy {

public:

Boy();

void print();

int getCode() ;

void setCode(int code);

bool isIn() ;

void leave(bool in);

private:

int code;

bool in;

};

// Boy.cpp

#include "Boy.h"

#include <iostream>

Boy::Boy() {

code = 0; //小孩编号

in = 1; //小孩是否在圆圈中，1表示在，0表示不再

}

void Boy::print() { //输出小孩编号

std::cout << "Id:" << code;

}

int Boy::getCode() { //获取当前小孩编号

return code;

}

void Boy::setCode(int code) { //设置当前小孩编号

this->code = code;

}

bool Boy::isIn() { //小孩是否在圆圈中

return in;

}

void Boy::leave(bool in) { //小孩离开

this->in = in;

}

// Ring.h

#ifndef RING\_H\_

#define RING\_H\_

#include "Boy.h"

class Ring {

public:

Ring(int n);

~Ring();

Boy getWinner(int m);

private:

Boy\* boys;

int numOfBoys;

void next(int\* i); //调整到下一个小孩

void countUpTo(int \*i,int m); //i当前小孩，m间隔数

};

#endif /\* RING\_H\_ \*/

// Ring.cpp

#include "Ring.h"

#include <iostream>

using namespace std;

Ring::Ring(int n) {

numOfBoys = n;

boys = new Boy[n];

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

boys[i].setCode(i + 1);

cout << "第" << i + 1 << "个小孩," << endl;

}

}

Ring::~Ring() {

delete[] boys;

}

Boy Ring::getWinner(int m) {

int n = numOfBoys;

int i = 0;

while (n > 1) {

//数m个小孩

countUpTo(&i, m);

//第m个小孩离开

cout << "第"<<boys[i].getCode()<<"小孩离开" << ",";

boys[i].leave(false);

//当前位置i调整到下一个小孩

next(&i);

n--;

}

//返回获胜者

Boy\* ptr = &boys[0];

for (int i = 0; i < numOfBoys + 1; i++,ptr++) {

if (boys[i].isIn()) {

return \*ptr;

}

}

}

void Ring::next(int\* i) { //i 当前小孩

//当前位置i调整到下一个小孩

\*i = (\*i + 1) % numOfBoys;

while (boys[\*i].isIn() == 0) //判断该小孩是否离开

{

\*i = (\*i + 1) % numOfBoys;

}

}

void Ring::countUpTo(int \*i,int m){

while (m > 1)

{

//当前位置i调整到下一个小孩

next(i);

m--;

}

}

// Jose.h

#ifndef JOSE\_H\_

#define JOSE\_H\_

#include "Ring.h"

class Jose {

public:

Jose(int boys, int interval);

~Jose();

Boy gameBegin();

private:

int numOfBoys;

int m;

Ring \*ring;

Boy \*win;

};

#endif /\* JOSE\_H\_ \*/

// Jose.cpp

#include "Jose.h"

Jose::Jose(int boys, int interval) {

numOfBoys = boys;

m = interval;

ring = new Ring(boys);

win = nullptr;

}

Jose::~Jose() {

delete ring;

delete win;

}

Boy Jose::gameBegin() {

if (!win) {

win = new Boy(ring->getWinner(m));

}

return \*win;

}

//app.cpp

#include "Jose.h"

#include <iostream>

using namespace std;

int main() {

Jose js1(7, 3);

js1.gameBegin().print();

int m, n;

cout << endl << "请输入小孩数和间隔数......" << endl;

cin >> n >> m;

Jose js2(n, m);

js2.gameBegin().print();

cout << endl << "现在宣布" << endl << "第一场获胜者是：" << endl;

js1.gameBegin().print();

cout << endl << "第二场获胜者是：" << endl;

js2.gameBegin().print();

system("pause");

}

2.请在Josephus游戏示例中增加异常处理功能。

**答：**参照例5.9设计异常处理类，如下图所示，增加JoseErr类，处理小孩数、间隔数初始不符合要求等异常。



JoseErr 类声明可以写在例5.9所示的MyErr.h文件中，实现这个异常放在MyErr.cpp中。下面是一个可能的实现：

# MyErr.h文件中增加JoseErr声明

## MyErr.h文件中

class JoseErr : public LogicErr {

public:

JoseErr(const char \*msg);

};

接着在MyErr.cpp中，实现这个异常类的构造函数：

JoseErr::JoseErr(const char \*msg) :LogicErr(21, msg) {

}

然后，在 Jose 类中，可以在需要时抛出这个异常。例如，在 Jose::Jose中检查输入是否有误；Jose::gameBegin()、Ring::getWinner 函数中，可以检查是否有系统异常发生，如果有异常，则抛出MemErr。在 main 函数中，可以捕获并处理这个异常。

// 在Ring类的Ring::getWinner函数中添加异常处理

Boy Ring::getWinner(int m) {

//数小孩

while (pCurrent != pCurrent->next()) {

countUpTo(m); //往下数m个小孩，数到的小孩离开

}

if (!pCurrent){

throw MemErr("申请内存失败！");

}

//返回获胜者

Boy win(\*pCurrent); //另外创建(复制)一个Boy对象

delete pCurrent;

return win;

}

// 在 Jose类的Jose、gameBegin函数中添加异常处理

#include "Jose.h"

#include "MyErr.h"

#include<iostream>

using namespace std;

Jose::Jose(int boys, int interval) {

if (boys < 1 || interval < 1)

{

throw JoseErr("输入有错，请重新输入");

}

numOfBoys = boys;

m = interval;

ring = new Ring(boys);

win = nullptr;

}

Jose::~Jose() {

delete ring;

delete win;

}

Boy Jose::gameBegin() {

try

{

if (!win)

win = new Boy(ring->getWinner(m));

return \*win;

}

catch (MemErr e){

cerr << "错误号:" << e.getErrNo() << "\t系统错误号:" << e.getSysErrNo()<< "\t错误信息:" << e.getErrMsg().getString();

throw MemErr("申请内存失败！");

}

}

// 在主函数中增加 try-catch 块来捕获异常

#include "MyErr.h"

int main() {

try{

Jose js1(7, 3);

js1.gameBegin().print();

int m, n;

cout << endl << "请输入小孩数和间隔数......" << endl;

cin >> n >> m;

Jose js2(n, m);

js2.gameBegin().print();

cout << endl << "现在宣布" << endl << "第一场获胜者是：" << endl;

js1.gameBegin().print();

cout << endl << "第二场获胜者是：" << endl;

js2.gameBegin().print();

}

catch (JoseErr e) {

cerr << "错误号:" << e.getErrNo() << "\t逻辑错误号:" << e.getLogErrNo()

<< "\t错误信息:" << e.getErrMsg().getString();

}

catch (MemErr e) {

cerr << "错误号:" << e.getErrNo() << "\t系统错误号:" << e.getSysErrNo()

<< "\t错误信息:" << e.getErrMsg().getString();

}

system("pause");

}

这样，如果在 Ring::getWinner 函数中发生了 MemErr 异常，它将抛出MemErr 异常，并在 main 函数中输出相应的错误信息。

1. Josephus游戏中，要求决出金牌、银牌和铜牌，请重新设计程序的静态模型和动态模型，并编程实现。

**答：**该题可采用数组实现，所有静态模型和动态模型与该章第一题一致，实现代码与第一题仅Boy Ring::getWinner(int m)，void Boy::print()，函数有所变化，Boy Ring::getWinner(int m)中增加了离开次序判断，并输出获得的成绩。以下是Boy Ring::getWinner(int m)，void Boy::print()函数参考代码：

Boy Ring::getWinner(int m) {

int n = numOfBoys;

int i = 0;

int j = 1; //离开次序

while (n > 1) {

//数m个小孩

countUpTo(&i, m);

if (n == 3)

{

//第m个小孩离开

cout << "第" << boys[i].getCode() << "小孩第" << j++ << "个离开，获得铜牌" << "," << endl;

boys[i].leave(false);

}

else if (n == 2)

{

//第m个小孩离开

cout << "第" << boys[i].getCode() << "小孩第" << j++ << "个离开，获得银牌" << "," << endl;

boys[i].leave(false);

}

else

{

//第m个小孩离开

cout << "第" << boys[i].getCode() << "小孩第"<<j++<<"个离开" << "," << endl;

boys[i].leave(false);

}

//当前位置i调整到下一个小孩

next(&i);

n--;

}

//返回金牌获胜者

Boy\* ptr = &boys[0];

for (int i = 0; i < numOfBoys + 1; i++, ptr++) {

if (boys[i].isIn()) {

return \*ptr;

}

}

}

void Boy::print() { //输出小孩编号

std::cout << "第" << code<<"小孩获得金牌";

}

1. 组织小学生玩Josephus游戏，希望公布哪个班级的哪个同学取得了胜利，请重新设计出程序的静态模型和动态模型，并编程实现。

**答：**参照教材5.1.3小节，图5.14所示静态模型Person类中增加classname数据成员,构造函数参数增加pClass，动态模型不变（教材图5.15），如下图所示。



Person类构造函数、拷贝构造函数、打印函数代码相应修改，其他代码参照教材5.1.3小节。Person类以及构造函数、拷贝构造函数、打印函数代码如下所示：

// Person.h

#ifndef PERSON\_H\_

#define PERSON\_H\_

#include "MyString.h"

#include "Tdate.h"

class Person {

public:

Person(const char\* pClass, const char \*pName, const Tdate &tBirthday, bool bSex);

Person(const Person &oldPerson);

void print();

protected:

MyString classname;

MyString name;

Tdate birthday;

bool sex;

};

#endif /\* PERSON\_H\_ \*/

Person::Person(const char\* pClass, const char\* pName, const Tdate& tBirthday, bool bSex) :classname(pClass),

name(pName), birthday(tBirthday) {

sex = bSex;

}

Person::Person(const Person& oldPerson) :

classname(oldPerson.classname), name(oldPerson.name), birthday(oldPerson.birthday) {

sex = oldPerson.sex;

}

void Person::print() {

cout << "班级号:";

classname.print();

cout << " , 姓名:";

name.print();

cout << " , 生日:";

birthday.print();

cout << " , 性别:" << (sex ? "男" : "女");

}

1. 矩阵计算是大数据智能化的基础，5.2节的示例中只介绍了矩阵和向量的乘法，请扩展示例程序的功能，编程实现矩阵的乘法、加法、减法和转置运算。

**答：**

矩阵运算主要有加减乘除等， **矩阵乘法**的一般形式为

Cmn=Amk×Bkn

其中，矩阵C为m行n列的矩阵，其元素的通项公式为

矩阵A乘以B得到C，必须满足如下规则：

（1）矩阵A的列数与矩阵B的行数相等；

（2）矩阵A的行数等于矩阵C的行数；

（3）矩阵B的列数等于矩阵C的列数。

**矩阵加、减法**的一般形式为

Cmn=AmnBmn

其中，矩阵C为m行n列的矩阵，其元素的通项公式为

矩阵A加减B得到C，必须满足如下规则：

（1）矩阵A、B、C的行数相等；

（2）矩阵A、B、C的列数相等。

1. **方法一 矩阵成员函数实现**

将乘法、加法、减法和转置运算设计成矩阵的成员函数，类图如下：



乘法、加法、减法和转置运算实现代码如下所示：

#ifndef MATRIX\_H\_

#define MATRIX\_H\_

class Matrix {

public:

Matrix Matrix::Multiply(Matrix &mat); //成员函数

Matrix Matrix::Add(Matrix &mat); //加法

Matrix Matrix::Minus(Matrix &mat); //减法

Matrix Matrix::Transpose(); //转置

int& Elem(int i, int j);

Matrix(int m, int n);

Matrix(const Matrix &oldMatrix);

~Matrix();

void print();

int getM();

int getN();

private:

int m;

int n;

int \*Mat;

};

#endif /\* MATRIX\_H\_ \*/

#include "Matrix.h"

#include <iostream>

using namespace std;

int& Matrix::Elem(int i, int j) {

if (i < m && j < n)

return Mat[i \* n + j]; //计算第i行第j列元素的位置

else

cout << "超标越界！！";

}

Matrix::Matrix(int m, int n) {

Matrix::m = m;

Matrix::n = n;

Mat = new int[m \* n];

}

Matrix::~Matrix() {

delete[](int\*) Mat;

}

void Matrix::print() {

for (int i = 0; i < m; i++) {

cout << Mat[i \* n + 0];

for (int j = 1; j < n; j++) {

cout << "\t" << Mat[i \* n + j];

}

cout << endl;

}

}

int Matrix::getM() {

return m;

}

int Matrix::getN() {

return n;

}

Matrix::Matrix(const Matrix &oldMatrix) {

m = oldMatrix.m;

n = oldMatrix.n;

Mat = new int[m \* n];

for (int i = 0; i < m \* n; i++)

Mat[i] = oldMatrix.Mat[i];

}

Matrix Matrix::Multiply(Matrix &mat){ //矩阵乘法

Matrix c(this->getM(),mat.getN());

for (int i = 0; i < m; i++) {

for (int j = 0; j < mat.getN(); j++) {

c.Elem(i, j) = 0;

for (int k = 0; k < this->getN();k++)

c.Elem(i, j) += Mat[i\*n + k] \* mat.Mat[k\*mat.getN() + j];

}

}

return c;

}

Matrix Matrix::Add(Matrix &mat) //加法

{

Matrix c(m, n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

c.Elem(i, j) = Mat[i\*n + j] + mat.Mat[i\*n + j];

}

}

return c;

}

Matrix Matrix::Minus(Matrix &mat) //减法

{

Matrix c(m,n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

c.Elem(i, j) = Mat[i\*n + j] - mat.Mat[i\*n + j];

}

}

return c;

}

Matrix Matrix::Transpose() //转置

{

Matrix c(n, m);

for (int j = 0; j < n; j++) {

for (int i = 0; i < m; i++) {

c.Elem(j, i) = Mat[i\*n + j] ;

}

}

return c;

}

#include "Matrix.h"

#include <iostream>

using namespace std;

void main() {

int m = 3, n = 4;

cout << "矩阵：" << endl;

Matrix a(m, n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

a.Elem(i, j) = (i + 1) \* 10 + j + 1;

}

}

a.print();

cout << "转置矩阵：" << endl;

a.Transpose().print();

Matrix c(a);

cout << "矩阵加法：" << endl;

a.Add(c).print();

cout << "矩阵减法：" << endl;

a.Minus(c).print();

cout << "请输入令一矩阵行列值：" << endl;

cin >> m >> n;

cout << "矩阵：" << endl;

Matrix b(m, n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

b.Elem(i, j) = (i + 1) \* 1 + j + 1;

}

}

b.print();

cout << endl << "矩阵×矩阵：" << endl;

a.Multiply(b).print(); //Matrix的成员函数声明为Vector友元

system("pause");

}

1. **方法二 运算符重载实现**

// Matrix.h

#ifndef MATRIX\_H\_

#define MATRIX\_H\_

class Matrix {

public:

// 获取矩阵元素

int& Elem(int i, int j);

// 构造函数，创建m行n列的矩阵

Matrix(int m, int n);

// 拷贝构造函数

Matrix(const Matrix& oldMatrix);

// 析构函数，释放内存

~Matrix();

// 打印矩阵内容

void print();

// 获取矩阵行数

int getM();

// 获取矩阵列数

int getN();

// 重载加法运算符

Matrix operator+(const Matrix& otherMatrix);

// 重载减法运算符

Matrix operator-(const Matrix& otherMatrix);

// 重载乘法运算符

Matrix operator\*(const Matrix& otherMatrix);

// 转置矩阵

Matrix transpose();

private:

int m; // 矩阵行数

int n; // 矩阵列数

int \*Mat; // 存储矩阵元素的数组

};

#endif /\* MATRIX\_H\_ \*/

// Matrix.cpp

#include "Matrix.h"

#include <iostream>

using namespace std;

// 获取矩阵元素的引用

int& Matrix::Elem(int i, int j) {

if (i < m && j < n)

return Mat[i \* n + j];

else {

cout << "索引越界！" << endl;

exit(EXIT\_FAILURE);

}

}

// 构造函数

Matrix::Matrix(int m, int n) {

Matrix::m = m;

Matrix::n = n;

Mat = new int[m \* n];

}

// 析构函数

Matrix::~Matrix() {

delete[] Mat;

}

// 打印矩阵内容

void Matrix::print() {

for (int i = 0; i < m; i++) {

cout << Mat[i \* n];

for (int j = 1; j < n; j++) {

cout << "\t" << Mat[i \* n + j];

}

cout << endl;

}

}

// 获取矩阵行数

int Matrix::getM() {

return m;

}

// 获取矩阵列数

int Matrix::getN() {

return n;

}

// 拷贝构造函数

Matrix::Matrix(const Matrix& oldMatrix) {

m = oldMatrix.m;

n = oldMatrix.n;

Mat = new int[m \* n];

for (int i = 0; i < m \* n; i++)

Mat[i] = oldMatrix.Mat[i];

}

// 重载加法运算符

Matrix Matrix::operator+(const Matrix& otherMatrix) {

if (m != otherMatrix.m || n != otherMatrix.n) {

cout << "矩阵维度不匹配，无法相加！" << endl;

exit(EXIT\_FAILURE);

}

Matrix result(m, n);

for (int i = 0; i < m \* n; i++)

result.Mat[i] = Mat[i] + otherMatrix.Mat[i];

return result;

}

// 重载减法运算符

Matrix Matrix::operator-(const Matrix& otherMatrix) {

if (m != otherMatrix.m || n != otherMatrix.n) {

cout << "矩阵维度不匹配，无法相减！" << endl;

exit(EXIT\_FAILURE);

}

Matrix result(m, n);

for (int i = 0; i < m \* n; i++)

result.Mat[i] = Mat[i] - otherMatrix.Mat[i];

return result;

}

// 重载乘法运算符

Matrix Matrix::operator\*(const Matrix& otherMatrix) {

if (n != otherMatrix.m) {

cout << "矩阵维度不匹配，无法相乘！" << endl;

exit(EXIT\_FAILURE);

}

Matrix result(m, otherMatrix.n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < otherMatrix.n; j++) {

result.Elem(i, j) = 0;

for (int k = 0; k < n; k++) {

result.Elem(i, j) += Elem(i, k) \* otherMatrix.Elem(k, j);

}

}

}

return result;

}

// 转置矩阵

Matrix Matrix::transpose() {

Matrix result(n, m);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

result.Elem(j, i) = Elem(i, j);

}

}

return result;

}

1. 根据下面的描述和要求设计并编写程序。

一个四口之家，大家都知道其父亲会开车，母亲会唱歌。但是其父亲还会修电视机，只有家里人知道。小孩既会开车又会唱歌甚至也会修电视机。母亲瞒着任何人在外面做小工以补贴家用。此外男孩还会打乒乓球，女孩还会跳舞。

主程序中描述这四口之家一天的活动：先是父亲出去开车，然后母亲出去工作（唱歌），母亲下班后去做两小时小工。小孩在俱乐部打球，在父亲回家后，开车玩，后又高兴地唱歌。晚上，两个小孩和父亲一起修电视机。

后来父亲的修电视机技术让大家知道了，人们经常上门要他修电视机。这时，程序要做什么样的变动。

**答：**



使用了类的继承和多态性质，通过继承实现了不同成员的特定功能，上图中Father类driveCar、repairTV、Mother类的singSong函数都为虚函数。

下面是根据描述和要求设计的C++程序。

#include <iostream>

using namespace std;

class Father

{

public:

virtual void driveCar(){

cout << "父亲出去开车" << endl;

}

virtual void repairTV(){

cout << "父亲修电视机" << endl;

}

};

class Mother

{

public:

virtual void singSong(){

cout << "母亲出去工作（唱歌）" << endl;

}

void doMannulWork(){

cout << "母亲下班后去做两小时小工" << endl;

}

};

class Boy : public Father, public Mother

{

public:

void driveCar() override{

cout << "男孩开车" << endl;

}

void repairTV() override {

cout << "男孩修电视机" << endl;

}

void singSong() override {

cout << "男孩唱歌" << endl;

}

void playPingpang(){

cout << "小孩在俱乐部打球" << endl;

}

};

class Girl : public Father, public Mother

{

public:

void repairTV() override {

cout << "女孩修电视机" << endl;

}

void singSong() override {

cout << "女孩唱歌" << endl;

}

void dance(){

cout << "女孩在俱乐部跳舞" << endl;

}

};

void main() {

Father father;

Mother mother;

Boy boy;

Girl girl;

father.driveCar();

mother.singSong();

mother.doMannulWork();

boy.playPingpang();

boy.driveCar();

boy.singSong();

father.repairTV();

boy.repairTV();

girl.repairTV();

}

这个程序创建了一个家庭的模型，包括父亲、母亲、男孩和女孩。每个类都有特定的行为，并且在主程序中按照要求描述了他们一天的活动。程序通过多态性质实现了不同成员的特定功能，并展示了类的继承关系。

7.第3章习题8要求针对大学生的选课场景设计并实现一个选课程序，请按照5.1节中的步骤和方法重新进行设计并实现。

**答：**参照第3章自行完成。

# 运算与重载

1. 按照如图6.21所示的类图对例6.8程序进行了扩展，编程实现其中的类RUB和Dollar，并编写一个主函数验证其正确性。

**答：参考代码如下**：

**//Currency.h**

#include <iostream>

using namespace std;

class Currency {

public:

Currency& operator ++();

Currency operator ++(int i);

operator double();

Currency& operator =(Currency v);

Currency operator +(Currency b);

Currency operator \*(int b);

Currency(int y, int f);

Currency(double d);

Currency();

virtual ~Currency();

virtual ostream& print(ostream &out);

virtual istream& input(istream &inp);

protected:

unsigned int yuan;

unsigned char fen;

};

Currency& Currency::operator++() {

int c;

fen++;

c = fen / 100;

fen %= 100;

yuan += c;

return \*this; //返回对象(变量)

}

Currency Currency::operator++(int) {

Currency rt(\*this); //保存对象原来的值

int c;

fen++;

c = fen / 100;

fen %= 100;

yuan += c;

return rt; //返回对象原来的值

}

Currency::operator double() {

return double(yuan + (double) fen / 100);

}

Currency& Currency::operator =(Currency v) {

yuan = v.yuan;

fen = v.fen;

return \*this;

}

Currency Currency::operator +(Currency b) {

int y, f;

f = fen + b.fen;

y = yuan + b.yuan + f / 100;

return Currency(y, f % 100);

}

Currency Currency::operator \*(int b) {

int y, f;

f = fen \* b;

y = yuan \* b + f / 100;

return Currency(y, f % 100);

}

Currency::Currency(double d) {

int t = (d + 0.005) \* 100;

fen = t % 100;

yuan = t / 100;

}

Currency::Currency(int y, int f) {

yuan = y + f / 100;

fen = f % 100;

}

Currency::Currency() {

}

Currency::~Currency() {

}

ostream& Currency::print(ostream &out) {

out << yuan << "元" << (int) fen % 100 << "分";

return out;

}

istream& Currency::input(istream &inp) {

int y, f;

inp >> y >> f;

yuan = y;

fen = f % 100;

return inp;

}

ostream& operator<<(ostream &out, Currency &c) {

c.print(out);

return out;

}

istream& operator>>(istream &inp, Currency &c) {

c.input(inp);

return inp;

}

#include "currency.h"

class Dollar:public Currency

{

public:

Dollar& operator= (const Currency v){

(Currency)(\*this) = v;

return \*this;

}

ostream& print(ostream& out){

Currency::print(cout);

cout << "美元";

return out;

}

Dollar(int y, int f):Currency(y,f){}

Dollar(double d):Currency(d){}

Dollar(){}

};

class RUB:public Currency

{

public:

RUB& operator= (const Currency v){

(Currency)(\*this) = v;

return \*this;

}

ostream& print(ostream& out){

Currency::print(cout);

cout << "卢布";

return out;

}

istream& input(istream& inp){

int kopeks, rub;

inp >> rub >> kopeks;

yuan = rub + kopeks / 100;

fen = kopeks % 100;

return inp;

}

RUB(int y, int f):Currency(y,f){}

RUB(double d):Currency(d){}

RUB(){}

};

void main()

{

const double d=2.4;

Dollar r1(1,23), r2(d),r3;

r1.print(cout);

cout << r2 << endl;

r3=r2;

cout << r3 << endl;

r1++;

cout << r1 << endl;

++r2;

cout << r2 << endl;

r3 = r1 + r2;

cout << r3 << endl;

r3 = r1 \* 3;

cout << r3 << endl;

RUB r4(1,23), r5(d),r6;

r4.print(cout);

cout << r5 << endl;

r6=r5;

cout << r6 << endl;

r4++;

cout << r4 << endl;

++r5;

cout << r5 << endl;

r6 = r4 + r5;

cout << r6 << endl;

r6 = r4 \* 3;

cout << r6 << endl;

cin >> r6;

cout << r6 << endl;

}

1. 对例6.8程序进行扩展，定义并重载货币到double的类型转换、货币与实数的乘法运算，并编写一个主函数验证其正确性。为了提高计算精度，类Currency中可使用一个实数来表示货币的值。

**答**：main函数代码详见第1题，类Currency改写参考代码如下：

class Currency

{

public:

operator double(){

return double(yuan + (double)fen/100);

}

Currency operator\*(double d){

money = yuan\*d + fen\*d/100;

return Currency(money);

}

Currency(double d) {

int t = (d + 0.005) \* 100;

fen = t % 100;

yuan = t / 100;

}

protected:

int yuan;

int fen;

double money;

};

1. 设计表示复数的类Complex，重载复数的加减乘除运算，并编写一个主函数验证其正确性。

**答**：参考代码如下：

#include <iostream>

using namespace std;

class MyComplex

{

protected:

double real;

double imag;

public:

MyComplex(){}

MyComplex(double r, double i){

real = r;

imag = i;

}

MyComplex operator+(MyComplex& c){

return MyComplex(real+c.real, imag+c.imag);

}

MyComplex operator-(MyComplex& c){

return MyComplex(real-c.real, imag-c.imag);

}

MyComplex operator\*(MyComplex& c){

double r, i;

r = real\*c.real - imag\*c.imag;

i = imag\*c.real + real\*c.imag;

return MyComplex(r,i);

}

MyComplex operator/(MyComplex& c){

double s,r=0,i=0;

s=c.real\*c.real+c.imag\*c.imag;

if(s>0){

r=(real\*c.real+imag\*c.imag) / s;

i=(imag\*c.real-real\*c.imag) / s;

}

return MyComplex(r,i);

}

double getR(){

return real;

}

double getI(){

return imag;

}

};

void main()

{

MyComplex c1(3,2), c2(2,1), res;

res = c1+c2;

cout << res.getR() << '+' << res.getI() << 'i' << endl;

res = c1-c2;

cout << res.getR() << '+' << res.getI() << 'i' << endl;

res = c1\*c2;

cout << res.getR() << '+' << res.getI() << 'i' << endl;

res = c1/c2;

cout << res.getR() << '+' << res.getI() << 'i' << endl;

}

1. 第2章2.3节的示例中声明了表示日期的类Tdate，请定义并重载类Tdate的赋值、加法、减法及比较运算，并编写主函数验证其正确性。例如，日期d加整数n得到n天之后的日期，日期d减整数n得到n天之前的日期，日期d1减日期d2得到两个日期之间的天数。

**答**：参考代码如下：

#include <iostream>

using namespace std;

class Tdate {

public:

Tdate();

Tdate(int d, int m, int y);

Tdate(const Tdate &oldTdate);

bool getDate(int &d, int &m, int &y) const;

void print() const;

bool isLeapYear() const;

Tdate add(int id);

Tdate sub(int id);

int Compare(Tdate& t);

Tdate& operator=(Tdate t);

~Tdate();

private:

int day;

int month;

int year;

};

Tdate::Tdate() {

}

Tdate::Tdate(int d, int m, int y) {

day = d;

month = m;

year = y;

}

bool Tdate::getDate(int &d, int &m, int &y) const {

d = day;

m = month;

y = year;

return true;

}

bool Tdate::isLeapYear() const {

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

}

void Tdate::print() const {

cout << month << "/" << day << "/" << year << endl;

}

Tdate::~Tdate() {

}

Tdate::Tdate(const Tdate &oldTdate) {

memcpy(this, &oldTdate, sizeof(Tdate));

}

Tdate Tdate::add(int id) {

Tdate rt(\*this);

rt.day += id; //省略增加日期的逻辑，读者自己完善

return rt;

}

Tdate Tdate::sub(int id)

{

Tdate rt(\*this);

rt.day -= id; //省略减少日期的逻辑，读者自己完善

return rt;

}

int Tdate::Compare(Tdate& t)

{

return abs(day - t.day); //省略非同年同月计算逻辑，读者自己完善

}

Tdate& Tdate::operator=(Tdate t)

{

year = t.year;

month = t.month;

day = t.day;

return \*this;

}

void main()

{

Tdate t1(18,3,2023),t2(10,3,2023),t3;

t1.add(2).print();

t2.sub(2).print();

cout << t1.Compare(t2) << endl;

t3 = t1;

t3.print();

}

1. 第3章3.5节中声明了表示字符串的类myString，请定义并重载类myString的赋值、加法（连接）运算，并编写一个主函数验证其正确性。

**答**：参考代码如下：

#include <iostream>

using namespace std;

class MyString {

public:

MyString(const char s[]);

MyString(const MyString &oldMyString);

~MyString();

void print() const;

int getLen() const;

const char\* getString() const;

MyString& operator=(MyString s);

MyString& add(MyString& s);

private:

char \*ptrCharArray;

int len;

};

MyString::MyString(const char s[]) {

len = strlen(s);

ptrCharArray = new char[len + 1];

strncpy(ptrCharArray, s, len + 1);

ptrCharArray[len] = '\0';

}

MyString::~MyString() {

delete ptrCharArray;

}

void MyString::print() const {

cout << ptrCharArray << endl;

}

int MyString::getLen() const {

return len;

}

const char\* MyString::getString() const {

return ptrCharArray;

}

MyString::MyString(const MyString &oldMyString) {

len = oldMyString.getLen();

ptrCharArray = new char[len + 1];

strncpy(ptrCharArray, oldMyString.getString(), len + 1);

}

MyString& MyString::operator=(MyString s)

{

len = s.len;

ptrCharArray = new char[len + 1];

strncpy(ptrCharArray, s.ptrCharArray, len + 1);

ptrCharArray[len] = '\0';

return \*this;

}

MyString& MyString::add(MyString& s)

{

char\* p = new char[len+1];

strncpy(p,ptrCharArray,len+1);

delete ptrCharArray;

ptrCharArray = new char[len+s.len+1];

strncpy(ptrCharArray,p,len);

ptrCharArray[len] = '\0';

delete p;

strcat(ptrCharArray,s.ptrCharArray);

ptrCharArray[len+s.len] = '\0';

len += s.len;

return \*this;

}

void main()

{

MyString s1("张三"), s2(" ");

s2 = s1;

s2.print();

s1.add(s2);

s1.print();

}

1. 第4章4.3.2节中声明了类Number、、Real和Int三个类，请参照例4.8中的实现方法重载三个类的赋值和加减乘除运算，并编写一个主函数验证其正确性。

**答**：参考代码如下：

#include <iostream>

using namespace std;

class Number

{

public:

virtual unsigned add(unsigned n)=0;

virtual unsigned sub(unsigned n)=0;

virtual unsigned mul(unsigned n)=0;

virtual unsigned div(unsigned n)=0;

unsigned digits;

};

class Real:public Number

{

public:

Real(float v){

float \*p = (float\*) &digits;

\*p = v;

}

unsigned add(unsigned n){

float \*p1, \*p2, rt;

p1 = (float\*) &digits;

p2 = (float\*) &n;

rt = \*p1 + \*p2;

return \*((unsigned\*)&rt);

}

unsigned sub(unsigned n){

float \*p1, \*p2, rt;

p1 = (float\*) &digits;

p2 = (float\*) &n;

rt = \*p1 - \*p2;

return \*((unsigned\*)&rt);

}

unsigned mul(unsigned n){

float \*p1, \*p2, rt;

p1 = (float\*) &digits;

p2 = (float\*) &n;

rt = \*p1 \* \*p2;

return \*((unsigned\*)&rt);

}

unsigned div(unsigned n){

if(n == 0)

return 0;

else{

float \*p1, \*p2, rt;

p1 = (float\*) &digits;

p2 = (float\*) &n;

rt = \*p1 / \*p2;

return \*((unsigned\*)&rt);

}

}

Real& operator=(Real n){

digits = n.digits;

return \*this;

}

};

class Int:public Number

{

public:

Int(int v){

int \*p = (int\*) &digits;

\*p = v;

}

unsigned add(unsigned n){

return digits + n;

}

unsigned sub(unsigned n){

return digits - n;

}

unsigned mul(unsigned n){

return digits \* n;

}

unsigned div(unsigned n){

if(n == 0)

return 0;

else{

return digits / n;

}

}

Int& operator=(Int n){

digits = n.digits;

return \*this;

}

};

unsigned fAdd(Number& op1, Number& op2)

{

unsigned t = op1.add(op2.digits);

return t;

}

unsigned fSub(Number& op1, Number& op2)

{

unsigned t = op1.sub(op2.digits);

return t;

}

unsigned fMul(Number& op1, Number& op2)

{

unsigned t = op1.mul(op2.digits);

return t;

}

unsigned fDiv(Number& op1, Number& op2)

{

unsigned t = op1.div(op2.digits);

return t;

}

void main()

{

unsigned t;

//Int i1(10),i2(2);

Real i1(10),i2(2);

t=fAdd(i1,i2);

cout << t<<endl;

t=fSub(i1,i2);

cout << t<<endl;

t=fMul(i1,i2);

cout << t<<endl;

t=fDiv(i1,i2);

cout << t<<endl;

}

1. 6.6节中R进制计算机的功能不完善，请使用“按位比较”方式重载自然数的比较运算，并编写一个主函数验证其正确性。

**答**：参考代码如下：

#include <iostream>

#include "string.h"

using namespace std;

const int R = 10; //指定进制

const int bitcount = 20; //R进制的固定位数

typedef char Bit; //用一个char模拟计算机中的位

class MyNumber {

public:

//按位比较

int compare(MyNumber& a);

//自然数记法与内部存储之间的转换

void setNum(const char s[]);

char\* getNum(char s[]);

public:

Bit bit[bitcount];

};

void MyNumber::setNum(const char s[]) {

int i = strlen(s) - 1, j = 0;

while ((i >= 0) && (j < bitcount)) {

if ((s[i] >= 'A') && (s[i] <= 'Z'))

bit[j] = (s[i] - 'A' + 10) % R;

else if ((s[i] >= 'a') && (s[i] <= 'z'))

bit[j] = (s[i] - 'a' + 10) % R;

else if (((s[i] >= '0') && (s[i] <= '9')))

bit[j] = (s[i] - '0') % R;

else

bit[j] = 0;

i--;

j++;

}

while (j < bitcount) {

bit[j] = 0;

j++;

}

}

const char aph[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

char\* MyNumber::getNum(char s[]) {

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

s[i] = aph[bit[bitcount - i - 1]];

s[bitcount] = 0;

return s;

}

int MyNumber::compare(MyNumber& a)

{

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

{

if(bit[i] > a.bit[i])

return 1;

else if(bit[i] < a.bit[i])

return -1;

else

;

}

return 0;

}

void main()

{

MyNumber a, b;

int res;

char Buffer1[30] = "1234";

char Buffer2[30] = "567";

char Buffer[30];

a.setNum(Buffer1);

b.setNum(Buffer2);

res = a.compare(b);

cout << res << endl;

}

1. 例6.21中实数加法运算不完善，请查阅资料并完善其中的尾数处理、小数点对齐等功能。

**答**：略。

1. 请完善6.6节中的R进制计算机，请先重载自然数的算术运算、关系运算和位运算等基本运算，然后再重载整数的基本运算，最后重载实数的基本运算。

**答**：略。

# 模板与模板库

1. 参考7.1.3类模板的具体化和实例化中的方法，写出模板类Complex<double>的声明代码和实现代码，并上机调试通过。
2. 完善模板Complex的功能，重载复数的乘法（）、除法(算术运算，重载复数的相等运算。

**答：**



代码解释：

下面这段代码是一个C++程序，它定义了一个名为Complex的模板类，用来表示复数，并且重载了一些运算符来支持复数的基本运算。

下面逐步解释代码的主要部分：

1. **类模板声明**
2. 模板参数：template <class T> 表示这是一个模板类，可以使用不同的数据类型（如int, float, double等）。
3. 类定义：class Complex 定义了一个名为Complex的类，用于表示复数。
4. 成员变量
5. T a 和 T b 分别是复数的实部和虚部。
6. 构造函数和析构函数
7. 默认构造函数：Complex(){} 是一个默认构造函数，不做任何初始化操作。
8. 带参数的构造函数：Complex<T>(T x,T y) 用于创建具有指定实部和虚部的复数。
9. 拷贝构造函数：Complex<T>(const Complex<T>& complex) 用于复制已有的Complex对象。
10. 析构函数：~Complex<T>() 是一个析构函数，虽然在这里没有实际的功能。
11. **运算符重载**
12. 赋值运算符：operator= 允许你将一个Complex对象赋值给另一个。
13. 加法运算符：operator+ 用于计算两个复数的和。
14. 减法运算符：operator- 用于计算两个复数的差。
15. 乘法运算符：operator\* 用于计算两个复数的乘积。
16. 除法运算符：operator/ 用于计算两个复数的商。如果除数为0，抛出异常。
17. 相等运算符：operator== 用于比较两个复数是否相等，因为是浮点数，所以使用了一定的误差阈值来判断。
18. **显示方法**

display方法：用于以特定的精度打印复数。

1. **测试代码**

在main函数中，创建了两个Complex<double>类型的对象，并测试了加、减、乘、除以及相等性比较的功能。

代码整体上是一个很典型的C++类模板示例，演示了如何使用模板和运算符重载来实现一个可以处理复数运算的类。

#include <iostream>

#include <iomanip>

using namespace std;

//模板类的声明

template <class T>

class Complex {

public:

//构造函数

Complex(){};

Complex<T>(T x,T y){

a = x;

b = y;

}

~Complex<T>(){}

//拷贝构造

Complex<T>(const Complex<T>& complex){

a = complex.a;

b = complex.b;

}

//加减以及赋值符号重载

Complex<T>& operator=(const Complex<T>& complex){

a = complex.a;

b = complex.b;

return \*this;

}

Complex<T> operator+(const Complex<T>& complex){

Complex<T> rt(a,b);

rt.a += complex.a;

rt.b += complex.b;

return rt;

}

Complex<T> operator-(const Complex<T>& complex){

Complex<T> rt(a,b);

rt.a -= complex.a;

rt.b -= complex.b;

return rt;

}

//第二题部分

//重载乘除以及相等

Complex operator\*(const Complex& other) const{

return Complex<T>(a \* other.a - b \* other.b, a \* other.b + b \* other.b);

}

Complex operator/(const Complex& other) const{

T denominator = other.a \* other.a + other.b \* other.b;

if (denominator == 0) {

throw std::runtime\_error("Attempted to divide by zero");

}

return Complex<T>((a \* other.a + b \* other.b) / denominator,

(b \* other.a - a \* other.b) / denominator);

}

bool operator==(const Complex& other) const{

// 由于浮点数的精度问题，通常不能直接比较两个浮点数是否相等

// 这里使用一个小的阈值来判断两个复数是否“相等”

return std::abs(a - other.a) < 1e-9 && std::abs(b - other.b) < 1e-9;

}

void display()const{

cout<<setprecision(10)<<a<<"+"<<b<<"i";

}

private:

T a;//实部

T b;//虚部

};

//以下为测试代码

int main() {

// 测试构造函数

Complex<double> complex1(1.2, 2.4);

Complex<double> complex2(3.4, 4.1);

// 测试加法

Complex<double> sum = complex1 + complex2;

std::cout << "Sum: ";

sum.display();

std::cout << std::endl;

// 测试减法

Complex<double> difference = complex1 - complex2;

std::cout << "Diff: ";

difference.display();

std::cout << std::endl;

// 测试乘法

Complex<double> product = complex1 \* complex2;

std::cout << "Product: ";

product.display();

std::cout << std::endl;

// 测试除法

Complex<double> quotient = complex1 / complex2;

std::cout << "Quotient: ";

quotient.display();

std::cout << std::endl;

// 测试相等性

std::cout << "Equality: " << (complex1 == complex2) << std::endl;

return 0;

}

1. 参考7.2.2节中的方法，使用函数模板对数组中的复数按照降序进行排序。函数模板的原型如下：

template<class T> void bubble(T a[l, int size, bool(\*fp)(T& a, T& b)

template<class T> bool descending(T& a, T& b)

template<class T> void print(T& array, int len)

**答：**



**代码思路：**

下面这段代码实现了一个复数类 **Complex** 和用于对复数数组进行冒泡排序的函数模板。整体解决方案分为三个部分：

1. **复数类 Complex**
2. **构造函数**：允许创建复数实例，有默认构造函数、带参数的构造函数和拷贝构造函数。
3. **运算符重载**：实现了赋值、加法、减法、乘法、除法以及比较运算符。特别地，乘法和除法运算符被重载以支持复数的相应运算，而比较运算符 == 和 < 分别用于检查两个复数是否相等（考虑到浮点数的精度问题）和比较其大小（基于复数的平方模）。
4. **冒泡排序函数模板 bubble**
5. **泛型实现**：这是一个泛型函数，可以对任何类型 **T** 的数组进行排序，只要该类型定义了相应的比较运算。
6. **降序排序**：通过传入的比较函数 **fp**（在本例中是 **descending**），该函数可以实现降序排序。
7. **测试**
8. **复数实例**：创建了三个 **Complex<double>** 类型的实例。
9. **排序和打印**：使用 **bubble** 函数对这些复数进行降序排序，并在排序前后打印数组内容。
10. **特别注意**
11. **错误处理**：在除法运算中，如果除数的平方模为零，则抛出运行时错误，防止除以零的情况。
12. **友元函数**：重载了 **<<** 运算符以便于输出 **Complex** 对象，这需要将该运算符声明为类的友元。

//基于1、2题

#include <iostream>

#include <iomanip>

#include <complex>

using namespace std;

//模板类的声明

template <class T>

class Complex {

public:

//构造函数

Complex(){};

Complex<T>(T x,T y){

a = x;

b = y;

}

~Complex<T>(){}

//拷贝构造

Complex<T>(const Complex<T>& complex){

a = complex.a;

b = complex.b;

}

//加减以及赋值符号重载

Complex<T>& operator=(const Complex<T>& complex){

a = complex.a;

b = complex.b;

return \*this;

}

Complex<T> operator+(const Complex<T>& complex){

Complex<T> rt(a,b);

rt.a += complex.a;

rt.b += complex.b;

return rt;

}

Complex<T> operator-(const Complex<T>& complex){

Complex<T> rt(a,b);

rt.a -= complex.a;

rt.b -= complex.b;

return rt;

}

//第二题部分

//重载乘除以及相等

Complex operator\*(const Complex& other) const{

return Complex<T>(a \* other.a - b \* other.b, a \* other.b + b \* other.b);

}

Complex operator/(const Complex& other) const{

T denominator = other.a \* other.a + other.b \* other.b;

if (denominator == 0) {

throw std::runtime\_error("Attempted to divide by zero");

}

return Complex<T>((a \* other.a + b \* other.b) / denominator,

(b \* other.a - a \* other.b) / denominator);

}

bool operator==(const Complex& other) const{

// 由于浮点数的精度问题，通常不能直接比较两个浮点数是否相等

// 这里使用一个小的阈值来判断两个复数是否“相等”

return std::abs(a - other.a) < 1e-9 && std::abs(b - other.b) < 1e-9;

}

//以下两个函数为第三题其中一部分，基于1、2题基础重载

// 重载小于运算符

// 用复数平方模比较大小

T squaredMagnitude() const {

return a \* a + b \* b;

}

// 比较复数的平方模大小

bool operator<(const Complex& other) const {

return this->squaredMagnitude() < other.squaredMagnitude();

}

// 重载流插入运算符以便于输出复数

friend ostream& operator<<(ostream& out, const Complex& c) {

out << "(" << c.a << ", " << c.b << "i)";

return out;

}

void display()const{

cout<<setprecision(10)<<a<<"+"<<b<<"i";

}

private:

T a;//实部

T b;//虚部

};

//第3题

//冒泡排序

template<class T>

void bubble(T\* array, int size, bool (\*fp)(T&, T&)) {

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

for (int j = 0; j < size - i - 1; ++j) {

if (fp(array[j], array[j + 1])) {

swap(array[j], array[j + 1]);

}

}

}

}

//用于降序排列

template<class T>

bool descending(T& a, T& b) {

return a < b; // 降序排列

}

template<class T>void print(T\* array, int size) {

for (int i = 0; i < size; ++i) {

cout << array[i] << " ";

}

cout << endl;

}

//用于测试的main函数

int main() {

// Testing Complex class

Complex<double> c1(1.0, 2.0);

Complex<double> c2(-2.0, 3.0);

Complex<double> c3(4.0, -1.0);

cout << "c1: " << c1 << endl;

cout << "c2: " << c2 << endl;

cout << "c3: " << c3 << endl;

// Testing bubble sort with Complex numbers

Complex<double> complexArray[] = {c1, c2, c3};

int arraySize = sizeof(complexArray) / sizeof(complexArray[0]);

cout << "Original array: ";

print(complexArray, arraySize);

bubble(complexArray, arraySize, descending);

cout << "Sorted array in descending order: ";

print(complexArray, arraySize);

return 0;

}

1. 使用类模板myArray 存储和管理如图3.23所示的汽车类Car、车轮类Wheel和发动机类Motor的对象，并根据各自的职责完善每个类的构造函数、析构函数和拷贝构造函数，以及重载需要的运算。

**答：**





**代码解释：**

下面这段代码主要展示了如何使用模板类 **myArray** 来管理汽车（**Car**）、车轮（**Wheel**）和发动机（**Motor**）对象，并展示了这些类的构造函数、析构函数和拷贝构造函数的实现。下面是各个部分的解释：

1. **myArray 模板类**
2. 构造函数：接受一个 size 参数，创建一个大小为 size 的数组。
3. 析构函数：释放数组内存。
4. 拷贝构造函数：创建一个新数组，并复制原数组的内容。
5. 重载操作符 []：返回数组中指定索引的元素。
6. **Wheel 类（继承自 WheelBase）**
7. 成员变量：serialNumber（序列号）和 size（大小）。
8. 构造函数：包括无参构造函数、带参数的构造函数和拷贝构造函数。
9. 打印函数：打印车轮信息。
10. 赋值运算符重载：用于拷贝构造函数中。
11. **Motor 类（继承自 MotorBase）**
12. 成员变量：serialNumber（序列号）和 power（动力）。
13. 构造函数：包括无参构造函数、带参数的构造函数和拷贝构造函数。
14. 打印函数：打印发动机信息。
15. **Car 类**
16. 成员变量：wheel（车轮数组）和 motor（指向发动机的指针）。
17. 构造函数：包括无参构造函数、带参数的构造函数和拷贝构造函数。
18. 析构函数：不需要手动删除成员，因为 myArray 和 Motor 类已经处理。
19. set 方法：用于设置发动机和车轮。
20. 打印函数：打印整车信息。

#include <iostream>

using namespace std;

template<typename T>class myArray {

T\* array;

int size;

public:

myArray(int size) : size(size) {

array = new T[size];

}

~myArray() {

delete[] array;

}

myArray(const myArray& other) : size(other.size) {

array = new T[size];

for (int i = 0; i < size; ++i) {

array[i] = other.array[i];

}

}

T& operator[](int index) {

return array[index];

}

};

// WheelBase类

class WheelBase {};

// MotorBase类

class MotorBase {};

// Motor类，继承自MotorBase

class Motor {

private:

int serialNumber;

float power;

public:

Motor(){}

Motor(int sn, float pPower) : serialNumber(sn), power(pPower) {}

~Motor() {}

Motor(const Motor& other) : serialNumber(other.serialNumber), power(other.power) {}

void print() {

cout << serialNumber << "," << power;

}

};

// Wheel类，继承自WheelBase

class Wheel : public WheelBase {

private:

int serialNumber;

float size;

public:

Wheel() {};

Wheel(int sn, float fSize) : serialNumber(sn), size(fSize) {}

~Wheel() {}

Wheel(const Wheel& other) : WheelBase(other), serialNumber(other.serialNumber), size(other.size) {}

void print() {

cout << serialNumber << "," << size ;

}

Wheel& operator = (Wheel& t) {//拷贝构造函数中使用赋值运算

serialNumber = t.serialNumber;

size = t.size;

return\*this;

}

};

// Car类

class Car {

private:

myArray<Wheel> wheel;

Motor \*motor;

public:

Car() :wheel(4){}

Car( Motor& mMotor, myArray<Wheel>& aWheels)

: motor(&mMotor), wheel(aWheels) {}

~Car() {

// 析构函数不再需要手动删除，因为 myArray 会自动处理

}

Car(const Car& other) : motor(other.motor), wheel(other.wheel) {}

Car& operator=(const Car& other) {

if (this != &other) {

motor = other.motor;

wheel = other.wheel;

}

return \*this;

}

Motor& set(Motor& depart) {

Motor& rt = \*motor;

motor = &depart;

return rt;

}

Wheel& set(Wheel& depart, int position) {

Wheel& rt = wheel[position];

wheel[position] = depart;

return rt;

}

void print() {

cout << "汽车\t发动机：";

motor->print();

cout << "\t\t车轮：";

for (int i = 0; i < 4; i++) {

wheel[i].print();

cout << "|";

}

cout << endl;

}

};

//测试用main函数

int main() {

Car car;

Motor\* m = new Motor(123, 1.6);

car.set(\*m);

for (int i = 0; i < 4; i++) {

Wheel w(201 + i, 10);

car.set(w, i);

}

car.print();

delete m;

}

1. 在网上查阅XML 标准文本或 Joson 资料，并从中选择一种，按照其中规定的数据格式及含义，重载课程成绩管理中学生 Student 和教学班TeachingClass 的提取和插入运算。

**答：**





**代码解释：**

这两个代码片段都包含一个名为 Student 的类，但分别用于处理 XML 和 JSON 数据。我将分别解释这两个文件的代码逻辑。

1. **XML 处理代码**
2. 引入库: 使用 <iostream>、<string> 和 "tinyxml2.h"，分别用于标准输入输出、字符串操作和处理 XML 数据。
3. 命名空间: 使用标准命名空间 std 和 tinyxml2。
4. Student 类定义:
5. 构造函数: 提供多个构造函数以支持不同的初始化方式。
6. 拷贝构造函数和赋值运算符: 用于对象的复制和赋值。
7. fromXML 方法: 将 XML 字符串解析成 Student 对象。
8. toXML 方法: 将 Student 对象的数据转换为 XML 字符串。
9. 重载输出运算符: 使 Student 对象可以通过 cout 直接输出。
10. fromXML 实现:
11. 解析 XML 字符串。
12. 从 XML 中提取 Student 的 ID 和名称。
13. toXML 实现:
14. 创建 XML 元素并添加 Student 的 ID 和名称。
15. 返回生成的 XML 字符串。
16. 主函数 (main): 创建一个 Student 对象，将其转换为 XML，然后再从 XML 重新创建一个 Student 对象并输出。
17. **JSON 处理代码**
18. 引入库: 使用 <iostream>、<string> 和 "json.hpp"，后者是用于处理 JSON 的库。
19. 命名空间: 使用标准命名空间 std 和 json，后者是 nlohmann::json 库的简称。
20. Student 类定义: 与 XML 版本相似，但 fromXML 和 toXML 方法分别被 fromJSON 和 toJSON 替换。
21. fromJSON 实现:
22. 解析 JSON 字符串。
23. 从 JSON 中提取 Student 的 ID 和名称。
24. toJSON 实现:
25. 创建 JSON 对象并添加 Student 的 ID 和名称。
26. 返回生成的 JSON 字符串。
27. 主函数 (main): 创建一个 Student 对象，将其转换为 JSON，然后再从 JSON 重新创建一个 Student 对象并输出。

在使用这些代码之前，需要确保相应的库（TinyXML2 和 JSON for Modern C++）已经正确安装并配置在您的开发环境中。如果遇到编译或链接问题，请检查库的安装和配置。

#include <iostream>

#include <string>

#include "tinyxml2.h"

using namespace std;

using namespace tinyxml2;

class Student {

public:

Student() : studentID(0) {}

Student(int id, string nm) : studentID(id), name(nm) {}

Student(const Student& other) : studentID(other.studentID), name(other.name) {}

Student& operator=(const Student& other) {

studentID = other.studentID;

name = other.name;

return \*this;

}

void fromXML(const string& xmlData);

string toXML() const;

friend ostream& operator<<(ostream& out, const Student& student) {

return out << "ID: " << student.studentID << ", Name: " << student.name;

}

private:

int studentID;

string name;

};

void Student::fromXML(const string& xmlData) {

XMLDocument doc;

doc.Parse(xmlData.c\_str());

XMLElement\* root = doc.FirstChildElement("Student");

if (root) {

studentID = root->IntAttribute("ID");

name = root->Attribute("Name");

}

}

string Student::toXML() const {

XMLPrinter printer;

printer.OpenElement("Student");

printer.PushAttribute("ID", studentID);

printer.PushAttribute("Name", name.c\_str());

printer.CloseElement();

return printer.CStr();

}

// 测试代码

int main() {

Student student(1, "Alice");

string xml = student.toXML();

cout << "Student to XML: " << xml << endl;

Student newStudent;

newStudent.fromXML(xml);

cout << "XML to Student: " << newStudent << endl;

return 0;

}

//处理 JSON 数据的文件（StudentJSON.cpp）

#include <iostream>

#include <string>

#include "json.hpp"

using namespace std;

using json = nlohmann::json;

class Student {

public:

Student() : studentID(0) {}

Student(int id, string nm) : studentID(id), name(nm) {}

Student(const Student& other) : studentID(other.studentID), name(other.name) {}

Student& operator=(const Student& other) {

studentID = other.studentID;

name = other.name;

return \*this;

}

void fromJSON(const string& jsonData);

string toJSON() const;

friend ostream& operator<<(ostream& out, const Student& student) {

return out << "ID: " << student.studentID << ", Name: " << student.name;

}

private:

int studentID;

string name;

};

void Student::fromJSON(const string& jsonData) {

auto j = json::parse(jsonData);

studentID = j["ID"].get<int>();

name = j["Name"].get<string>();

}

string Student::toJSON() const {

json j;

j["ID"] = studentID;

j["Name"] = name;

return j.dump();

}

// 测试代码

int main() {

Student student(2, "Bob");

string jsonStr = student.toJSON();

cout << "Student to JSON: " << jsonStr << endl;

Student newStudent;

newStudent.fromJSON(jsonStr);

cout << "JSON to Student: " << newStudent << endl;

return 0;

}

在这两个文件中，Student 类包含了从 XML/JSON 提取数据 (fromXML/fromJSON) 和将数据转换为 XML/JSON (toXML/toJSON) 的方法。主函数 (main) 包含了对这些方法的测试代码。确保您的开发环境中已正确配置了对应的库。如果遇到编译或链接错误，请检查库的安装和配置。

1. 使用向量 vector 存储矩阵，并实现矩阵的加法、减法、乘法和转置运算。

**答：**



**代码解释：**

这段代码实现了一个简单的矩阵操作类 **Matrix**，以及一个 **main** 函数来演示如何使用这个类来进行矩阵的基本操作，包括加法、减法、乘法和转置。下面是代码的逐步解释：

1. **Matrix 类**
2. **私有成员变量**:
3. **data**：存储矩阵元素的二维向量。
4. **rows**、**cols**：分别表示矩阵的行数和列数。
5. **构造函数**:
6. 初始化矩阵的行数、列数，并将所有元素初始化为0。
7. **设置和获取元素**:
8. set 方法用于设置矩阵中特定位置的元素值。
9. get 方法用于获取矩阵中特定位置的元素值。
10. **运算符重载**:
11. operator+：实现矩阵加法。
12. operator-：实现矩阵减法。
13. operator\*：实现矩阵乘法，注意这里采用了标准的矩阵乘法规则。
14. transpose：实现矩阵的转置操作。
15. **打印矩阵**:
16. **print** 方法用于打印矩阵的所有元素。
17. **main 函数**
18. 创建两个 2x2 的矩阵 **m1** 和 **m2**，并设置它们的元素值。
19. 打印 **m1** 和 **m2**。
20. 计算并打印 **m1** 和 **m2** 的和（**sum**）、差（**diff**）、积（**prod**）。
21. 计算并打印 **m1** 的转置矩阵（**trans**）。

//在这个程序中，main 函数首先创建两个 2x2 矩阵 m1 和 m2。

//然后分别计算并打印它们的和、差、积和 m1 的转置矩阵。

#include <vector>

#include <iostream>

class Matrix {

private:

std::vector<std::vector<int>> data;

int rows, cols;

public:

// 构造函数

Matrix(int r, int c) : rows(r), cols(c), data(r, std::vector<int>(c, 0)) {}

// 设置元素值

void set(int r, int c, int value) {

data[r][c] = value;

}

// 获取元素值

int get(int r, int c) const {

return data[r][c];

}

// 矩阵加法

Matrix operator+(const Matrix& other) const {

Matrix result(rows, cols);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

result.set(i, j, this->get(i, j) + other.get(i, j));

}

}

return result;

}

// 矩阵减法

Matrix operator-(const Matrix& other) const {

Matrix result(rows, cols);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

result.set(i, j, this->get(i, j) - other.get(i, j));

}

}

return result;

}

// 矩阵乘法

Matrix operator\*(const Matrix& other) const {

Matrix result(rows, other.cols);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < other.cols; j++) {

int sum = 0;

for (int k = 0; k < cols; k++) {

sum += this->get(i, k) \* other.get(k, j);

}

result.set(i, j, sum);

}

}

return result;

}

// 矩阵转置

Matrix transpose() const {

Matrix result(cols, rows);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

result.set(j, i, this->get(i, j));

}

}

return result;

}

// 打印矩阵

void print() const {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

std::cout << data[i][j] << " ";

}

std::cout << std::endl;

}

}

};

int main() {

Matrix m1(2, 2);

m1.set(0, 0, 1);

m1.set(0, 1, 2);

m1.set(1, 0, 3);

m1.set(1, 1, 4);

Matrix m2(2, 2);

m2.set(0, 0, 5);

m2.set(0, 1, 6);

m2.set(1, 0, 7);

m2.set(1, 1, 8);

std::cout << "Matrix m1:" << std::endl;

m1.print();

std::cout << "Matrix m2:" << std::endl;

m2.print();

Matrix sum = m1 + m2;

std::cout << "Sum of m1 and m2:" << std::endl;

sum.print();

Matrix diff = m1 - m2;

std::cout << "Difference of m1 and m2:" << std::endl;

diff.print();

Matrix prod = m1 \* m2;

std::cout << "Product of m1 and m2:" << std::endl;

prod.print();

Matrix trans = m1.transpose();

std::cout << "Transpose of m1:" << std::endl;

trans.print();

return 0;

}

1. 先在MSDN 中查找链表 list 的资料，然后按照5.1节中的步和方法，选用list存储和管理小孩，重新编程实现。

答：



**代码解释：**

这段代码实现了一个基于环形链表的“约瑟夫问题”（Josephus problem）的模拟。下面是代码的逻辑解释：

1. **包含头文件**：
2. **#include <iostream>**：包含标准输入输出流的头文件。
3. **#include <list>**：包含标准模板库中的链表容器类。
4. **类定义 - Boy**：
5. 这个类代表游戏中的小孩，每个小孩有一个唯一的标识符 **id**。
6. Boy(int id)：构造函数，用于初始化小孩的编号。
7. print() 方法：打印小孩的编号。
8. getId() 方法：返回小孩的编号。
9. static void leave() 方法：静态成员函数，用于从链表中移除当前迭代器指向的小孩，并更新迭代器指向下一个小孩。
10. **主函数 - main()**：
11. 输入小孩的数量 n 和步数 m。
12. 创建一个 std::list<Boy> 类型的链表 boys，并填充编号为 1 到 n 的小孩。
13. 使用迭代器 it 指向链表的开始。
14. 当链表中的元素个数大于1时，循环执行以下步骤：
    * + 循环 **m-1** 次，每次迭代器向前移动一步。如果迭代器到达链表末尾，则重新指向链表的开始。
      + 调用 **Boy::leave(it, boys)** 方法移除迭代器当前指向的小孩，并更新迭代器指向下一个小孩。如果迭代器到达链表末尾，则重新指向链表的开始。
15. 当只剩下一个小孩时，打印这个小孩的编号。

这个程序通过循环链表模拟了“约瑟夫问题”，在每个循环中按照给定的步数 **m** 移除一个小孩，直到只剩下一个小孩为止。

#include <iostream>

#include <list>

class Boy {

public:

Boy(int id) : id(id) {} // 构造函数

void print() const {

std::cout << "Id: " << id << std::endl;

}

int getId() const {

return id;

}

static void leave(std::list<Boy>::iterator& it, std::list<Boy>& boys) {

std::cout << "Leaving: " << it->getId() << std::endl;

it = boys.erase(it); // 删除当前元素

}

private:

int id; // 小孩的编号

};

int main() {

int n, m;

std::cout << "Enter the number of boys: ";

std::cin >> n;

std::cout << "Enter the step count: ";

std::cin >> m;

std::list<Boy> boys;

for (int i = 1; i <= n; ++i) {

boys.push\_back(Boy(i));

}

auto it = boys.begin();

while (boys.size() > 1) {

for (int step = 1; step < m; ++step) {

++it;

if (it == boys.end()) {

it = boys.begin();

}

}

Boy::leave(it, boys);

if (it == boys.end()) {

it = boys.begin();

}

}

std::cout << "Last boy standing: ";

boys.front().print();

return 0;

}

# 课程成绩管理应用案例

1. 结合就读学校的具体情况，完善课程成绩管理系统，并参照本章介绍的步骤和方法编程实现。

**答：**

1. **对场景进行分析**

课程成绩管理系统主要涉及教师、学生和教学管理教师。教师期望录入成绩，学生期望查询成绩，教学管理教师期望对成绩进行统计分析，以评价教师的教学效果和学生的学习效果，教师、学生和教学管理教师在处理课程成绩过程中，还要从教务系统中获取教师、学生和课程，以及课程安排等基础数据。画出相应的用例图。分析课程的组织实施场景，概念，排除不相关的概念，合并相同的概念，删除重复术语，找到课程成绩管理中涉及的主要概念及其相互关系。

1. **教师录入成绩的视图**

教师使用课程成绩管理系统的目的是录入成绩，主要关心的是课程成绩、学生及教学班。从教师的角度调整课程成绩管理的关系视图。并生成教学班TeachingClass类，课程Course类，学生Student类，被学生类继承的StudentBase类，课程分数GradeOfCourse类，教师Teacher类，基于概念的内涵完成类中的属性，并设置“编号”属性。加入ManagerOfGrade类用来管理教学班。

1. **代码实现成员函数**

//StudentBase设计学生基础类

int StudentBase::getStudentID()

void StudentBase::setStudentID(int newStudentID)

bool StudentBase::operator == (const StudentBase& b) const

bool StudentBase::operator > (const StudentBase& b) const

bool StudentBase::operator < (const StudentBase& b) const

StudentBase::operator int()

//GradeOfCourse设计课程成绩类

float GradeOfCourse::getGrade(void)

void GradeOfCourse::setGrade(float newGrade)

Student GradeOfCourse::getStudent(vector<Student>& StudentTbl)

int GradeOfCourse::setStudent(int newStudentID, vector<Student>& StudentTbl)

int GradeOfCourse::setStudent(Student& newStudent, vector<Student>& StudentTbl)

//设计教学班TeachingClass类

TeachingClass::TeachingClass(vector<Student>& st) : gradeTbl(0), course()

int TeachingClass::modifyGrade(int studentID, float grade)

list<GradeOfCourse>& TeachingClass::getGradeTbl(void)

GradeOfCourse TeachingClass::findGradeByID(int studentID)

//设计管理教学班类MangerOfGrade

int MangerOfGrade::readTeacher(char\*)

int MangerOfGrade::readCourse (char\*)

int MangerOfGrade::readTeachingClass(char\*)

TeachingClass MangerOfGrade::createTeachingClass(char\*, char\*, char\*)

int MangerOfGrade::addTeachingClass(TeachingClass)

vector<GradeOfStudent> ManagerOfGrade::findByStudent(int id)

int loadInitData(MangerOfGrade)

int editGrade(MangerOfGrade)

1. **代码实现main函数**

void main(){

ManagerOfGrade mg;

loadInitData(mg);

editGrade(mg);

cout << endl << "学生：202106 成绩表" << endl;

vector<GradeOfStudent> gsTbl = mg.findByStudent(202106);

for (vector<GradeOfStudent>::iterator it = gsTbl.begin(); it != gsTbl.end(); it++)

cout << it->courseNo << "\t" << it->title << "\t" << it->grade << endl;

}

1. 查阅就读学校选课的管理文件，并根据规定的选课流程，设计一个选课程序，并编程实现。具体要求可参考第三章练习8。

答： 参考代码如下：

#include <iostream>

#include <fstream>

#include <vector>

#include<string.h>

using namespace std;

struct kc

{

char bianhao[10];

char name[20];

int xueshi;

int xuefen;

char kaixueinf[10];

};

class course

{

public:

int cnt; //已添加课程数量

vector<struct kc> C;

public:

course();

void printfc();

void reinf();

void deinf();

void getinf();

bool filesave(char\*temp); //保存文本文件

bool fileread(char\*temp); //读取文本文件

};

course::course()

{

cnt = 0;

}

void course::printfc()

{

printf("课程编号，课程名称，课程学时，课程学分，开学时间\n");

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

{

printf("%s,%s,%d,%d,%s\n", C.at(i).bianhao, C.at(i).name, C.at(i).xueshi, C.at(i).xuefen, C.at(i).kaixueinf);

}

}

void course::reinf()

{

char temp[20];

cout << "请输入课程的编号或者名称" << endl;

scanf("%s", temp);

int i = 0;

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

{

if (strcmp(C.at(i).bianhao, temp) == 0 || strcmp(C.at(i).name, temp) == 0)break;

}

printf("查询课程信息为%s,%s,%d,%d,%s\n", C.at(i).bianhao, C.at(i).name, C.at(i).xueshi, C.at(i).xuefen, C.at(i).kaixueinf);

cout << "请输入修改后的 课程编号，课程名称，课程学时，课程学分，开学时间（xxxx,x,x）" << endl;

scanf("%s", C.at(i).bianhao);

scanf("%s", C.at(i).name);

scanf("%d", &C.at(i).xueshi);

scanf("%d", &C.at(i).xuefen);

scanf("%s", C.at(i).kaixueinf);

cout << "修改成功" << endl;

}

void course::deinf()

{

char temp[20];

cout << "请输入课程的编号或者名称" << endl;

scanf("%s", temp);

int i = 0;

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

{

if (strcmp(C.at(i).bianhao, temp) == 0 || strcmp(C.at(i).name, temp) == 0)break;

}

if (i == cnt)

{

cout << "无此课程" << endl;

return;

}

printf("删除课程信息为%s,%s,%d,%d,%s\n是否确认删除?是输入1，不是输入0", C.at(i).bianhao, C.at(i).name, C.at(i).xueshi, C.at(i).xuefen, C.at(i).kaixueinf);

int Isde=0;

cin >> Isde;

if (Isde)

{

C.erase(C.begin() + i); //参数为迭代器地址

cnt--;

cout << "删除成功" << endl;

}

}

void course::getinf()

{

int N = 0;

cout << "请输入要录入的课程数量" << endl;

cin >> N;

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

{

struct kc a;

cout << "请输入 课程编号，课程名称，课程学时，课程学分，开学时间（xxxx.x.x）" << endl;

scanf("%s", a.bianhao);

scanf("%s", a.name);

scanf("%d", &a.xueshi);

scanf("%d", &a.xuefen);

scanf("%s", a.kaixueinf);

//scanf("%s,%s,%d,%s,%d", a.bianhao, a.name, &a.xueshi, a.kaixueinf, &a.xuefen);

C.push\_back(a);

cnt++;

}

cout << "录入完成" << endl;

}

bool course::filesave(char\*temp) //保存文本文件

{

ofstream out;

out.open(temp, ios::out);

if (out.is\_open()==0)return false;

out << "课程信息" << endl;

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

{

out << C.at(i).bianhao << endl;

out << C.at(i).name << endl;

out << C.at(i).kaixueinf << endl;

out << C.at(i).xuefen << endl;

out << C.at(i).xueshi << endl;

}

out << "录入完成" << endl;

out.close();

return true;

}

bool course::fileread(char\*temp) //读取文本文件

{

char data[20];

ifstream in;

in.open(temp, ios::in);

if (in.is\_open() == 0)return false;

in >> data;

if (strcmp("课程信息", data) != 0)

{

cout << "不是课程数据文件，打开失败" << endl;

return false;

}

while (1)

{

struct kc a;

in >> a.bianhao;

if (strcmp(a.bianhao, "录入完成") == 0)break;

in >> a.name ;

in >> a.kaixueinf;

in >> a.xuefen ;

in >> a.xueshi ;

C.push\_back(a);

cnt++;

}

in.close();

return true;

}

//设计student类//

class student

{

public:

char yixuankec[256];

char yanz[10];

char xuehao[20];

int kecnt ; //已选课程数目

public:

student(char \*\_xuehao=NULL,char \*\_yanz=NULL);

void choiceC(char\* \_bianhao); //选课功能

void checkC(); //查询功能

void getkc(int cnt, char\* temp); //返回第cnt个课程的编号

void setyanz(char \*\_yanz); //重新设置验证码

void setxuehao(char \*\_xuehao); //设置学号

bool xuehaopipei(char \*\_xuehao); //学号是否匹配

bool yanzpipei(char\*\_yanz); //验证码是否匹配

};

student::student(char \*\_xuehao, char \*\_yanz)

{

kecnt = 0;

yixuankec[0] = 0;

if (\_xuehao != NULL)strcpy(xuehao, \_xuehao);

if (\_yanz!=NULL)strcpy(yanz, \_yanz);

}

void student::choiceC(char\* \_bianhao) //选课功能

{

int i = 0;

for (i = 0; i < 256; i++)

{

if (yixuankec[i] == 0)

{

if (kecnt != 0)

{

yixuankec[i] = ',';

i++;

}

break;

}

}

strcpy(yixuankec + i, \_bianhao);//此处没有判断i是否越界

kecnt++;

}

void student::checkC() //查询功能

{

printf("学号：%s\n", xuehao);

printf("登陆验证：%s\n", yanz);

printf("已选课程编号：%s\n", yixuankec);

}

void student::getkc(int cnt, char\* temp)

{

char temp1[10];

int c=0;

for (int i = 0,j=0; i < 256; i++)

{

if (yixuankec[i] != 0 && yixuankec[i] != ',')

{

temp1[j] = yixuankec[i];

j++;

}

else //yixuankec[i] == ',' \0

{

c++;

temp1[j] = 0;

if (c == cnt)

{

strcpy(temp, temp1);

return;

}

j = 0;

}

}

}

void student::setyanz(char \*\_yanz) //重新设置验证码

{

strcpy(yanz, \_yanz);

//cout << "设置成功" << endl;

}

void student::setxuehao(char \*\_xuehao) //设置学号

{

strcpy(xuehao, \_xuehao);

//cout << "申请成功" << endl;

}

bool student::xuehaopipei(char \*\_xuehao) //学号是否匹配

{

if (strcmp(xuehao, \_xuehao) == 0)return true;

else return false;

}

bool student::yanzpipei(char\*\_yanz) //验证码是否匹配

{

if (strcmp(yanz, \_yanz) == 0)return true;

else return false;

}

//设计主函数main//

int main()

{

course A; //10门课

student B[10]; //10个学生

int stucnt=0;

int dengluflag=-1;

while (1)

{

int SW;

cout << "欢迎来到选课系统,请选择编号进行操作" << endl;

cout << "1.注册账号" << endl;

cout << "2.登陆已有账号" << endl;

cin >> SW;

switch (SW)

{

case 1:

char temp[20];

cout << "请输入注册学号账号" << endl;

scanf("%s", temp);

B[stucnt].setxuehao(temp);

cout << "请输入登陆验证密码" << endl;

scanf("%s", temp);

B[stucnt].setyanz(temp);

cout << "注册成功,注册信息如下" << endl;

B[stucnt].checkC();

stucnt++;

system("pause");

break;

case 2:

char temp1[10];

cout << "请输入学号账号" << endl;

scanf("%s", temp);

go1:cout << "请输入登陆验证密码" << endl;

scanf("%s", temp1);

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

{

if (B[i].xuehaopipei(temp) == true)

{

if (B[i].yanzpipei(temp1) == true)

{

cout << "登陆成功" << endl;

dengluflag = i;

}

else

{

cout << "验证密码错误，重试" << endl;

goto go1;

}

}

}

if (dengluflag < 0)cout << "学号不存在" << endl;

system("pause");

break;

}

if(dengluflag>=0)

while (1)

{

system("cls");

int SW1=0;

cout << "欢迎登陆，请选择功能编号" << endl;

cout << "1.课程信息录入" << endl;

cout << "2.课程信息修改" << endl;

cout << "3.课程信息删除" << endl;

cout << "4.课程信息浏览" << endl;

cout << "5.开始选课" << endl;

cout << "6.选课结果查询" << endl;

cout << "7.课程信息保存" << endl;

cout << "8.退出登陆" << endl;

cin >> SW1;

switch (SW1)

{

char temp[20];

case 1:

{

int a1 = 0;

system("cls");

cout << "1.手动录入" << endl;

cout << "2.从文件读取课程信息" << endl;

cin >> a1;

if (a1 == 1)A.getinf();

else

{

cout << "请输入录入文件地址:";

char adress1[256];

cin >> adress1;

if (A.fileread(adress1) == true)cout << "录入成功" << endl;

}

system("pause");

break;

}

case 2:

A.reinf();

system("pause");

break;

case 3:

A.deinf();

system("pause");

break;

case 4:

A.printfc();

system("pause");

break;

case 5:

A.printfc();

cout << "请输入要选的课程编号" << endl;

scanf("%s", temp);

B[dengluflag].choiceC(temp);

cout << "选课成功" << endl;

system("pause");

break;

case 6:

{

system("cls");

int sw = 0;

int i=0;

cout << "1.按学号查询学生选课情况" << endl;

cout << "2.按课程查询学生选课情况" << endl;

cin >> sw;

switch (sw)

{

case 1:

cout << "请输入学号" << endl;

scanf("%s", temp);

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

{

if (B[i].xuehaopipei(temp))B[i].checkC();

}

system("pause");

break;

case 2:

char temp62[20];

cout << "请输入查询课程编号" << endl;

scanf("%s", temp62);

cout << "选择此门课程的学号有" << endl;

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

{

for (int j = 1; j <= B[i].kecnt; j++)

{

char temp1[20];

B[i].getkc(j, temp1);

if (strcmp(temp1, temp62) == 0)cout << B[i].xuehao << ",";

}

}

cout << endl;

break;

}

system("pause");

break;

}

case 7:

{

char adress7[256];

cout << "请输入文件的保存地址：";

cin >> adress7;

if (A.filesave(adress7) == true)cout << "保存成功";

else cout << "保存失败" << endl;

break;

}

case 8:

dengluflag = -1;

break;

}

if (dengluflag == -1)

{

cout << "正在退出登录...." << endl;

break;

}

\_sleep(500);

system("cls");

}

\_sleep(1000);

system("cls");

}

system("pause");

return 0;

}