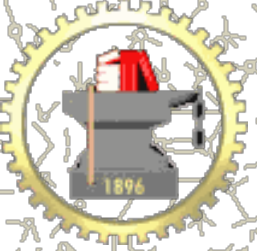
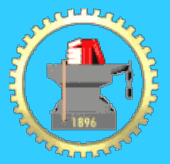


计算机程序设计（C++）

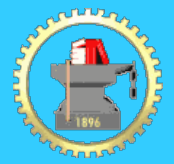
数据的抽象与封装

内容提要



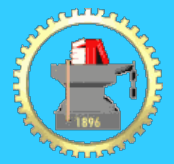
- **多种数据特征和函数的封装——类**
  - **类**
  - **对象**
  - **构造函数与析构函数**
  - **类的组合**
  - **多文件结构**

多种数据特征和函数的封装——类



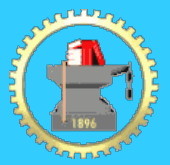
- 在结构体类型中可以定义若干个成员表示一个整体，比如通话记录这个结构体可以包含对方电话号码、持续时间、发生日期、发生时间等。在使用结构体类型时需要定义结构体变量，然后对它所包含的各个成员进行赋值。在默认情况下，这些成员在结构体之外是公开的（称为具有public权限），外部可以任意使用。其实，C++语言对结构体进行了一定程度的改良，比如在结构体类型中不但可以增加类似成员的内部变量，还可以增加内部函数，这样就扩展了结构体的用途范围，而这些内部函数对外也是公开的。以下将要讲到的类与结构体有相似之处，它不但包含结构体所体现的功能，而且设计上更灵活，更易于管理，数据也更安全。

6.2.1类

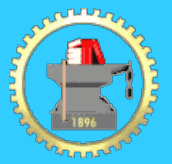


- 类与自然界分门别类是同样一个概念，例如人是一个大类，学生和教师是其子类；数学是一个大类，高等数学和初等数学是其子类。在C++中，类也是一种数据类型。对象是每个类中的一个个实体和事项，例如学生是一个子类，而王东同学作为学生类的一个实体，他不仅属于该类，他还能展示该类所具有的基本特征、性能、动作和功能等，如学号、班级、所学课程、成绩等。在C++中，类的所有特征、性能和功能等均要通过对象才能展示和实现。

类定义



```
class 类名
{
        成员访问权限:
                成员变量定义;
        成员访问权限:
                成员函数定义;
        ......
};
```



- **成员变量**

  数据类型 成员变量名;

- **成员函数**

  – 如果在类内定义，其形式为：

  返回类型 成员函数 (数据类型 形式参数1 , 数据类型 形式参数2 , ...)

  {

  函数体

  }

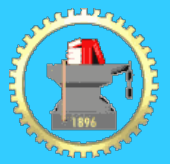
  – 如果在类外定义，其形式为：

  返回类型 类名::成员函数 (数据类型 形式参数1 ,.....)

  {

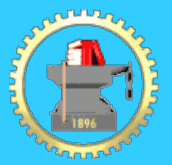
  函数体

  }



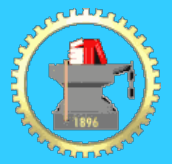
- 内联函数
  - 隐式定义
  - 显式定义

```
inline 返回类型 类名::成员函数 ( 数据类型 形式参数1 ,..... )
{
    函数体
}
```



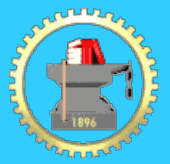
```
class mobile
{
private:
char mynumber[11];//机主的电话号码
  public:
      void init(char number[11]=”0000000000”)//隐式定义的内联函数
      {
            mynumber=number;
      }
      void dial();       //拨打电话
      void answer(char othernumber[11]);   //接听电话
      void hangup();     //挂断电话
}
```

6.2.2对象



- 类实际上是一种抽象机制，描述了一类问题的共同属性和行为，在C++中，类的对象就是该类的某一特定实体（也称实例）。如果将类看作是一种类型，那么类的对象就可以看作为该类型的变量。

对象定义



- **单独定义**

  类名 对象变量名 ;

  mobile m1,m2; //定义类对象m1和m2

- **混合定义**

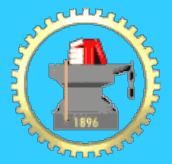
  class mobile

  {

  ...

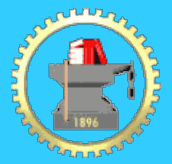
  } mobile m1,m2; //定义类对象m1和m2

对象初始化



```
class mobile
{
public:
char mynumber[11];//机主的电话号码
  .......
} m1=”1111111111”;
```

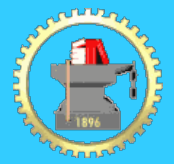
对象使用



- 对象成员的引用
  - 对象变量名.成员变量或成员函数
  - 对象指针名->成员变量或成员函数
- 对象的整体赋值

  对象1=对象2
- 对象作为函数的形式参数

【例6-6】

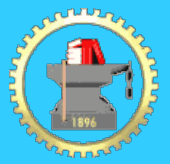


- 定义手机类mobile，包括电话号码mynumber变量以及初始化函数init、拨打电话函数dial、接听电话函数answer、挂断电话函数hangup。
- 【问题分析】

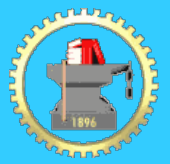
  手机类的定义。
- 【算法描述】

  定义mobile类

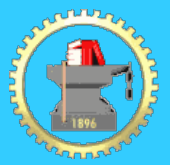
【源程序】



```
#include <iostream>
using namespace std;
class mobile
{
private:
char mynumber[11];//机主的电话号码
public:
  void init(char number[11]=”0000000000”)//隐式定义的内联函数
  {   strcpy(mynumber,number);}
      void dial();        //拨打电话
      void answer(char othernumber[11]);   //接听电话
      void hangup();     //挂断电话
};
```

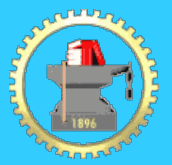


```
void mobile::dial()
{
  cout<<" Dialing number is" <<mynumber<<endl;
  cout<<" Dialing on..." <<endl;
}
void mobile::answer(char othernumber[11])
{
  cout<<" Answering number is" <<othernumber<<endl;
  cout<<" Answering in..." <<endl;
}
inline void mobile::hangup()   //显示定义的内联函数
{
  cout<<" Hanging up..." <<endl;
}
```



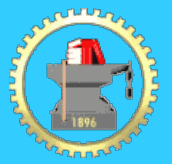
```
int main()
{
  mobile m1;
  m1.init(” 1111111111”); //对象对成员函数
  init的使用
  m1.dial();
  m1.hangup();
  m1.answer(” 2222222222”);
  m1.hangup();
  return 0;
}
```

运行结果



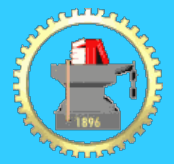
```
Dialing number is1111111111
Dialing on...
Hanging up...
Answering number is2222222222
Answering in...
Hanging up...
```

【思路扩展】



- **日常生活中，还有哪些事物可以定义为类？**

【例6-7】

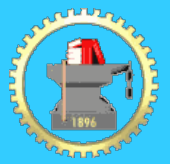


- 定义手机类mobile，包括电话号码mynumber变量，定义初始化函数init、拨打电话函数dial、接听电话函数answer、挂断电话函数hangup，要求dial和init用对象作为函数的参数，实现与【例6-6】相同的功能。
- 【问题分析】

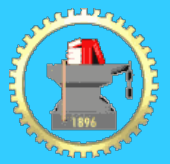
  函数dial和init的定义
- 【算法描述】

  定义mobile类和能够使用该类对象的普通函数dial和init

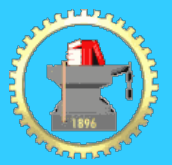
【源程序】



```
#include <iostream>
using namespace std;
class mobile
{public:
   char mynumber[11];     //机主的电话号码
};
void init(mobile &m)      //隐式定义的内联函数
{
cin>>m.mynumber;
}
void dial(mobile m)                  //拨打电话
{
   cout<<" Dialing number is" <<m.mynumber<<endl;
   cout<<" Dialing on..." <<endl;
}
```

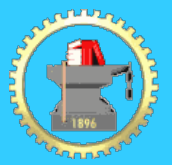


```
void answer(char othernumber[11])//接听电话
{
  cout<<" Answering number is"<<othernumber<<endl;
  cout<<" Answering in..."<<endl;
}
  void hangup()//挂断电话
{
  cout<<" Hanging up..."<<endl;
}
```



```
int main()      //main function
{
  mobile m1;
   init(m1);
  dial(m1);
  hangup();
  answer(" 2222222222" );
  hangup();
   return 0;
}
```

【运行结果】



1111111111

Dialing number is1111111111

Dialing on...

Hanging up...

Answering number is2222222222

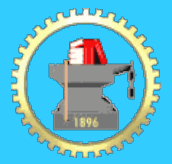
Answering in...

Hanging up...

【思路扩展】

- 【例6-6】与【例6-7】的区别有哪些？

对象与指针



- **指向对象的指针**

  类名 *对象指针名;

- **指向对象成员的指针**

  返回值类型名(类名::*指针变量名)(参数表);

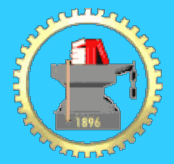
  指针变量名=类名::成员函数名;

  指向对象的指针和指向对象成员的指针都只能访问公有成员。

- **this指针**

  存放对象的起始地址

【例6-8】

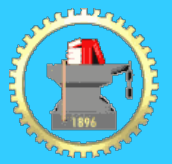


- 利用指向对象的指针以及指向对象成员函数指针完成手机mobile类中拨打、接听等函数的调用。
- 【问题分析】

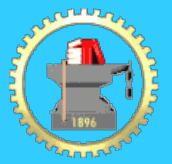
  指向对象的指针、指向对象成员变量的指针以及指向对象成员函数指针的使用
- 【算法描述】

  在mobile类中使用指向对象的指针和指向对象成员函数指针

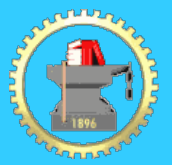
【源程序】



```
#include <iostream>
using namespace std;
class mobile
{
public:
   char mynumber[11];
   void init(char number[11]=”0000000000”)//隐式定义的内联函数
        {strcpy(mynumber,number);}
   void dial();              //拨打电话
   void answer(char othernumber[11]);//接听电话
   void hangup();  //挂断电话
};
```



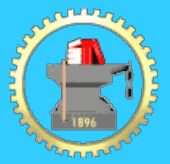
```
void mobile::dial()
{
  cout<<" Dialing number is" <<mynumber<<endl;
  cout<<" Dialing on..." <<endl;
}
void mobile::answer(char othernumber[11])
{
  cout<<" Answering number is" <<othernumber<<endl;
  cout<<" Answering in..." <<endl;
}
void mobile::hangup()    //显示定义的内联函数
{
  cout<<" Hanging up..." <<endl;
}
```



```
int main()
{
  mobile m1;
  mobile *p1=&m1;          //指向对象的指针
  p1->init(”1111111111”);
  p1->dial();
  p1-> hangup();
  p1-> answer(”2222222222”);
  p1-> hangup();
  char *p2;
  p2=m1.mynumber;
  cout<<*p2<<endl;
```

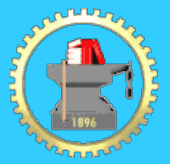
```
    void(mobile::*p3)(char number[11]);
    void(mobile::*p4)();
    void(mobile::*p5)(char othernumber[11]);
    void(mobile::*p6)();
    p3=mobile::init;
    p4=mobile::dial;
    p5=mobile::answer;
    p6=mobile::hangup;
    (m1.*p3)(" 1111111111");
    (m1.*p4)();
    (m1.*p6)();
    (m1.*p5)(" 2222222222");
    (m1.*p6)();
    return 0;
}
```

【运行结果】



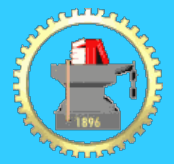
```
Dialing number is1111111111
Dialing on...
Hanging up...
Answering number is2222222222
Answering in...
Hanging up...
1
Dialing number is1111111111
Dialing on...
Hanging up...
Answering number is2222222222
Answering in...
Hanging up...
```

【思路扩展】



- 指向对象的指针、指向对象成员变量的指针以及指向对象成员函数指针会在哪些情况中使用？

6.2.3构造函数



- **构造函数**

  在对象被创建时利用特定的值构造对象，将对象初始化为一个特定的状态。

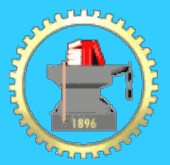
- **拷贝构造函数**

  使用一个已经存在的对象（由拷贝构造函数的参数指定），去初始化同类的一个新对象。

- **析构函数**

  用于完成对象被删除前的一些善后工作

构造函数



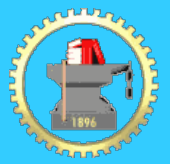
类名（ 形参列表 ）

{

函数体

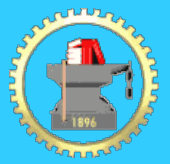
}

拷贝构造函数



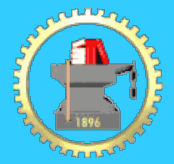
```
类名(类名 &对象名)
{
   函数体;
}
```

析构函数



```
~类名()
{
    函数体
}
```

【例6-9】

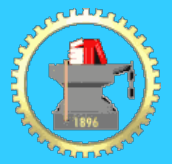


- 定义mobile类，仅成员变量mynumber以及构造函数和析构函数。
- 【问题分析】

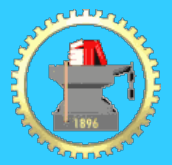
  定义构造函数和析构函数
- 【算法描述】

  定义带构造函数和析构函数的mobile类。

【源程序】

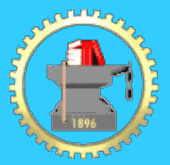


```
#include <iostream>
using namespace std;
class mobile
{
  public:
      char mynumber[11];
      mobile();//不带参数的构造函数mobile
      mobile(char number[11]);// 带参数的构造函数mobile
      ~mobile();
};
mobile::mobile()
{
  strcpy(mynumber,” 0000000000” );
  cout<<mynumber<<endl;
}
```



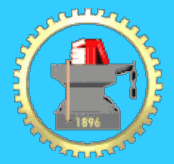
```
mobile::mobile(char number[11])
{
   strcpy(mynumber,number);
   cout<<mynumber<<endl;
}

~mobile()
{
   cout<<" Turn off the phone" <<endl;
}
int main()
{
   mobile m1;
   return 0;
}
```

【运行结果】



```
0000000000
Turn off the phone
```

6.2.4类的组合



- 组合

  类名::类名(形参表):内嵌对象1(形参表)，内嵌对象2(形参表),...

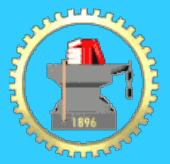
  {

  类的初始化

  }

- 引用
  - **循环依赖**
  - **向前引用**

【例6-10】

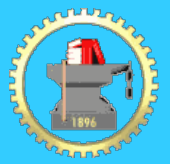


- 定义具有GSM和CDMA两种发射制式的mobile类
- 【问题分析】

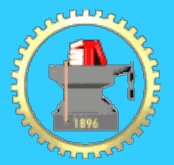
  内嵌对象的使用
- 【算法描述】

  在mobile类中使用内嵌对象、构造函数和析构函数

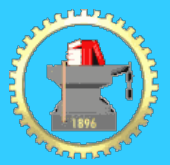
【源程序】



```
#include <iostream>
using namespace std;
class mobilecdma
{
private:
   char mynumber[11];     //机主的电话号码
public:
   mobile(char number[11])//初始化
{
   mynumber=number;
   cout<<" Turn on the mobile" <<mynumber<<endl;
}
   ~mobile()
{ cout<<" Turn off the mobile" <<mynumber<<endl; }
};
```

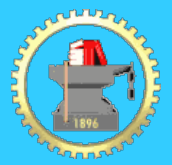


```
class mobile
{
private:
   char mynumber[11];     //机主的电话号码
   mobilecdma core;//CDMA内核，内嵌对象
public:
   mobile(char *number,char *number1):core(number1)//初始化
{
   strcpy(mynumber,number);
   cout<<" Turn on the mobile" <<mynumber<<endl;
}
   ~mobile()
{ cout<<" Turn off the mobile" <<mynumber<<endl;}
};
```



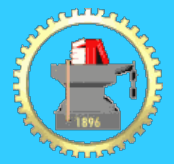
```
int main()
{
  mobile m1(” 1111111111” ,” 2222222222” );
  return 0;
}
```

【运行结果】



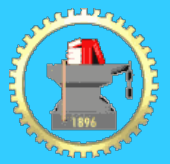
```
Turn on the mobile2222222222
Turn on the mobile1111111111
Turn off the mobile1111111111
Turn off the mobile2222222222
```

6.2.5多文件结构



- 一般把类和成员变量的定义放在一个.h文件，把成员函数和静态成员的定义放在.cpp文件中，主函数放在另外的一个.cpp文件中。

【例6-11】

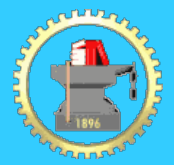


- 使用多文件结构组织mobile类。
- 【问题分析】

  源文件划分
- 【算法分析】

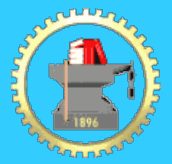
  mobile类的多文件结构

.h

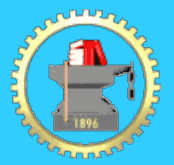


```
//mobil.h文件放mobile的定义
#ifndef MOBILE_H
#define MOBILE_H
class mobile
{
private:
char mynumber[11];//机主的电话号码
  public:
      mobile(char number[11]);//构造函数
void dial();       //拨打电话
      void answer(char othernumber[11]);   //接听电话
      void hangup();     //挂断电话
  ~mobile();       //析构函数
}
#endif
```

.cpp

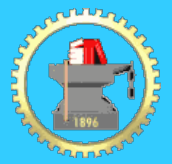


```
//mobile.cpp文件放mobile成员函数的定义
#include <iostream>
#include "mobile.h"
using namespace std;
mobile::mobile(char number[11])//初始化
{
   strcpy(mynumber,number);
   cout<<" Turn on the mobile" <<mynumber<<endl;
}
mobile::~mobile()
{
   cout<<" Turn off the mobile" <<mynumber<<endl;
}
```



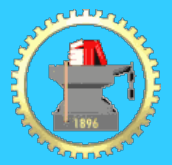
```
void mobile::dial()
{
  cout<<" Dialing number is" <<mynumber<<endl;
  cout<<" Dialing on..." <<endl;
}
void mobile::answer(char othernumber[11])
{
  cout<<" Answering number is" <<othernumber<<endl;
  cout<<" Answering in..." <<endl;
}
void mobile::hangup()
{
  cout<<" Hanging up..." <<endl;
}
```

.cpp



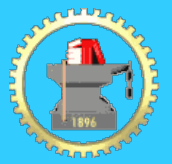
```
//main.cpp放主函数
#include "mobile.h"
#include <iostream>
using namespace std;
int main()
{
  mobile m1(”1111111111”);
  m1.dial();
  m1.hangup();
  m1.answer(”2222222222”);
  m1.hangup();
  return 0;
}
```

【运行结果】



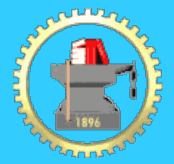
```
Turn on the mobile1111111111
Dialing number is1111111111
Dialing on...
Hanging up...
Answering number is2222222222
Answering in...
Hanging up...
Turn off the mobile1111111111
```

【思路扩展】



- 在什么情况下会使用类的多文件结构？

【例6-12】

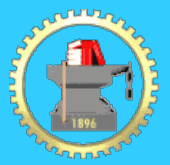


- 对每个学生的成绩进行判断，找出有不及格的课程，然后显示该同学的学号姓名以及不及格课程的名称和分数。
- 【问题分析】

  定义结构体的数据类型student，其成员分别是学号、姓名和一个数组，该数组用来保存5门课的成绩，初始化时给出每个学生的学号、姓名和各门课程的成绩。
- 【算法描述】

  在学生成绩结构体数组中查找不及格信息。

【源程序】



```
#include <iostream>
using namespace std;
int main()
{
struct student
{
   char stno[9];
   char stname[20];
   int score[5];
};
```

```
student stud[10]={ {"08091101","张品",76,87,69,76,78},
                   {"08091102","李利",97,67,79,45,90},
                   {"08091103","应一利",54,69,76,79,56},
                   {"08091104","周勤",87,88,97,99,76},
                   {"08091105","吴风",69,56,98,34,32},
                   {"08091106","赵强",77,87,99,65,76},
                   {"08091107","李力平",91,67,67,87,65},
                   {"08091108","张军利",87,45,77,56,79},
                   {"08091109","冯红",100,69,89,89,76},
                   {"08091110","孙李",99,76,97,96,79} };
char course[5][20]={"数学","物理","化学","英语","计算机"};
int i,j,cnt;
cout<<"每个同学的成绩："<<endl;
cout<<"    学号\t姓名\t数学\t物理\t化学\t英语\t计算机"<<endl;
cout<<"=========================================="<<endl;
```

```
for(i=0;i<10;i++)
{
   cout<<stud[i].stno<<'\t'<<stud[i].stname<<'\t';
   for(j=0;j<5;j++)
       cout<<stud[i].score[j]<<'\t';
   cout<<endl;
}
cout<<"=============================================="<<endl;
cout<<"不及格同学的课程和成绩如下："<<endl;
// 处理不及格分数
for(i=0;i<10;i++)
{  cnt=0;
```

```
    for(j=0;j<5;j++)
        if(stud[i].score[j]<60)
          cnt++;
    if(cnt>0)
    {   cout<< “姓名：”<<stud[i].stname<< “ 学号：
    ”<<stud[i].stno<< “ 不及格门数："<<cnt<<endl;
        cout<<"======================================"<<endl;
        for(j=0;j<5;j++)
              if(stud[i].score[j]<60)
                    cout<<course[j]<<":"<<stud[i].score[j]<<endl;
        cout<<"======================================"<<endl;
    }
}
return 0;
}
```

【运行结果】

每个同学的成绩：

学号	姓名	数学	物理	化学	英语	计算机
08091101	张品	76	87	69	76	78
08091102	李利	97	67	79	45	90
08091103	应一利	54	69	76	79	56
08091104	周勤	87	88	97	99	76
08091105	吴风	69	56	98	34	32
08091106	赵强	77	87	99	65	76
08091107	李力平	91	67	67	87	65
08091108	张军利	87	45	77	56	79
08091109	冯红	100	69	89	89	76
08091110	孙李	99	76	97	96	79

不及格同学的课程和成绩如下：

姓名：李利 学号：08091102 不及格门数：1

==========

英语:45

==========

姓名：应一利 学号：08091103 不及格门数：2

==========

数学:54

计算机:56

==========

姓名：吴风 学号：08091105 不及格门数：3

==========

物理:56

英语:34

计算机:32

==========

姓名：张军利 学号：08091108 不及格门数：2

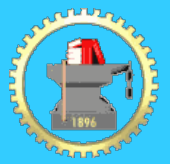
==========

物理:45

英语:56

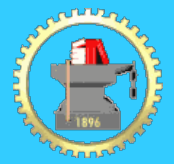
==========

【思路扩展】



- 如果要求找出5门课程都及格的同学，程序应该如何修改？

【例6-13】

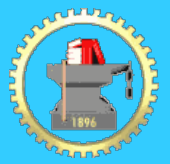


- 定义公民类Citizen，其中Citizen类所包括的内容有：身份证号id、姓名name、性别gende、年龄age;、籍贯birthplace、家庭住址familyAddress等变量，至少两种构造函数，析构函数，修改这六个变量值的函数set，输入和显示公民信息的函数input和output。
- 【问题分析】

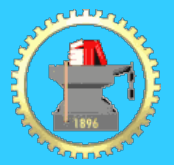
  成员函数的实现。
- 【算法描述】

  定义Citizen类

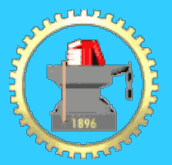
【源程序】



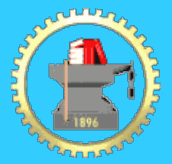
```
#include <cstring>
#include <iostream>
using namespace std;
class Citizen
{
public:
Citizen ()
{ set( "1234" , "张一" , 'm', 18 , "陕西省" , "西安市" );}
Citizen (char id[],char name[],char gender,int age,char
  birthplace[],char familyAddress[])
{ set( id,name,gender,age,birthplace,familyAddress );}
~Citizen (){}
```



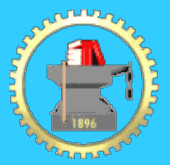
```
void set (char id[],char name[],char gender,int age,char
  birthplace[],char familyAddress[])
{
  strcpy(this->id,id) ;
  strcpy(this->name,name) ;
  this->gender=gender ;
  this->age=age ;
  strcpy(this->birthplace,birthplace) ;
  strcpy(this->familyAddress,familyAddress) ;
}
```



```
void input()
{ cout<<"请输入以下数据："<<endl ;
  cout<<"身份证号\t" ;
  cout<<"姓名\t" ;
  cout<<"性别\t" ;
  cout<<"年龄\t" ;
  cout<<"籍贯\t" ;
  cout<<"家庭住址"<<endl ;
  cin>>this->id ;
  cin>>this->name ;
  cin>>this->gender ;
  cin>>this->age ;
  cin>>this->birthplace ;
  cin>>this->familyAddress ;
}
```

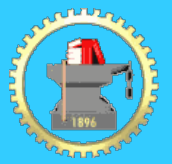


```
void output()
{
   cout<<"身份证号\t" ;
   cout<<"姓名\t" ;
   cout<<"性别\t" ;
   cout<<"年龄\t" ;
   cout<<"籍贯\t" ;
   cout<<"家庭住址"<<endl ;
   cout<<this->id<<'\t' ;
   cout<<this->name<<'\t' ;
   cout<<this->gender<<'\t' ;
   cout<<this->age<<'\t' ;
   cout<<this->birthplace<<'\t' ;
   cout<<this->familyAddress<<endl ;
}
```



```
private:
char id[18] ; // 省份证号
   char name[20] ; // 姓名
   char gender ; // 性别
     int age; // 年龄
   char birthplace[20] ; // 籍贯
   char familyAddress[30] ; // 家庭住址
};
int main()
{ Citizen citizen1;
   citizen1.input();
   citizen1.output();
   Citizen citizen2 ("6101010","李",'f',30,"北京","昌平");
   citizen2.output();
   return 0 ;}
```

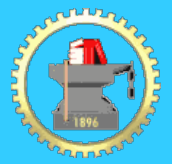
【运行结果】



请输入以下数据：

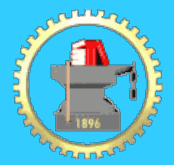
身份证号	姓名	性别	年龄	籍贯	家庭住址
6101012	张	m	20	西安	渭南
身份证号	姓名	性别	年龄	籍贯	家庭住址
6101012	张	m	20	西安	渭南
身份证号	姓名	性别	年龄	籍贯	家庭住址
6101010	李	f	30	北京	昌平

【思路扩展】



（1）能否对每个变量均增加一个修改函数和取值函数，比如对id变量可以定义如下两个函数：

```
void setID(char id[])
{
   strcpy(this->id,id) ;
}
char * getID()
{
   return this->id ;
}
```



（2） 可否把全部成员函数定义在类之外，比如对于set函数的定义可以改为以下形式：

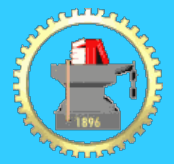
在类中进行声明：

```
void set (char id[],char name[],char gender,int age,
char birthplace[],char familyAddress[]);
```

在类外进行定义：

```
void Citizen::set (char id[],char name[],char gender,int age,
char birthplace[],char familyAddress[])
{ strcpy(this->id,id) ;
  strcpy(this->name,name) ;
  this->gender=gender ;
  this->age=age ;
  strcpy(this->birthplace,birthplace) ;
  strcpy(this->familyAddress,familyAddress) ;}
```

【例6-14】

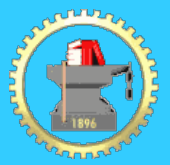


- 定义学生成绩信息类Info，包括学号Id、姓名Name、程序设计课程成绩Programing、计算机网络课程成绩Network、数据库课程成绩Database，总分Total以及设置信息函数函数Set_info、取程序设计课程成绩函数Get_Pro、取计算机网络课程成绩函数Get_Net、取数据库课程成绩函数Get_Dat、取总分成绩函数Get_Tol、显示信息函数Show。编写程序显示3门课程总分从高到低的排名和每门课程成绩都大于85分的学生名单。
- 【问题分析】

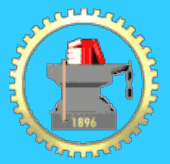
  定义Info类以及在主程序中按要求获取对象的信息。
- 【算法描述】

  对已赋值的类对象按总分排序并查找满足条件的类对象。

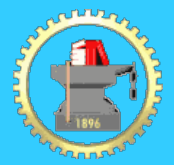
【源程序】



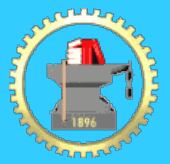
```
#include <iostream>
#include <cstring>
using namespace std;
class Info
{
   int Id;
   char Name[20];
   int Programming;
   int Network;
   int Database;
   int Total;
```



```
public:
   void Set_info(int id,char *name,int programming,int
   network,int database);
   int Get_Pro();
   int Get_Net();
   int Get_Dat();
   int Get_Tol();
   void Show();
};
void Info::Set_info(int id,char *name,int programming,int
   network,int database)
{ Id=id;    strcpy(Name,name);Programming=programming;
   Network=network;Database=database;
   Total=programming+network+database;
}
```



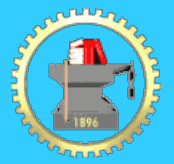
```
int Info::Get_Pro()
{ return Programming;}
int Info::Get_Net()
{ return Network;}
int Info::Get_Dat()
{ return Database;}
int Info::Get_Tol()
{ return Total;}
void Info::Show()
{ cout<<Id<<" \t" ;
  cout<<Name<<" \t" ;
  cout<<Programming<<" \t" ;
  cout<<Network<<" \t" ;
  cout<<Database<<" \t" ;
  cout<<Total<<endl;}
```



```
int main()
{ const int COUNT=5;
  int i=0,j=0;
  int id;
  char name[20];
  int programming;
  int network;
  int database;
  Info student[COUNT],tmp;
  cout<<”请输入学生成绩的信息”<<endl;
  cout<<”学号 姓名 程序设计 计算机网络 数据库”<<endl;
  for(i=0;i<COUNT;i++)
  {   cin>>id>>name>>programming>>network>>database;
      student[i].Set_info(id,name,programming,network,database);}
  cout<<”按总分高低排名如下：”<<endl;
```

```
for(i=0;i<COUNT;i=i+1)
   for(j=COUNT-1;j>i;j=j-1)
       if(student[j].Get_Tol()>student[j-1].Get_Tol())
       {     tmp=student[j];student[j]=student[j-1];
             student[j-1]=tmp;        }
cout<<"学号 姓名 程序设计 计算机网络 数据库 总分"<<endl;
for(i=0;i<COUNT;i++)
   student[i].Show();
cout<<"每门课程成绩都大于85分的学生名单："<<endl;
cout<<"学号 姓名 程序设计 计算机网络 数据库 总分"<<endl;
for(i=0;i<COUNT;i++)
   if(student[i].Get_Pro()>85&&student[i].Get_Net()>85&&student[i]
   .Get_Dat()>85)
       student[i].Show();
return 0;}
```

【运行结果】



请输入学生成绩的信息

学号	姓名	程序设计	计算机网络	数据库
99001	张伟	100	95	90
99002	王伟	70	80	90
99003	李伟	50	60	70
99004	赵伟	70	80	75
99005	张飞	100	100	100

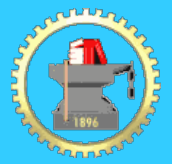
按总分高低排名如下：

学号	姓名	程序设计	计算机网络	数据库	总分
99005	张飞	100	100	100	300
99001	张伟	100	95	90	285
99002	王伟	70	80	90	240
99003	李伟	50	60	70	225
99004	赵伟	70	80	75	180

每门课程成绩都大于85分的学生名单：

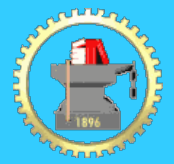
学号	姓名	程序设计	计算机网络	数据库	总分
99005	张飞	100	100	100	300
99001	张伟	100	95	90	285

【思路扩展】

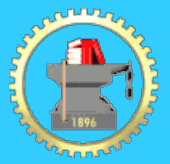


- 本例中为了访问私有的数据成员，专门编写了相应的公有函数，还有其它方法吗？
- 如果要管理的学生不是5名，而是由用户输入的n名学生，应该如何修改程序？

小结



- 类是具有相同的属性和操作方法，并遵守相同规则的对象的集合。
- 类成员的访问控制权限有3种：私有（Private）、公有（Public）和保护（Protected）。
- 成员变量说明为私有可以阻止外界对它的随意访问，说明为公有的成员函数是外界访问类中成员变量的统一接口。
- 一个类的任何成员可以访问该类的其他任何成员，而在该类作用域之外对该类成员的访问则受到一定限制。
- 内联成员函数可以提高程序的执行效率。
- 对象是类的实例。
- 同类对象之间可以整体赋值。
- 一个类的对象可以作为另一个类的成员变量。



- 良好软件工程的一个基本原则是将接口与实现分离。
- 构造函数在每次生成类对象（实例化）时自动调用。
- 对象的生存期结束时，系统自动调用析构函数来撤销该对象。
- 构造函数允许重载，提供初始化类对象的不同方法，析构函数不能重载。
- this指针包含了某个类对象的地址，通过这个地址可以获得该对象的成员变量和成员函数，甚至本身。

- 定义学生类，数据包括姓名、学号、性别信息，行为包括输出学生信息和设置学生信息，给出函数实现。（12分）