

目 录

第 1 章 C++程序设计基础.....	1
1.1 基本知识点、内容概要与学习要求.....	1
1.1.1 基本知识点.....	1
1.1.2 内容概要.....	1
1.1.3 学习要求.....	3
1.2 典型例题（或典型算法）分析.....	4
1.2.1 分析程序运行结果.....	4
1.2.2 程序设计与算法分析.....	6
1.2.3 填空.....	10
1.3 上机实验指导.....	13
1.3.1 实验目的.....	13
1.3.2 实验内容与补充习题.....	13
第 2 章 函数.....	18
2.1 基本知识点、内容概要与学习要求.....	18
2.1.1 基本知识点.....	18
2.1.2 内容概要.....	18
2.1.3 学习要求.....	20
2.2 典型例题（或典型算法）分析.....	20
2.2.1 分析程序运行结果.....	20
2.2.2 程序设计与算法分析.....	23
2.2.3 填空.....	26
2.3 上机实验指导.....	30
2.3.1 实验目的.....	30
2.3.2 实验内容与补充习题.....	30
第 3 章 数组.....	39
3.1 基本知识点、内容概要与学习要求.....	39
3.1.1 基本知识点.....	39
3.1.2 内容概要.....	40
3.1.3 学习要求.....	42
3.2 典型例题（或典型算法）分析.....	42

3.2.1 分析程序运行结果	42
3.2.2 程序设计与算法分析	45
3.2.3 填空	49
3.3 上机实验指导	51
3.3.1 实验目的	51
3.3.2 实验内容与补充习题	52
第4章 类和对象	56
4.1 基本知识点、内容概要与学习要求	56
4.1.1 基本知识点	56
4.1.2 内容概要	56
4.1.3 学习要求	61
4.2 典型例题（或典型算法）分析	61
4.2.1 分析程序运行结果	61
4.2.2 程序设计与算法分析	66
4.2.3 填空	69
4.3 上机实验指导	72
4.3.1 实验目的	72
4.3.2 实验内容与补充习题	72
第5章 程序结构	77
5.1 基本知识点、内容概要与学习要求	77
5.1.1 基本知识点	77
5.1.2 内容概要	77
5.1.3 学习要求	80
5.2 典型例题（或典型算法）分析	80
5.2.1 分析程序运行结果	80
5.2.2 程序设计与算法分析	83
5.2.3 填空	86
5.3 上机实验指导	90
5.3.1 实验目的	90
5.3.2 实验内容与补充习题	90
第6章 指针	95
6.1 基本知识点、内容概要与学习要求	95
6.1.1 基本知识点	95
6.1.2 内容概要	96
6.1.3 学习要求	99
6.2 典型例题（或典型算法）分析	99

6.2.1 分析程序运行结果	99
6.2.2 程序设计与算法分析	104
6.2.3 填空	109
6.3 上机实验指导	115
6.3.1 实验目的	115
6.3.2 实验内容与补充习题	115
第7章 继承与派生	120
7.1 基本知识点、内容概要与学习要求	120
7.1.1 基本知识点	120
7.1.2 内容概要	120
7.1.3 学习要求	123
7.2 典型例题（或典型算法）分析	123
7.2.1 分析程序运行结果	123
7.2.2 程序设计与算法分析	129
7.2.3 填空	133
7.3 上机实验指导	136
7.3.1 实验目的	136
7.3.2 实验内容与补充习题	137
第8章 多态性	142
8.1 基本知识点、内容概要与学习要求	142
8.1.1 基本知识点	142
8.1.2 内容概要	142
8.1.3 学习要求	145
8.2 典型例题（或典型算法）分析	145
8.2.1 分析程序运行结果	145
8.2.2 程序设计与算法分析	150
8.2.3 填空	154
8.3 上机实验指导	156
8.3.1 实验目的	156
8.3.2 实验内容与补充习题	157
第9章 流类库与输入/输出	162
9.1 基本知识点、内容概要与学习要求	162
9.1.1 基本知识点	162
9.1.2 内容概要	162
9.1.3 学习要求	164
9.2 典型例题（或典型算法）分析	165

9.2.1 分析程序运行结果	165
9.2.2 程序设计与算法分析	166
9.2.3 填空	171
9.3 上机实验指导	172
9.3.1 实验目的	172
9.3.2 实验内容与补充习题	173
第 10 章 异常处理	177
10.1 基本知识点、内容概要与学习要求	177
10.1.1 基本知识点	177
10.1.2 内容概要	177
10.1.3 学习要求	179
10.2 典型例题（或典型算法）分析	179
10.2.1 分析程序运行结果	179
10.2.2 程序设计与算法分析	181
10.2.3 填空	184
10.3 上机实验指导	185
10.3.1 实验目的	185
10.3.2 实验内容与补充习题	186
第 11 章 Visual C++环境下 Windows 程序开发概述	187
11.1 基本知识点、内容概要与学习要求	187
11.1.1 基本知识点	187
11.1.2 内容概要	187
11.1.3 学习要求	190
11.2 典型例题（或典型算法）分析	190
11.3 上机实验指导	190
11.3.1 实验目的	190
11.3.2 实验内容	190
附录	192
参考文献	194

第1章 C++ 程序设计基础

1.1 基本知识点、内容概要与学习要求

1.1.1 基本知识点

- C++程序书写格式要求。
- 标识符的定义及命名规则。
- C++数据类型。
- 变量的概念。
- 常量的概念。
- 引用的概念及定义。
- 运算符及表达式的运用。
- 数据如何通过键盘输入及如何在显示器上显示。
- 算法的基本控制结构。

1.1.2 内容概要

1. 标识符

标识符用来标识程序中的一些实体，如函数、变量、类、对象等所起的名字。其构成规则如下：

- 以大写字母、小写字母或下划线(_)开始。
- 可以由大写字母、小写字母、下划线(_)或数字(0~9)组成。
- 大写字母和小写字母代表不同的标识符。
- 不能是C++关键字。

2. C++数据的两种基本形式

C++数据的两种基本形式为变量和常量。每个变量和常量都属于某种数据类型。

3. 变量

变量是指在程序运行的整个过程中其值可以改变的量，应该有一个名字，且在内存中占据一定的存储单元。在使用变量之前需要首先声明其类型和名称。

变量定义的语法形式为

数据类型 变量名1, 变量名2, ..., 变量名n;

常量是指在程序运行的整个过程中其值始终不改变的量。

引用是个别名，当建立引用时，程序用另一个变量或对象(指针型的)的名字初始化它。从那时起，引用作为目标的别名而使用，对引用的改动实际是对目标的改动。引用的说明形式为

类型说明符 &引用名;

表达式由运算符(例如: +、-、*、/)、运算对象(也称操作数, 可以是常量、变量等)和括号组成, 使用形式为

的运算符称为二元运算符(或双目运算符),另一些运算符只需要一个操作数,称为一元运算符(或单目运算符)。

运算符具有优先级与结合性。如果一个运算对象的两边有不同的运算符，首先执行优先级较高的运算；如果一个运算对象两边的运算符的级别相同，则按照运算符的结合性规定的顺序运算。

当表达式中出现了多种类型数据的混合运算时，往往需要进行类型转换。表达式中的类型转换分为两种：隐含转换和强制转换。隐含转换是系统按照以下基本原则自动完成转换的：

强制转换的语法形式如下:

类型说明符(表达式)

或

(类型说明符) 表达式

计算机解决问题的方法和步骤称为算法，它是程序设计的关键。

算法有三种基本控制结构：顺序结构、选择结构和循环结构。

(1) 顺序结构。顺序结构以语句排列的先后次序为执行依据，是程序设计中最基本的结构形式。

(2) 选择结构的支持语句。

- if 选择结构语句的语法形式如下:

形式 1: if(表达式) 语句 1

形式 2: if(表达式) 语句 1

else 语句 2

形式 3: if(表达式 1) 语句 1

else if(表达式 2) 语句 2

else if(表达式 3) 语句 3

⋮

else 语句 n+1

- if 选择结构语句的嵌套形式如下:

if(表达式 1)

if(表达式 2) 语句 1

else 语句 2

else

if(表达式 3) 语句 3

else 语句 4

- switch 选择结构的语法形式如下:

switch(表达式)

{ case 常量表达式 1: 语句 1

case 常量表达式 2: 语句 2

⋮

case 常量表达式 n: 语句 n

default: 语句 n+1

}

(3) 循环结构的支持语句。

- while 语句的语法形式如下:

while(表达式)

循环体

- do-while 语句的语法形式如下:

do

循环体

while(表达式);

- for 语句的语法形式如下:

for(表达式 1; 表达式 2; 表达式 3)

循环体

其中, 循环体可以是单个语句, 也可以是由大括号“{ }”组成的多个语句的复合语句。

1.1.3 学习要求

- 掌握 C++ 基本数据类型的基本概念及其定义方法。
- 了解引用的概念及操作, 为后续章节引用的使用奠定基础。
- 熟练掌握 C++ 语言的各种表达式的正确表示。

- 掌握基本输入/输出语句的使用。
- 熟练掌握算法的选择结构和循环结构的语句形式与使用规则。
- 熟练掌握编制简单的 C++ 程序的方法。

1.2 典型例题（或典型算法）分析

1.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include <iostream.h>
   void main( )
   {
       int x,y;
       cout<<"Enter x and y: ";
       cin>>x>>y;
       if(x!=y)
           if(x>y)
               cout<<"x>y"<<endl;
           else
               cout<<"x<y"<<endl;
       else
           cout<<"x=y"<<endl;
   }
```

运行结果为：

Enter x and y: 45 67

x<y

分析 这是一个比较两个数 x 和 y 大小的程序。将两个数 x 和 y 进行比较，结果有三种可能： $x=y$, $x>y$, $x<y$ 。因此需要进行多次判断，要用多重选择结构，这里选用嵌套的 `if-else` 语句。先判断“ x 不等于 y 吗？”，结果只可能是“是”或“否”。

若结果是“否”，说明 x 不是不等于 y ，也就是说 x 等于 y ，输出结果“ $x=y$ ”。若结果是“是”，则继续嵌套的 `if` 判断“ x 大于 y 吗？”同样结果只可能是“是”或“否”。此时，若结果是“是”，则说明 x 大于 y ，输出结果“ $x>y$ ”；若结果是“否”，则说明 x 不等于 y 且 x 也不大于 y ，那么只可能是 x 小于 y ，输出结果“ $x<y$ ”。

本程序运行后，首先屏幕显示“Enter x and y:”，这是执行第一条语句“`cout<<"Enter x and y:"`；”（此语句起提示作用）与“`cin>>x>>y;`”的结果；然后根据提示输入 45 和 67，即给 x 输入 45，给 y 输入 67，因为 $x!=y$ ，所以就进行“`if(x>y)`”判断，因为此条件不成立，所以就执行“`cout<<"x<y"<<endl;`”，因此屏幕显示“ $x<y$ ”。

2. #include<iostream.h>

```
void main( )
{
    int n,right_digit;
    cout<<"Enter the number:";
    cin>>n;
    cout<<"The number in reverse order is: ";
    do
    {
        right_digit=n%10;
        cout<<right_digit;
        n/=10;
    } while(n!=0);
    cout<<endl;
}
```

运行结果为:

Enter the number: 789

The number in reverse order is: 987

分析 这是一个输入一个整数,将各位数字反转后输出的程序。将一个整数反转输出,即先分离、输出个位,然后是十位、百位……可以采用不断除以 10 取余数,直到商数等于 0 为止(说明已经完成整个数字的翻转输出)的方法。这是一个循环过程,循环的条件就是商“n!=0”。由于无论整数是几,至少要输出一个个位数,因此可以使用 do-while 循环语句,即先执行循环体,后判断循环控制条件。

本程序运行后,首先显示“Enter the number:”,这是执行“cout<<"Enter the number:";”语句的结果,然后按提示要求输入 789,即给了变量 n。回车后屏幕显示“The number in reverse order is:”,这是执行“cout<<"The number in reverse order is: ";”语句的结果。此后进入 do-while 循环,执行“right_digit=n%10;”后, right_digit=9,输出 9,继续执行下面语句(n/=10;),此时 n=78;执行“while(n!=0);”语句,因为条件满足,继续循环, right_digit=8,显示 8, n=7;循环条件满足继续循环, right_digit=7,显示 7, n=0;执行“while(n!=0);”语句,循环条件不满足,退出循环,执行“cout<<endl;”换行,结束程序的执行。

3. #include <iostream.h>

```
void main( )
{
    int n,i,j;
    for(i=100;j<=999;i=i+100)
        for(j=i/100;j<=99;j=j+10)
        {
            n=i+j;
```

```

        cout<<" "<<n;
    }
    cout<<"\n";
}

```

运行结果为:

(略)

分析 这是一个求 100~999 之间的回文数(回文数是指正读与反读都一样的数,如 121)的程序,同时也是—个双重循环的应用程序示例。程序利用了当百位数字为 i ($i=1,2,\cdots,9$) 时, iji ($j=1,2,\cdots,9$) 就是回文数的特点。如果对于任意数要判断其是否为回文数,请参见本书 2.2.2 节第 3 题。

```

4. #include <iostream.h>
   void main( )
   {
       int n,i,k=0;
       cout<<"请输入整数: ";
       cin>>n;
       for(i=1;i<n;i++)
           if (n%i==0) k=k+i;
       if (n==k) cout<<n<<" 是完数\n";
       else cout<<n<<" 不是完数\n";
   }

```

运行结果为:

请输入整数: 28

28 是完数

分析 这是一个判断某一个整数是否为完数(完数是指真因子之和等于自身的整数,如 $6=1+2+3$)的程序。此题的关键是对整数 n 的各因子的判断,循环“for($i=1;i<n;i++$) if ($n\%i==0$) $k=k+i$,”实现对小于 n 的整数逐一进行整除判断,若能被整除,自然就是真因子,则将其加入变量 k 中;然后将 k 与 n 进行比较判断,当 k 等于 n 时,则 n 就是完数,否则就不是完数。

此题中变量 k 起记录累计因子和的作用,初值要被赋为 0。for 循环中的循环控制表达式“ $i<n$ ”可否改为“ $i\leq n/2$ ”,请读者思考。进一步考虑编写程序,求任意两个整数之间的所有完数。

1.2.2 程序设计与算法分析

1. 用公式“ $\pi/4\approx 1-1/3+1/5-1/7+1/9\cdots$ ”求 π 的近似值,直到最后一项的绝对值小于 10^{-8} 为止。

分析 首先根据公式可求出 $\pi/4$ ，然后乘以4即可得 π 的值。分析求 $\pi/4$ 的公式，根据前后项的关系，可以设计一个循环，项值为 $x=1/n$ ($n=1,2,\dots$)，循环的条件是项值 $x>10^{-8}$ ，循环体为：累加项值到和单元s，将原项分母加2，符号取反，求得新项。程序代码如下：

```
#include<iostream.h>
#include<math.h>
#include<iomanip.h>
void main( )
{
    double s=0,x=1;
    long k=1;
    int sign=1;
    while(fabs(x)>1e-8)
    {
        s+=x;
        k+=2;
        sign*=-1;
        x=sign/double(k);
    }
    s*=4;
    cout<<"\the pi is "
        <<setprecision(9)
        <<s<<endl;
}
```

运行结果为：

the pi is 3.14159263

此程序中，要求 π 的值，就需定义一个变量来存储 π 的结果。变量需先声明后使用，而且应声明为实型双精度double型。因为float型的有效位数是7位，而该问题中的最小项的精度要求达到小数点后8位，所以存放累加和 π 的变量s和项值的变量x都应为double型。同理，分母变量k应声明为整型，而且应为long型，因为分母的最大值不超过 10^8 ，int型的范围是无法满足的。

在求新项值x时，用到了 $\text{sign}/\text{double}(k)$ 算术表达式，它把k强制转换为double型，这是因为如果用 sign/k 算术表达式，则结果将为整数，这样会导致在k大于1情况下，其结果都为0，也就得出了 $\pi=4$ 的错误结果。而 $\text{sign}/\text{double}(k)$ 算术表达式的结果为double型，因为 $\text{double}(k)$ 为double型，系统会自动地进行隐含转换，即把sign也自动转换为double型，所以其结果为double型。

2. 运输公司对用户计算费用。路程(s，单位 km)越远，每公里运费越低。标准如下：

$s < 250$	没有折扣
$250 \leq s < 500$	2%折扣
$500 \leq s < 1000$	5%折扣
$1000 \leq s < 2000$	8%折扣
$2000 \leq s < 3000$	10%折扣
$3000 \leq s$	15%折扣

设每公里每吨货物的基本运费为 p ，货物重为 w ，距离为 s ，折扣为 $d\%$ ，则总运费 f 的计算公式为

$$f = p * w * s * (1 - d\%)$$

分析 从上面的标准可以看出，折扣的“变化点”都是 250 的倍数（250, 500, 1000, 2000, 3000）。利用这一特点，设 c 为 250 的倍数，当 $c < 1$ 时，表示 $s < 250$ ，无折扣；当 $1 \leq c < 2$ 时，表示 $250 \leq s < 500$ ，折扣 $d\% = 2\%$ ；当 $2 \leq c < 4$ 时，表示 $500 \leq s < 1000$ ，折扣 $d\% = 5\%$ ；当 $4 \leq c < 8$ 时，表示 $1000 \leq s < 2000$ ，折扣 $d\% = 8\%$ ；当 $8 \leq c < 12$ 时，表示 $2000 \leq s < 3000$ ，折扣 $d\% = 10\%$ ；当 $c \geq 12$ 时， $d\% = 15\%$ 。程序代码如下：

```
#include<iostream.h>
void main( )
{
    int c,s;
    float p,w,d,f;
    cin>>p>>w>>s;
    if(s>=3000) c=12;
    else c=s/250;
    switch(c)
    {
        case 0: d=0;break;
        case 1: d=2;break;
        case 2:
        case 3: d=5;break;
        case 4:
        case 5:
        case 6:
        case 7: d=8;break;
        case 8:
        case 9:
        case 10: d=10;break;
        case 12: d=15;
    }
    f=p*w*s*(1-d/100.0f);
    cout<<"freight="<<f<<endl;
}
```

在此程序中我们巧妙地进行了路程分段的判断,这是该程序的核心所在,因为 c 、 s 为整型变量,因此 $c=s/250$ 为整数。当 $s \geq 3000$ 时,令 $c=12$,而不使 c 随着 s 的增人而增人,这是为了在 `switch` 语句中,用一个 `case` 可以处理所有 $s \geq 3000$ 的情况。

在求费用 $f=p*w*s*(1-d/100.0f)$ 的算术表达式的值时,如果折扣 $d\%$ 写成 $d/100$,则折扣永远是 0。因为 d 是整数,100 是整型常量,所以 $d/100$ 的结果为整型,在 $d < 100$ 的情况下 $d/100$ 永远为 0。要计算正确可由两种方法来实现:一是像前面介绍过的,写成 $\text{float}(d)/100$;二是写成 $d/100.0f$,即把 100 写成 `float` 型常量。若写成 $d/100.0$,则程序编译会有一个警告性错误,指出“ $f=p*w*s*(1-d/100.0f)$ ”语句把 `double` 型数据转换成 `float` 型,会有精度损失。这是因为对“100.0”,C++默认为 `double` 型常量,所以在计算表达式 $p*w*s*(1-d/100.0f)$ 时,系统自动进行隐含转换,其结果为 `double` 型,当把该值赋给 `float` 型的变量 f 时,根据隐含转换规则,`double` 型的结果就隐含转换为 `float` 型。

3. 对任意给定的正整数 n , 求 $n!$ 中末尾 0 的个数。

分析 对于此题,读者一般首先会想到要计算出 $n!$ 的值,然后再想办法求出末尾 0 的个数。但是会发现即便是 `long int` 型数据,最大也只能表示 $12! \sim 13!$ 之间的数,即当 $n > 13$ 时将无法进行计算。

我们注意到一个数末尾有几个 0,就含有几个 10 的因子。10 为 2 和 5 的乘积,而从 $1 \sim n$ 的所有正数乘积中,2 的因子显然比 5 的因子多,因此只要计算出 $n!$ 中含有因子 5 的个数,即为含有因子 10 的个数,亦为 $n!$ 末尾 0 的个数。故可设计程序代码如下:

```
#include <iostream.h>
void main()
{
    long n;
    int ni,k,sun=0;
    cout<<"请输入一个正整数: ";
    cin>>n;
    for(int i=5;i<=n;i+=5)          //只有 5 的倍数才含 5 的因子,故步长为 5
    {
        ni=i;
        for(k=0;ni%5==0;k++)        //对 ni 求含 5 因子的个数
            ni/=5;
        sun+=k;
    }
    cout<<n<<"!中末尾 0 的个数为: "<<sun<<endl;
}
```

当 n 给定为 1000 时,程序运行结果为:

请输入一个正整数: 1000

1000!中末尾 0 的个数为: 249

4. 猴子吃桃问题。猴子第一天摘下若干桃子，当即吃了一半，不过瘾，还多吃了一个。以后每天如此，至第 10 天，只剩下一个桃子。编写程序，计算第一天猴子摘的桃子数。

分析 根据题意，可知猴子共吃了 9 次桃子，而前一次拥有的桃子数总是后一次拥有的桃子数加 1 的 2 倍，已知最后只有 1 个桃子，所以可设计程序代码如下：

```
#include <iostream.h>
void main( )
{
    int day,n1,n2;
    day=9;
    n2=1;
    while (day>0)
    {
        n1=(n2+1)*2;    // 前一天的桃子数是第二天桃子数加 1 后的 2 倍
        n2=n1;
        day--;
    }
    cout<<"第一天共摘的桃子数为："<<n1<<endl;
}
```

运行结果为：

第一天共摘的桃子数为：1534

1.2.3 填空

1. 以下程序根据输入的三角形的三边判断是否能组成三角形，若可以，则输出它的面积和三角形的类型。请填空完成程序。

```
#include <iostream.h>
#include <math.h>
void main( )
{
    float a,b,c;
    float s,area;
    cout<<"Input a,b,c:";
    cin>>a>>b>>c;
    if( ① )
    {
        s=(a+b+c)/2;
        area=sqrt(s*(s-a)*(s-b)*(s-c));
        cout<<area<<endl;
    }
}
```

```

        if( ② )cout<<"等边三角形";
    else
        if( ③ )
            cout<<"等腰三角形";
        else
            if( ④ )
                cout<<"直角三角形";
            else
                cout<<"一般三角形";
    }
    else cout<<"不能组成三角形";
}

```

参考答案:

① $a+b>c \&\& a+c>b \&\& b+c>a$

② $a==b \&\& b==c$

③ $a==b || b==c || a==c$

④ $a*a==b*b+c*c || b*b==a*a+c*c || c*c==a*a+b*b$

分析 本程序的算法是很简单的,只要知道满足三角形、等边三角形、等腰三角形及直角三角形的条件即可。最主要的是把这些条件用正确的C++逻辑表达式加以描述。

显然,①处应判断a、b、c是否组成三角形,②处应判断是否为等边三角形,③处应判断是否为等腰三角形,④处应判断是否为直角三角形。

2. 下面程序的功能是从3个红球、5个白球和6个黑球中任意取出8个球,且其中必须有白球,输出所有的可能方案。请填空完成程序。

```

#include <iostream.h>
void main( )
{
    int i,j,k;
    cout<<"\n  red  while  black\n";
    for(i=0;i<=3;i++)
        for( ① ;j<=5;j++)
            {
                k=8-i-j;
                if( ② )
                    cout<<"\t"<<i
                        <<"\t"<<j
                        <<"\t"<<k<<endl;
            }
}

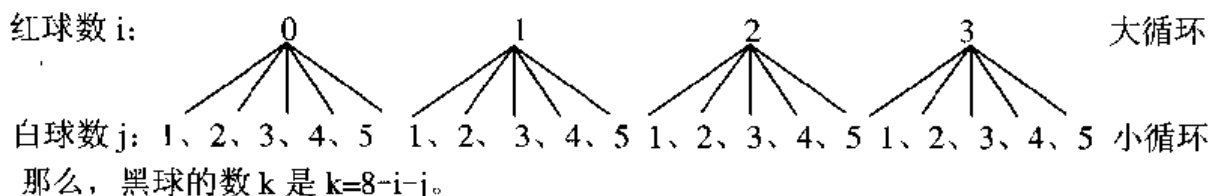
```

参考答案:

① $j=1$

② $k \leq 6$

分析 本程序是要从 3 种颜色的球中取出 8 个,问有多少种取法。若首先从红球中取球,可以有 0、1、2、3 种取法,在这 4 种取法下,白球可以有 1、2、3、4、5 种取法。取法示意如下图所示。



因此可以用循环的嵌套来实现。当 $0 \leq \text{黑球数值} \leq 6$ 时,说明这种取法是合理的,若不满足此条件,说明要求黑球的数大于 6,而黑球没有那么多,也就说明此种取法是不合理的。实际在程序中,大于等于 0 这个条件并没有写出,因为这个条件无论在什么情况下都肯定是满足的,由于红球最大数是 3,白球最大数是 5,因此黑球数最小为 0,所以这个条件在程序中可略去不写,故②处可填 $k \leq 6$,而①空处显然是对白球数的循环,故应填 $j=1$ 。

3. 下面程序实现输出从“!”号开始的 91 个 ASCII 字符及码值,要求每行输出 7 项。请填空完成程序并上机运行验证。

```
#include <iostream.h>

void main( )
{
    char c='!';
    int d,n=1;
    do
    {
        while(n<=7)
        {
            d=c;
            cout<<c<<"="<<d<<" ";
            ①;
            n++;
        }
        cout<<"\n";
        ②;
    } while(③);
    cout<<"\n";
}
```

参考答案:

① `c++`

② `n=1`

③ `c<='!'+90` 或 `c<'!'+91`

分析 本程序是对字符进行处理的简单程序,记录字符的变量 `c` 的初值为“!”,整型变量 `n` 用于对每行输出 7 项的要求进行控制。显然①处应使变量 `c` 表示下一个字符,故可填 `c++` 或 `c=c+1`;②处是换行后重新开始 7 项计数,故应填 `n=1`;对从“!”号开始的 91 个字符进行控制显然是由③处的循环条件来实现的,故③处应填 `c<='!'+90` 或 `c<'!'+91`。

1.3 上机实验指导

1.3.1 实验目的

- (1) 学习如何实现输入输出,掌握常用的转义字符。
- (2) 学习如何把给定的条件书写成 C++合法的逻辑表达式。
- (3) 学习简单的 C++程序的编程方法。

1.3.2 实验内容与补充习题

1. 分析并写出下列程序的运行结果,然后上机运行验证。

```
#include<iostream.h>
void main( )
{
    cout<<"\a How are you? \n";
}
```

运行结果为:

2. 分析下面程序运行结果,并上机运行验证。

```
#include <iostream.h>
void main( )
{
    int x=1,y=0,a=0,b=0;
    switch(x)
    {
        case 1:
            switch(y)
            {
                case 0: a++;break;
```

```
        case 1: b++;break;
    }
    case 2: a++;b++;break;
    case 3: a++;b++;
}
cout<<"a="<<a<<" , b="<<b<<endl;
}
```

运行结果为：

3. 分析并写出下列程序的运行结果，然后上机运行验证。

```
#include<iostream.h>
void main( )
{
    int year,n=0;
    bool IsLeapYear;
    for(year=1990;year<=2000;year++)
    {
        IsLeapYear=((year%4==0&&year%100!=0)||(year%400==0));
        if(IsLeapYear)
        {
            cout<<year<<" is a leap year"<<endl<<endl;
            n++;
        }
    }
    cout<<"There are "<<n<<" leap years.\n";
}
```

运行结果为：

```
_____
_____
_____
_____
```

4. 下面程序是从键盘输入学号，然后输出学号中的百位数字是3的学号，输入0时结束循环。请填空完成程序并上机运行验证。

```
#include <iostream.h>
void main( )
{
```

```

long int num;
cin>>num;
do
{
    if( ① ) cout<<num<<endl;
    cin>>num;
} while( ② );
}

```

5. 下面程序的功能是输出将一元人民币换成一分、两分、五分的所有方案。请填空完成程序并上机调试运行。

```

#include <iostream.h>
#include <iomanip.h>
void main( )
{
    int i,j,k,l=0,s=0;
    for(i=0;i<=20;i++)
        for(j=0;j<=50;j++)
        {
            ① ;
            if (k>=0)
            {
                //setw()用于设置输出域宽
                cout<<setw(3)<<i<<setw(3)<<j<<setw(4)<<k<<",";
                l=l+1;s++;
                if (l%5==0) cout<<endl;
            }
        }
    cout<<"共有方案 "<<s<<" 种"<<endl;
}

```

问题1 ①的位置应填_____。

问题2 语句 if(l%5==0) cout<<endl;的作用是_____。

6. 下面程序实现求一元二次方程 $ax^2+bx+c=0$ 的解。请填空完成程序并上机调试运行。

```

#include<iostream.h>
#include<math.h>
void main( )
{
    float a,b,c,disc,x1,x2,realpart,imagpart;
    cin>>a>>b>>c;

```

```

cout<<"The equation ";
if( ① ) cout<<"is not a quadratic.\n";
else
{
    disc=b*b-4*a*c;
    if( ② ) cout<<"has two equal roots: "<<-b/(2*a)<<endl;
    else
        if( ③ )
        {
            x1=( -b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
            cout<<"has distinct real roots: "<<x1<<" and "<<x2<<endl;
        }

    else
    {
        realpart= -b/(2*a);
        imagpart=sqrt(- disc)/(2*a);
        cout<<"has complex roots: \n";
        cout<<realpart<<"+"<<imagpart<<"i"<<endl;
        cout<<realpart<<"-"<<imagpart<<"i"<<endl;
    }
}
}

```

说明 该程序中值得提出的是, 判断 b^2-4ac 是否等于 0 时, 由于 `disc` 是实数, 而实数在计算和存储时会有一些微小的误差, 因此不能直接进行如下判断: `if (disc==0) ...` 因为这样可能会出现本来是零的量, 由于上述误差而被判断为不等于零而导致结果错误。因此采取的办法是判断 `disc` 的绝对值是否小于一个很小的数 (如 10^{-6}), 如果小于此数, 就认为 `disc` 等于 0。

7. 编程求下列表达式的值:

$$\sum_{K=1}^{100} K + \sum_{K=1}^{50} K^2 + \sum_{K=1}^{10} \frac{1}{K}$$

8. 设计程序, 要求对一个不大于 5 位的正整数能:

- (1) 判断出它是几位数;
- (2) 按逆序打印出各位数字, 例如原数为 321, 应输出 123。

9. 每个苹果 0.8 元, 第一天买 2 个苹果, 从第二天开始, 每天买前一天的 2 倍, 直至购买的苹果个数达到不超过 100 的最大值。编写程序求平均每天花多少钱?

10. 设计程序, 实现从键盘输入 $n(0 < n < 100)$ 个整数, 统计并输出:

(1) 其中出现次数最多的整数;

(2) 最大的整数及其出现的次数。

11. 用公式 $e \approx 1 + 1/1! + 1/2! + 1/3! + \cdots + 1/n!$, 求 e 的近似值, 直到最后一项的绝对值小于 10^{-7} 为止。

*12. 用牛顿迭代法求方程

$$2x^3 - 4x^2 + 3x - 6 = 0$$

在 1.5 附近的根。

*13. 两个乒乓球队进行比赛, 各出三人。甲队为 A、B、C 三人, 乙队为 X、Y、Z 三人, 且已通过抽签确定了比赛名单。有人向队员打听比赛的名单, A 说他不和 X 比, C 说他不和 X、Z 比。请编程找出三对赛手的名单。

第2章 函 数

2.1 基本知识点、内容概要与学习要求

2.1.1 基本知识点

- 函数的概念。
- 函数的定义。
- 函数的值调用。
- 函数的引用调用。
- 递归函数的概念、定义与调用。
- 函数的重载、函数模板。
- 系统函数的使用。

2.1.2 内容概要

1. 函数的概念

函数是 C++ 语言的重要概念，每个函数一般都具有相对独立的功能。尽可能地引用系统提供的库函数，合理地编写用户自己定义的函数，不但可以简化程序的设计过程，更重要的是可以使程序具有模块化结构，便于阅读和调试。

高级语言中的函数（或子程序）就是用来实现这种模块划分的。

在面向对象的程序设计中，函数是功能抽象的基本单位。一个较为复杂的系统往往需要划分为若干子系统，再分别对各子系统进行开发和调试。函数编写好以后，可以被重复使用，使用时只关心函数的功能和使用方法，不必了解函数功能的具体实现。这样有利于代码重用，可以提高开发效率，增强程序的可靠性，也便于分工合作和修改维护。

2. 函数的定义

函数定义的语法形式如下：

```
<返回类型> <函数名> (形式参数表)
{
    函数体;
}
```

3. 函数的调用形式

<函数名>(实参列表)

实参列表中对对应给出与函数原型形参个数相同、类型相符的实参。函数调用可以作为一条语句，此时函数可以没有返回值。函数调用也可以出现在表达式中，此时必须有一个明确的返回值。

1) 值调用

值调用是指当发生函数调用时，给形参分配内存空间，并用实参来初始化形参(直接将实参的值传递给形参)。这一过程是参数值的单向复制传递过程，一旦形参获得了值便与实参脱离关系，此后无论形参发生了怎样的改变，都不会影响到实参。被调用函数的返回值只有通过 `return` 语句才能传回给调用函数，且只能传回一个值。

2) 引用调用

引用是一种特殊类型的变量，可以被认为是另一个变量的别名。若由一个引用作为函数的形参，则当发生函数调用时，并不给引用型形参分配内存空间，而是与对应的实参公用同一存储空间。这就是说，对形参的任何操作也就直接作用于实参，反过来，对实参的任何操作也就直接作用于形参。当函数的返回值有不只一个时，通常可以借助多个引用型参数，而无须使用 `return` 语句。

4. 函数的嵌套调用

一个 C++ 程序可以由一个主函数和若干子函数构成。主函数是程序执行的开始点，由主函数调用子函数，子函数还可以再调用其他子函数，这种调用形式称为函数的嵌套调用。

5. 递归函数

递归函数又称为自调用函数，其特点是在函数内部直接或间接地自己调用自己。递归算法的实质是将原有的问题分解为新的问题，而解决新问题时又用到了原有问题的解法。按照这一原则分解下去，每次出现的新问题都是原有问题的简化的子集，而最终分解出来的问题是一个已知解的问题，这使是有限递归调用。只有有限的递归调用才是有意义的，无限的递归调用永远得不到解，没有实际意义。为了防止递归调用过程无休止地继续下去，在递归函数内必须设置某种条件（通常用 `if` 语句表示），当该条件成立时终止自调用过程，并使程序控制从函数中返回。

递归过程一般分为以下两个阶段：

第一阶段：递推。将原问题不断分解为新的子问题（即规模不断变小），逐渐从未知向已知推进，最终达到已知的条件即递归结束的条件，这时递推阶段结束。

第二阶段：回归。从已知的条件出发，按照递推的逆过程，逐一求值回归，最后达到递推的开始处，结束回归阶段，完成递归调用。

6. 函数的重载

函数的重载使得具有类似功能的不同函数可以使用同一名称，这样便于使用，也能增加程序的可读性。重载函数是按照形参来区分的，同名的重载函数其形参类型或个数必须不同。

7. 函数模板

函数模板实现了类型参数化，并将函数处理的数据类型作为参数，提高了代码的可重用性。

8. 系统函数

C++的系统库提供了几百个可供使用的函数。系统函数的原型声明全部由系统提供，分类存在于不同的头文件中。只需用 `include` 指令嵌入相应的头文件，就可以使用相应的系统函数。

2.1.3 学习要求

- 掌握函数的定义。
- 掌握并区别函数的值调用与引用调用。
- 理解递归函数的概念、定义与调用。
- 了解及运用函数的重载、函数模板、内联函数和带默认值的函数。
- 会正确使用系统函数。

2.2 典型例题（或典型算法）分析

2.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include<iostream.h>
   int func(int a,int b);
   void main( )
   {
       int k=4,m=1,p;
       p=func(k,m);
       cout<<"p=" <<p<<endl;
       k=p+m;
       p=func(k,m);
       cout<<"p=" <<p<<endl;
   }
   int func(int a,int b)
   {
       int m=0,i=2;
       i+=m+1;
```



```

    m=i+a+b;
    return(m);
}

```

运行结果为:

```

p=8
p=13

```

分析 这是一个函数参数值传递的例子。在 `main()` 函数第 4 行首次调用 `func()` 函数, 此时, 实参变量 `k` 和 `m` 的值分别为初值 4 和 1, 函数被调用时 (也是虚实结合时), 系统动态地为形参 `a`、`b` 分配存储空间, 然后将 `k` 的值复制 (传递) 给 `a`, 将 `m` 的值复制 (传递) 给 `b`, 因此, 虚参变量 `a` 和 `b` 的值分别为 4 和 1, 程序的运行轨迹也同时从 `main()` 函数的调用处转入被调函数 `func()` 中。执行到 “`return(m);`” 语句时, 被调函数 `func()` 中的局部变量 `m` 的值为 8, 程序的运行轨迹返回调用函数 `main()` 中的同时, `m` 的值被赋值给 `p`, 因此, 第 5 行的输出语句显示的运行结果为 “`p=8`”。同理, 当第 7 行第二次调用 `func()` 函数时, 实参变量 `k` 和 `m` 的值分别为初值 9 和 1 (此处的 `m` 是 `main()` 函数中的局部变量, 与被调函数 `func()` 中的同名局部变量 `m` 都是独立存在的), 因此第 8 行的输出语句显示的运行结果为 “`p=13`”。

2. #include<iostream.h>

```

void computeCircle(double &area, double &circumference, double r)
{
    const double PI=3.1415926;
    area=PI*r*r;
    circumference =2*PI*r,
}
void main( )
{
    double r,a,c;
    cout<<"输入圆半径: ";
    cin>>r;
    computeCircle(a,c,r);
    cout<<"面积="<<a<<" , 周长="<<c<<endl;
}

```

运行结果为:

```

输入圆半径: 100
面积=31415.9, 周长=628.319

```

分析 这是在一个函数中包含一个普通变量参数及两个引用型参数, 并使函数能返回给定半径的圆的面积和周长的例子, 即当函数的返回值有不止一个时, 可以借助多个引用

型参数，而无需使用 return 语句。程序 main() 函数第 5 行 “cin>>r;” 语句，先给实参变量 r 由键盘输入一个数值（假定输入 100），实参 a 及 c 为普通变量。第 6 行为调用 func() 函数语句，执行此语句时，函数 func() 被调用（首先进行虚实参数结合），实参变量 r 的值被复制给虚参 r，虚参 area 是实参 a 的别名，虚参 circumference 是实参 c 的别名，即 area 与 a 公用同一存储空间，circumference 与 c 公用同一存储空间，程序的运行轨迹也同时从调用处转入被调用函数中。执行 func() 函数体中第 2 条语句后，area 中的值为 31 415.9，则 a 中的值也就为 31 415.9；执行 func() 函数体中第 3 条语句后，circumference 中的值为 628.319，则 c 中的值也就为 628.319。func() 函数体执行完毕，程序的运行轨迹返回调用函数 (main() 函数) 中，易知输出语句 “cout<<“面积=”<<a<<“, 周长=”<<c<<endl;” 的运行结果为 “面积=31415.9, 周长=628.319”。

3. #include<iostream.h>

```
void func(double x, int &part1, double &part2)
{
    part1=int(x)+1000;
    part2=(x+1000-part1)*100;
}

void main( )
{
    int n;
    double x, f;
    x=1002.0703;
    func(x,n,f);
    cout<<"Part1="<<n<<“, Part2="<<f<<endl;
}
```

运行结果为：

Part1=2002, Part2=7.03

分析 这是一个函数参数既有普通变量又有引用的例子，函数的返回值不止一个。程序中 main() 函数第 6 行为调用 func() 函数，此时，实参变量 x 的值为 1002.0703，实参 n 及 f 为一般变量。函数被调用时（也是虚实结合时），实参 x 的值被复制给虚参 x，part1 成为 n 的别名，part2 成为 f 的别名，因此，虚参变量 x 的值为 1002.0703，n 与 part1 公用同一存储空间，f 与 part2 公用同一存储空间，程序的运行轨迹也同时从调用处转入被调用函数 func() 中。执行 func() 函数体中第 1 条语句时，int(x) (x 取整) 的结果为 1002，因此，part1 (或 n) 的值为 2002。执行 func() 函数体中第 2 条语句时，part2 (或 f) 的值为表达式 “(x+1000-part1)*100” 即运算式 “(1002.0703+1000-2002)*100” 的值 7.03。func() 函数体执行完毕，程序的运行轨迹返回调用函数 (main() 函数) 中，输出语句显示的运行结果为 “Part1=2002, Part2=7.03”。

2.2.2 程序设计与算法分析

1. 编写函数，求两个自然数 M 和 N 的最大公约数及最小公倍数。

分析 最大公约数就是能同时整除 M 和 N 的最大正整数，用欧几里德算法（也称辗转相除法）求解，其步骤如下：

第一步：输入两个自然数 M 和 N 。

第二步：求余数 $R(0 \leq R < N)$ 。

第三步：置换。 $N \rightarrow M, R \rightarrow N$ 。

第四步：判断。当 $R \neq 0$ 时，返回第二步；当 $R = 0$ 时，顺序执行第五步。

第五步：输出结果。 M 为所求最大公约数。

最小公倍数 $= M \times N / \text{最大公约数}$

源程序代码如下：

```
#include <iostream.h>
int iZuidagys(int m,int n);
void main( )
{
    int m,n,u,v;
    cout<<"输入一个自然数:";
    cin>>m;
    cout<<"输入另一个自然数:";
    cin>>n;
    u=iZuidagys(m,n);
    v=m*n/u;
    cout<<m<<"和"<<n<<"的最大公约数:"<<u<<endl;
    cout<<m<<"和"<<n<<"的最小公倍数:"<<v<<endl;
}
int iZuidagys(int m,int n)
{
    int temp;
    if(m<n)
    {
        temp=m;
        m=n;
        n=temp;
    }
    while(n!=0)
    {
        temp=m%n;
        m=n;
```

```

        n=temp;
    }
    return m;
}

```

运行结果为:

输入一个自然数: 90

输入另一个自然数: 72

90 和 72 的最大公约数: 18

90 和 72 的最小公倍数: 360

2. 在屏幕上显示杨辉三角形

```

          1
        1 1
       1 2 1
      1 3 3 1
     1 4 6 4 1
    1 5 10 10 5 1

```

注: 杨辉三角形中的数, 恰好是 $(x+y)$ 的 N 次方展开式中各项的系数。

分析 从杨辉三角形的特点出发, 可以总结出如下规律:

(1) 第 N 行有 $N+1$ 个值 (起始行是第0行);

(2) 当 $N \geq 2$ 时, 对于第 N 行的第 J 个值: 当 $J=1$ 或 $J=N+1$ 时, 其值为1; 当 $J \neq 1$ 且 $J \neq N+1$ 时, 其值为第 $N-1$ 行的第 $J-1$ 个值与第 $N-1$ 行第 J 个值之和。将此特点用数学公式描述如下:

$$yh(x,y)=\begin{cases} 1 & x=1 \text{ 或 } x=N+1 \\ yh(x-1,y-1)+yh(x-1,y) & x \neq 1 \text{ 且 } x \neq N+1 \end{cases}$$

以上数学公式是递归形式, 编程时可使用递归函数实现。

程序代码如下:

```

#include <iostream.h>
#include <iomanip.h>
int yh(int x,int y);
void main( )
{
    int i,j,n;
    do
    {
        cout<<"输入要打印的杨辉三角形的行数(>=2):";
        cin>>n;
    }
    while(n<2);
    for(i=0;i<n;i++)
    {
        for(j=0;j<=i;j++)
            cout<<yh(i,j)<<" ";
        cout<<endl;
    }
}

```

```

}while(n<2);          //控制输入正确的值以保证图形的正确
for(i=0;i<=n;i++)      //控制输出 N 行
{
    for(j=0;j<36-3*i;j++)
        cout<<" ";          //控制输出第 i 行前面的空格
    for(j=1;j<i+2;j++)
        cout<<setw(6)<<yh(i,j); //输出第 i 行的第 j 个值
    cout<<endl;            //行末的回车换行
}
}
int yh(int x,int y)
{
    if(y==1||y==x+1)
        return 1;
    else
        return yh(x-1,y-1)+yh(x-1,y);
}

```

运行结果为:

输入要打印的杨辉三角形的行数(>=2):10

```

          1
        1  1
      1  2  1
    1  3  3  1
  1  4  6  4  1
1  5 10 10 5  1
1  6 15 20 15 6  1
1  7 21 35 35 21 7  1
1  8 28 56 70 56 28 8  1
1  9 36 84 126 126 84 36 9  1
1 10 45 120 210 252 210 120 45 10 1

```

3. 用牛顿迭代法求方程 $e^{-x}-X=0$ 在 $x_0=-2$ 附近的一个实数根, 直到满足 $|X_n-X_{n+1}|<10^{-6}$ 。

分析 方程 $f(x)=0$ 的实根在几何图形上为曲线 $f(x)$ 与 x 轴的交点 x^* , 可用对 $f(x)$ 求导的方法, 从 x_0 开始, 使方程的根逐渐逼近 x^* , 这就是牛顿迭代法的基本思想。所谓迭代就是重复执行一组指令, 不断用新值去迭代老值, 直到达到求解精度为止。

牛顿迭代公式为

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad n=0,1,2,\dots$$

程序代码如下:

```
#include <iostream.h>
#include <math.h>
double fun(double x)
{
    return exp(-x)-x;
}
double fundao(double x)
{
    return -exp(-x)-1;
}
void main( )
{
    int n=0;double x0,x1;
    cout<<"输入 x1: "<<endl;
    cin>>x1;
    do
    {
        x0=x1; n++;
        x1=x0-fun(x0)/fundao(x0);
        cout<<"x("<<n<<" )="<<x1<<endl;
    }while(fabs(x1-x0)>=1.0e-6);
}
```

运行结果为:

```
输入 x0: -2
x(1)=-0.880797
x(2)=-0.084275
x(3)=0.519307
x(4)=0.566723
x(5)=0.567143
x(6)=0.567143
```

2.2.3 填空

1. 下面程序中的函数可计算 $S=1+1/2!+1/3!+\cdots+1/n!$, 请填空完成程序。

```
#include <iostream.h>
double fun(int n)
{
```

```

double s=0.0,fac= ①;
int j;
for(j=1;j<=n;j++)
{
    fac=fac ②;
    s=s+fac;
}
return s;
}
void main( )
{
    cout<<fun(6)<<endl;
}

```

参考答案:

① 1.0

② /j

分析 本题的函数算法思想是第 n 个累加项是第 $n-1$ 个累加项的值除以 n 。

2. 以下程序中的函数 iSwsun(int iNum)可计算任意整数 iNum 的各位数字之和 (例如 -1782 的各位数字之和是 $1+7+8+2=18$)，通过调用该函数可计算任意输入的 5 个整数的各位数字之和。请填空完成程序。

源程序代码如下:

```

#include <iostream.h>
int iSwsun(int iNum);
void main( )
{
    int iCount,iNum,iTotal;
    iCount=5;
    iTotal=0;
    cout<<"输入 5 个整型数: "<<endl;
    while(iCount>0)
    {
        cin>>iNum;
        iTotal += ①;
        ②;
    }
    cout<<"5 个整型数的数位之和是"<<iTotal<<endl;
}
int iSwsun(int iNum)

```

```
{  
    if(iNum<0) iNum=-iNum;  
    int iSum=0;  
    do  
    {  
        _____③_____;  
        iNum=iNum/10;  
    }while(____④____);  
    return(iSum);  
}
```

完成程序后，试运行结果样例如下：

输入 5 个整型数：

-182

19658

-2002

75

-3

5 个整型数的数位之和是 59

参考答案：

① iSwsun(iNum)

② iCount--

③ iSum +=iNum%10

④ iNum

分析 本题的算法思想是：对输入的一个整数，先求其绝对值，然后分解各位上的数字（从个位开始分解），边分解边进行累加，直到分解完毕。在计算一个整数的各位数字之和的函数 iSwsun() 中，第 2 行的作用是，被分解数 iNum 若是负数，则将其转化为正数；第 3 行的 iSum 变量用于存放整数的各个数位之和，初值为 0；第 6 行应能从最低位开始分离一个数字(num 除 10 得到的余数)并累加，故③处应填 iSum +=iNum%10；第 7 行扔掉被分解数 iNum 的最低位数字（该数字已被累加过，采用除 10 后取整的方法），显然循环结束的条件应为“iNum 等于 0”，故④处可填 iNum 或 iNum!=0。

在主函数 main() 中，变量 iCount 和 iTotal 分别用于表示计数器 and 累计和，iNum 用于存放每次输入的整数。语句“iTotal +=__①__;”实现各整数位数之和的累加，由此可知①处应填 iSwsun(iNum)，②处应填 iCount=iCount-1 或 iCount--，以实现循环次数的控制。

3. 下面程序中的函数可判断任一整数是否为回文数（回文数即左读右读都相同的数，如 12 321），凡通过函数调用求出 10~200 之间满足 n 、 n^2 、 n^3 均为回文数的数 n 。请填空完成程序。

程序代码如下：


```

#include <iostream.h>
void main( )
{
    bool bSame(long lOld);
    long lNum;
    for(lNum=11;lNum<200;lNum++)
        if( _____ ① _____ )
            cout<<"n="<<lNum<<" n*n="<<lNum*lNum<<" *n*n="<<lNum*lNum*lNum<<endl;
}
bool bSame(long lOld)
{
    long copy_lOld, lNew;
    copy_lOld=lOld; lNew=0;
    while(lOld)
    {
        lNew=lNew*10+lOld%10;
        _____ ② _____;
    }
    if( _____ ③ _____ )
        return true;
    else
        _____ ④ _____;
}

```

完成程序后，运行结果为：

```

n=11  n*n=121  n*n*n=1331
n=101  n*n=10201  n*n*n=1030301
n=111  n*n=12321  n*n*n=1367631

```

参考答案：

- ① bSame(lNum)&&bSame(lNum*lNum)&&bSame(lNum*lNum*lNum)
- ② lOld=lOld/10
- ③ lNew==copy_lOld
- ④ return false

分析 通过对程序的阅读分析可知，程序采用除以 10 取余的方法从最低位开始，依次取出该数的各位数字，然后用最低位充当最高位，按反序重新构成新数，比较新数与原数是否相等，若相等，则原数为回文。

程序代码第 4 行为函数原形说明，当被调用函数的定义出现在调用函数定义之后，函数原形说明必须先行出现在调用函数体内的说明部分。被调用函数第 5 行的 while 循环是依

据已有数(IOld)从右边逐位取最低数位构成新数(INew), 比较已有数(IOld)与构成的新数(INew)是否相等, 若相等则是回文, 返回给调用函数一个真值, 否则, 返回给调用函数一个假值。

2.3 上机实验指导

2.3.1 实验目的

- (1) 掌握函数的定义、函数的调用方法。
- (2) 掌握函数中参数的两种不同传递方式。
- (3) 掌握递归函数和嵌套函数的使用。
- (4) 了解函数的重载、函数模板、内联函数、带默认值的函数等概念。

2.3.2 实验内容与补充习题

1. 分析并写出下列程序的运行结果, 然后上机运行验证。

```
(1) #include <iostream.h>
int func1(int a,int b)
{
    int c;
    a+=a;
    b+=b;
    c=func2(a,b);
    return(c*c);
}
int func2(int a,int b)
{
    int c;
    c=a*b%3;
    return(c);
}
void main( )
{
    int x=7,y=17;
    cout<<func1(x,y)<<endl;
}
```

运行结果为:

(2) #include <iostream.h>

```
int n=10;
int func( )
{
    int k=0;
    k=k+n;
    n=n+10;
    return(k);
}
void main( )
{
    int j=1;
    j=func( );
    cout<<j<<" ";
    j=func( );
    cout<<j<<endl;
}
```

运行结果为:

(3) #include <iostream.h>

```
void t(int x,int y,int cp,int dp)
{
    cp=x*x+y*y;
    dp=x*x-y*y;
}
void main( )
{
    int a=4,b=3,c=5,d=6;
    t(a,b,c,d);
    cout<<c<<" "<<d<<endl;
}
```

运行结果为:

(4) #include <iostream.h>

```
void t(int x,int y,int &cp,int &dp)
{
    cp=x*x+y*y;
```

```
        dp=x*x-y*y;
    }
    void main( )
    {
        int a=4,b=3,c=5,d=6;
        t(a,b,c,d);
        cout<<c<<"."<<d<<endl;
    }
```

运行结果为:

```
(5) #include <iostream.h>
int func(int a,int b)
{
    return(a+b);
}
void main( )
{
    int x=2,y=5,z=8,r;
    r=func(func(x,y),z);
    cout<<r<<endl;
}
```

运行结果为:

```
(6) #include <iostream.h>
long func(int n)
{
    if(n>2) return(func(n-1)+func(n-2));
    else return(2);
}
void main( )
{
    cout<<func(4)<<endl;
}
```

运行结果为:

2. 下列程序的功能是利用函数求 π 的值, 公式如下:

$$\pi = 16 \arctan(1/5) - 4 \arctan(1/239)$$

其中, \arctan 用如下形式的级数计算:

$$\arctan(x) = x - x^3/3 + x^5/5 - x^7/7 + \dots$$

直到级数某项的绝对值不大于 10^{-15} 为止; π 和 x 均为 `double` 型。请填空完成程序并上机运行验证。

```
#include <iostream.h>
#include <math.h>
void main( )
{
    double arctan(double x);
    double x1,x2;
    x1=16.0*arctan(1/5.0),
    x2=4.0*arctan(1/239.0);
    cout<<"PI="<<x1-x2<<endl;
}
double arctan(double x)
{
    int i;
    double r,e;
    r=0;
    e=x;
    i=1;
    while(fabs(e)/i>1e-15)
    {
        r = ① ;
        e = ② ;
        i+=2;
    }
    return r;
}
```

提示 本题的关键是求解数列和, 数列本身的特点为: 正负相间(前一项乘以-1即为后一项的符号), 分子总比前一项多乘 x^2 , 分母总比前一项大 2 (第 12 行代表分母的变量 i 每循环一轮递增 2)。为了提高算法的效率, 尽量在计算后一项时, 能利用已计算过的前一项的结果。注意: 第 7 行、第 8 行中为了避免两个整数相除后的结果取整, 使参与除法运算的任一项为实数。否则, 若将 $\arctan()$ 函数的参数写成 $1/5$ 和 $1/239$, 两次调用的返回值均为 0。

3. 下列程序使用函数的重载计算一个三角形和一个圆的面积。用户将被询问是要计算三角形的面积还是想计算圆的面积。根据用户的响应(1 代表三角形, 2 代表圆), 程序能采集用户的输入并计算面积。请填空完成程序并上机运行验证。

```
#include <iostream.h>
double area(double bottom,double height)
{
    return bottom*height*0.5;
}
double area(double radius)
{
    return 3.1415926*radius*radius;
}
void main( )
{
    int xuanze;
    float r,bottom,height;
    cout<<endl<<"请输入您的选择: "<<endl;
    cout<<endl<<"-----"<<endl;
    cout<<endl<<"1-----计算三角形面积"<<endl;
    cout<<endl<<"2-----计算圆面积"<<endl;
    cout<<endl<<"0-----退出该程序"<<endl;
    cin>>xuanze;
    switch(xuanze)
    {
        case 1: cout<<"请输入三角形的底和高: ";
                cin>>bottom>>height;
                cout<<"三角形的面积是"<<_____①<<endl;
                break;

        case 2: cout<<"请输入圆半径: ";
                cin>>r;
                cout<<"圆的面积是"<<_____②<<endl;
                break;

        case 0: cout<<"未作任何计算,退出";
                break;
        default: cout<<"无效选择,退出";
    }
}
```

4. 下面程序是验证哥德巴赫猜想对 2000 以内的正偶数均成立, 即 2000 以内的正偶数都能够分解成为两个素数之和。请填空完成程序并上机运行验证。

分析 为了验证哥德巴赫猜想对 2000 以内的正偶数是成立的, 要将整数分解为两部分, 然后判断分解出的两个整数是否均为素数。若是, 则满足题意; 否则, 重新进行分解和判断。

```
#include <iostream.h>
int isprime(int iNum);
void main( )
{
    int iNum,n,m=0;
    for(iNum=4;iNum<=2000;iNum+=2)
    {
        for(n=2;n<iNum;n++)
            if(①)
                if(②)
                {
                    m++;
                    cout<<iNum<<"="<<n<<"+"<<iNum-n<<" ";
                    if(m%5==0)
                        cout<<endl;
                    break;
                }
    }
    cout<<endl;
}

int isprime(int iNum)
{
    int iChushu;
    bool bFlag=false;
    for(iChushu=2;iChushu<=iNum-1;iChushu++)
    {
        if(iNum%iChushu==0)
        {
            bFlag=true;
            break;
        }
    }
    if(bFlag==false)
```

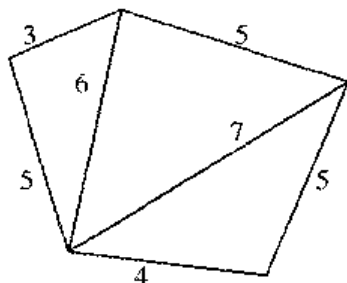
```

    return 1;
else
    return 0;
}

```

提示 被调用函数的第4行设置了一个标志变量 bFlag 并置初值 false; 第7行 iNum 被某数除尽时(余数为零), 就可断定 iNum 不是素数, 且在第9行标志变量值被更改; 第10行用 break 语句跳出 for 循环体(无须再试除后面所有数); 第13行是测试标志变量是否保持为初值 false, 若是, 则 iNum 是素数, 返回给主调函数 1; 第15行标志变量由初值 false 变为 true, 说明 iNum 不是素数, 返回给主调函数 0。

5. 编程求下图所示任意五边形的面积。要求能判断输入的三条边能否构成三角形。若能则求面积, 否则给出错误输入的提示。



提示 求任意五边形面积, 实际就是求三个三角形面积之和。已知一个三角形的三条边 a、b、c, 求其面积可用公式 $S = \sqrt{p(p-a)(p-b)(p-c)}$, 其中 $p = (a+b+c)/2$ 。由于求三角形面积所用算法一致, 只是数值不同, 因此无须编写三段相同的代码, 仅需将求三角形面积算法编写成含有三个形参的函数, 在主调函数中分别以不同的三组实参调用该函数三次并累加其返回值即可。程序代码的第14行用 if (逻辑条件) 语句表示任意两边之和大于第三边, 以判断输入的三个数是否能构成三角形。

6. 古希腊人喜欢用几何形状对数字分类, 例如, 被他们称为三角形的数字是指该数字所代表的小圆石的数目能排列成等边三角形, 如 0, 1, 3, 6, 10, 15, 21, 28 就是所谓的三角形数字。试编写并验证函数 int isTriangular(int n), 若 n 是三角形数字则返回数值 1, 否则返回数值 0。

提示 三角形数 n 具有如下特点: $1+2+3+4+\cdots+m=n$ ($m < n$)。

```

#include <iostream.h>
int isTriangular(int n);
void main()
{
    const int MAX=12;                //整型常量 MAX 表示被验证数的上限
    for(int j=0;j<MAX;j++)
        if(isTriangular(j))
            cout<<j<<"是三角形数字"<<endl;
}

```



```
else
    cout<<j<<"不是三角形数字"<<endl;
}
int isTriangular(int n)
{
    //读者应补充代码的位置
}
```

程序完成后，运行结果为：

```
0 是三角形数字
1 是三角形数字
2 不是三角形数字
3 是三角形数字
4 不是三角形数字
5 不是三角形数字
6 是三角形数字
7 不是三角形数字
8 不是三角形数字
9 不是三角形数字
10 是三角形数字
11 不是三角形数字
```

7. 试编写并验证函数 `bool isSquare(int n)`，若 `n` 是某个整数的平方则返回数值 `true`，否则返回数值 `false`。

```
#include <iostream.h>
bool isSquare(int n);
void main( )
{
    const int MAX=12;           //整型常量 MAX 表示被验证数的上限
    for(int j=0;j<MAX;j++)
        if(isSquare(j))
            cout<<j<<"是平方数字"<<endl;
        else
            cout<<j<<"不是平方数字"<<endl;
}
bool isSquare(int n)
{
    //读者应补充代码的位置
}
```

程序完成后，运行结果为：

0 是平方数字

1 是平方数字

2 不是平方数字

3 不是平方数字

4 是平方数字

5 不是平方数字

6 不是平方数字

7 不是平方数字

8 不是平方数字

9 是平方数字

10 不是平方数字

11 不是平方数字

8. 用递归法计算从 n 个正整数中选择 k 个数的不同组合数。

9. 编写一个函数求满足以下条件的最大的 n ：

$$1^2+2^2+3^2+\cdots+n^2<1000$$

10. 编写一个程序将用户输入的一个十进制数分别转换成二进制数、八进制数和十六进制数。

第3章 数 组

3.1 基本知识点、内容概要与学习要求

3.1.1 基本知识点

1. 数组的概念

- 数组的定义。
- 数组的类型。
- 数组的维数。
- 数组的元素与下标。

2. 数组运算

- 数组元素的引用。
- 数组赋值。
- 数组元素的初始化。

3. 数组与函数之间的关系

- 数组作为函数参数。
- 数组元素作为函数参数。

4. 数组与字符串

- 字符数组。
- 字符串的基本运算 (求字符串的长度、字符串的复制、字符串的连接)。

5. 数组的应用

- 排序。
- 查找。
- 统计。

6. 其他构造数据类型

- 结构体。
- 共用体。
- 枚举类型。

7. 类型自定义语句

3.1.2 内容概要

1. 数组

数组是一种构造数据类型，是具有统一名称和相同类型的一组数据元素的有序集合，它占用连续内存单元进行存储。要引用数组中的特定位置或元素，就要指定数组中的特定位置或元素的位置号 (position number)。如果一个数组的元素本身也是数组，就形成多维数组，如二维数组由若干个一维数组组成，它常用于表示由行和列组成的表格。

2. 数组的定义

1) 一维数组的定义格式

数据类型 数组名[长度];

其中，数据类型指的是每一个数组元素的数据类型。

2) 多维数组的定义格式

数据类型 数组名[第 1 维长度][第 2 维长度]…[第 n 维长度];

3) 数组占用的内存空间的计算

字节数=第 1 维长度×第 2 维长度×…×第 n 维长度×该数组数据类型占用的字节数
即数组要占用内存空间的大小与数组元素的类型、个数有关。

3. 数组元素

数组中用来存储实际数据的一个存储空间称为数组的一个元素。数组元素可存储的数据类型由声明时的数据类型决定，可以用数组名加上方括号 ([]) 中该元素的位置号引用该元素，方括号中的位置号通常称为下标 (subscript)。

其中：

(1) 下标应为整数或整型表达式，其值的允许范围从 0 开始到数组长度-1，下标等于 0 代表要访问的元素在数组的第 1 个位置上。

(2) 一定要注意“数组第 7 个元素”与“数组元素 7”之间的差别，由于数组下标从 0 开始，因此“数组第 1 个元素”的下标为 0，而“数组元素 7”的下标为 7，是第 8 个元素，这常常是“差 1 错误”的原因。

4. 数组的初始化

数组元素可以用定义、赋值或输入进行初始化。

对数组初始化时，可由所赋初值的个数来定义数组的长度。如果初值表中的数据个数比数组元素少，则用 0 来填补不足的数组元素。

对二维数组进行初始化的方式有两种：

(1) 分行赋初值：

(2) 按数组在内存中的排列顺序赋初值。

在进行二维数组定义时，可以省略对第一维长度的说明。这时，第一维的长度由所赋初值的行数确定。

5. 数组与函数

1) 数组作为函数参数

将数组传递给函数实际上传递的是数组存储起始地址,采用地址传送方式可以很好地解决数组中大量数据在函数间传递的问题,在这种方式中,把数组名作为实参来调用函数。在被调用的函数中,以指针变量作为形参接收数组的地址,该指针被赋予数组的地址后,它就指向了数组的存储空间。将函数的参数指定为数组时,方括号中的数组长度不是必需的。

2) 数组元素作为函数参数

数组元素作为函数参数时,每个数组元素实际上代表内存中的一个存储单元,因此和普通变量一样,对应的形参必须是类型相同的变量。

3) 数组元素的引用格式

数组名[元素的下标]

6. 数组与字符串

C++中的字符串是通过字符数组的每一个元素保存一个字符,并用空字符('\0')来结尾进行处理的,因此可以用数组下标元素直接访问字符串中的各个字符。

可以用字符串直接初始化字符数组,也可以用字符常量值列表的方式初始化字符数组。

7. 数组的应用

(1) 排序。排序是指将数组中的所有元素的位置重新排列,使得其值递增或递减有序。

(2) 查找。查找是通过元素值找到元素的所在位置(下标值)或判断元素是否存在。

(3) 统计。当需要知道某个数据出现的次数或某类数据的总体特性时,就要用到统计算法。

8. 其他构造数据类型

(1) 结构体:指出相互关联但类型不同的数据组成的数据结构。

- 结构体的应用:首先应建立一个结构体类型(称为结构体类型的定义),然后再由这个结构体类型去定义相应的变量。

- 结构体中的成员可以是基本数据类型,也可以是另一个结构体类型。

- 结构体变量一般以其成员为单位进行访问。

- 结构体变量本身不代表一个特定的值,但结构体变量可以作为函数参数,同类型结构体变量之间也可以赋值。

- 若干个相同类型的结构体变量也可以用数组的形式来处理,称为结构体数组。

(2) 共用体:指用同一段内存空间来存储不同类型的数据,且对其中一个成员赋值,将影响到其他成员的值。

注意:共用体变量不允许赋初始值。

(3) 枚举类型:是一种允许用符号代表数据的数据类型,其主要功能是增加程序代码的可读性。

9. 类型自定义

使用 typedef 语句可以将一个已有的类型名定义为与其等同的一个新类型名(别名)。

3.1.3 学习要求

- 理解数组作为一种构造数据类型的重要意义。
- 掌握一维数组、二维数组的定义与引用方法。
- 区别数组作为函数参数与一般变量作为函数参数的异同之处,掌握数组作为函数参数的应用方法。
- 了解 C++ 语言中字符串同数组之间的关系,掌握一般字符串的处理方法。
- 掌握数组的基本应用算法(排序、查找、统计等)。
- 了解其他构造数据类型(结构体、共用体、枚举类型)的特点与作用,特别对结构体类型要做比较深入的学习与理解,为以后学习类结构打好基础。
- 了解类型自定义语句的作用与用法。

3.2 典型例题(或典型算法)分析

3.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include <iostream.h>
   void main( )
   {
       int a[10]={76,83,54,62,40,75,90,93,77,85};
       int b[5]={60,70,80,90,101};
       int c[5]={0};
       for(int i=0;i<10;i++)
       {
           int j=0;
           while(a[i]>=b[j]) j++;
           c[j]++;
       }
       for(i=0;i<5;i++) cout<<c[i]<<" ";
       cout<<endl;
   }
```

运行结果为:

2 1 3 2 2

分析 这是一个根据考试成绩,统计各分数段人数的程序。数组 `a` 中存放一组考试成绩,数组 `b` 设定分数段界限值,数组 `c` 用来存放各分数段的统计人数。

程序第 7 行的 `for` 循环针对每一个考试成绩需要进行的处理过程。在此循环过程中, `j` 指向分数段界限值的下标,注意到数组 `b` 中的各分数段界限值是由小到大有序排列的,因此,由循环语句“`while(a[i]>=b[j]) j++;`”就可确定 `j` 的序号值,即分数 `a[i]` 所处的分数段。然后给 `j` 指定的分数段统计变量 `c[j]` 累加 1(即 `c[j]++;`)。最后由循环语句“`for(i=0;i<5;i++) cout<<c[i]<<" ";`”输出各分数段的人数。

```
2. #include <iostream.h>
   void main( )
   {
       int Array[10];
       Array[0]=1;
       for(int i=1;i<10;i++)
       {
           int nValue=Array[i-1]*2;
           if(i%2) nValue=-nValue;
           Array[i]=nValue;
           cout<<Array[i]<<" ";
       }
       cout<<endl;
   }
```

运行结果为:

-2 -4 8 16 -32 -64 128 256 -512

分析 此程序比较简单,给数组 `Array[10]` 中第一个元素赋值为 1,其后各元素均为前一元素的 2 倍。关键是要理解当判断式 `if(i%2)` 为真,即 `i` 不可被 2 整除时其值要反号(`nValue=-nValue;`),同时注意若前一个元素为负数时,即使 `i` 可被 2 整除,其值也是负数(负数的 2 倍不变号),故此程序会出现两负两正相互交替的情况。

```
3. #include <iostream.h>
   void main( )
   {
       char str1[ ]="tHiS is A STRing";
       char str2[100];
       int i=0;
       do
       {
           if(str1[i]>='a'&&str1[i]<='z')
```

```

        str2[i]=str1[i]-'a'+'A';
    else
        str2[i]=str1[i];
    i++;
} while(str1[i]!=NULL);
str2[i]='\0';
cout<<str1<<endl;
cout<<str2<<endl;
}

```

运行结果为:

```

tHiS is A STRing
THIS IS A STRING

```

分析 本程序实现将一个字符串 `str1` 中的小写字母变为大写字母、大写字母保持不变的方式存入另一个字符串 `str2` 中。

程序的关键语句是“`if(str1[i]>='a'&&str1[i]<='z')`”，即判断字符串 `str1` 中各字母是否为小写，若为小写，则需转换为大写后赋给 `str2`，即 `str2[i]=str1[i]-'a'+'A'`；

另外，循环控制表达式“`while(str1[i]!=NULL)`”用来判断字符串 `str1` 是否结束，其中 `NULL` 是在定义并初始化数组 `str1` 时(即“`char str1[]="tHiS is A STRing";`”)由系统自动给定的。而语句“`str2[i]='\0';`”是人为给字符串 `str2` 赋值结束标记，这就是用字符数组存储字符串时需要注意的一点，也可等价地表示为 `str2[i]=NULL;`或 `str2[i]=0;`。

4. #include <iostream.h>

```

void sub(int a[],int n)
{
    int i,j,t;
    for(i=0;i<n;i++)
    {
        for(t=a[i],j=i-1;j>=0&&t<a[j];j--)
            a[j+1]=a[j];
        a[j+1]=t;
    }
}

void main( )
{
    int i,c[ ]={15,3,8,9,6,2,12,7,22,11};
    sub(c,10);
    for(i=0;i<10;i++)
        cout<<c[i]<<" ";
}

```



```

        cout<<endl;
    }

```

运行结果为:

```
2 3 6 7 8 9 11 12 15 22
```

分析 函数 `sub(int a[],int n)` 的功能是用插入法将数组 `a` 的各元素由小到大排序。其基本思路是: 假定 `a` 数组的前 `i-1` 个元素已排好序, 对第 `i` 个元素, 先将 `a[i]` 存入临时变量 `t` 中, `j` 从 `i-1` 开始, 向前寻找第一个比 `t` (此时即 `a[i]`) 小的 `a[j]`; 在寻找过程中, 若 `a[j]` 比 `t` 大, 则将 `a[j]` 向后挪动 (`a[j+1]=a[j];`), 注意此时 `a[i]` 的值已被保存在临时变量 `t` 中; 若一旦找到 `a[j]` 比 `t` 小, 当然是第一次, 由于前 `i-1` 个元素已经排好序, 之前的元素一定都比 `t` 小, 故此时 `j+1` 就是 `a[i]` 应插入的位置, 从而结束循环, 停止挪动, 执行语句 “`a[j+1]=t;`” 实现将 `a[i]` 插入数组 `a` 中第 `j+1` 的位置。

此函数的关键语句就是循环控制语句 “`for(t=a[i],j=i-1;j>=0&& t<a[j];j--)`”, 其中, `j--` 即实现向前比较的过程, `j>=0&&t<a[j]` 在要求 `t<a[j]` 的同时, 下标 `j` 不能比 0 小 (等于 0 即为与 `a` 数组的第一个元素进行比较)。

3.2.2 程序设计与算法分析

1. 有一个数列, 它的第一项为 0, 第二项为 1, 以后每一项都是它的前两项之和, 试产生出此数列的前 20 项并按逆序显示出来。

分析 此题如果不要求按逆序显示, 可以由前两项计算出各项值, 然后依次输出显示。程序代码段如下:

```

#include <iostream.h>
void main()
{
    int a=0,b=1,c;
    cout<<a<<" "<<b<<" ";
    for(int i=1;i<=18;i++)
    {
        c=a+b;
        a=b;
        b=c;
        cout<<c<<" ";
    }
}

```

在此程序中, `a`、`b` 中存放前两项的值, 当 `a` 与 `b` 的和存放到 `c` 中后, 则将 `b` 的值存入 `a`, `c` 的值存入 `b`, 为计算下一项 `c` 作准备。

现在题目要求按逆序显示, 就必须首先考虑将所计算出的各项值进行保存的问题, 这

就要用到数组。程序代码如下：

```
#include <iostream.h>
void main( )
{
    int a[20];
    a[0]=0,a[1]=1;           // 对数列的前两项赋值
    for(int i=2;i<20;i++) a[i]=a[i-1]+a[i-2];
    for(i=19;i>0;i--) cout<<a[i]<<" ";
}
```

此程序中，首先由循环语句“for(int i=2;i<20;i++) a[i]=a[i-1]+a[i-2];”实现对数列 2~20 项值的计算并存入数组 a 中，然后由循环语句“for(i=19;i>0;i--) cout<<a[i]<<" ”;”将数组各元素按逆序进行显示。

2. 编写程序，打印如下的杨辉三角形：

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```

分析 本题与 2.2.2 节中的第 2 题虽说是同一个题，但解决的方法却截然不同，在此题中可将杨辉三角形看作一个方阵的下三角部分，如果使用一个 5 行 5 列的二维数组来存放数据，那么程序的结构比较简单，但本题程序中只使用了一个长度为 5 的一维数组，因此，程序设计中需要一定的技巧。程序代码如下：

```
#include <iostream.h>
void main( )
{
    int a[5];
    for(int i=0;i<5;i++)
    {
        a[i]=1;
        for(int j=i-1;j>0;j--) a[j]=a[j]+a[j-1];
        for(int k=0;k<=5-i;k++) cout<<" ";
        for(j=0;j<=i;j++) cout<<a[j]<<" ";
        cout<<endl;
    }
}
```

从整体上看，整个程序由一个内外嵌套的循环组成。外循环共 5 次，每次循环输出一行数据，循环体中的前两条语句用来在数组 a 中形成当前这一行要输出的数据，第三条语

句用来定位每行的输出位置,后两条语句用来显示数组 a 中的数据。

分析数据的执行流程不难发现,循环体中的第一条语句为“ $a[i]=1$ ”。这条语句在循环中起到“一石二鸟”的作用:第1次循环中($i=0$),使 $a[0]$ 的值为1,从而设置每一行的第一个1;在以后的循环中($i>=1$),使 $a[i]$ 的值为1,即设置每一行末尾的1。

要正确理解此程序,应把握杨辉三角形中的数据特点。从杨辉三角形的第三行起,每一行的首尾两个元素的值均为1,中间的每一个元素都是上一行相应的两个元素之和,即本行元素 $a[i]$ 是上一行元素 $a[i-1]$ 与 $a[i]$ 之和。

根据杨辉三角形中数据的特点,语句“ $a[j]=a[j]+a[j-1];$ ”是利用上一轮的结果来计算本次数据的。注意到程序中只使用一个一维数组,即上一行已形成的数据与本行将形成的数据放在同一个一维数组中,而上一行除首尾两个1以外的每一个数据,在形成下一行数据时都要用到两次。为避免上一行的数据过早地被覆盖,在形成本行数据时,应采用从后向前逐个形成的方法,元素的下标 j 应由大向小变化,因此,要特别注意循环控制结构“ $\text{for}(\text{int } j=i-1;j>0;j--)$ ”的作用。

由于杨辉三角形中的每行数据起始位置都不同,且由多到少缩进空位,因此用循环“ $\text{for}(\text{int } k=0;k<=5-i;k++) \text{cout}<< " ";$ ”来实现每行的输出定位。

最后,请注意 i 在程序中的作用,它控制着杨辉三角形输出的行数与每行输出的元素个数,当然也要影响到数组 a 的大小。

3. 编写程序,找出一个二维数组中的鞍点,即该位置上的元素在该行上最大,在该列上最小。

分析 对一个二维数组,显然满足“在该位置上的元素在该行上最大,在该列上最小”的鞍点存在则惟一或不存在(用反证法易证明)。

解决此问题的思路是:先在某行中找出该行元素的最大值,然后判断该元素在其所处的列是否最小,若是,则输出该元素并结束程序运行;否则,继续下一行的查找,直至二维数组的所有行;若都不满足鞍点要求,则显示该数组无鞍点。程序代码如下:

```
#include <iostream.h>
#define M 3
#define N 3
void main( )
{
    int i,j,a[N][M],max_min,k,flag=0;
    cout<<"请输入数组的各元素值: \n";
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            cin>>a[i][j];
    for(i=0;i<N;i++) //显示数组
    {
        cout<<"\n";
        for(j=0;j<M;j++)
```

```

        {
            cout.width(5);          //设定输出元素的域宽
            cout<<a[i][j];
        }
    }
    for(i=0;i<N;i++)
    {
        k=0;
        max_min=a[i][0];
        for(j=0;j<M;j++)
            if(max_min<a[i][j]) {max_min=a[i][j];k=j;}
        for(j=0;j<M;j++)
            if(max_min>a[j][k]) break;
        if(j==M)
        {
            flag=1;
            cout<<"\n 满足条件的元素为第 "<<i+1<<" 行第 "<<k+1<<" 列的: ";
            cout<<max_min<<endl;
            break;
        }
    }
    if(flag!=1) cout<<"\n 没有满足条件的元素! \n";
}

```

此程序以一个 3 行 3 列的二维数组为示例，程序的前两个循环结构(7~19 行)用来进行数组的输入与显示，在此无须详解。关键的第三个循环结构(20~35 行)完成了鞍点的求解过程，其中，最外层循环“for(i=0;i<N;i++)”对二维数组的行进行控制；k 用来记录某行中最大元素所在的列；max_min 用来表示“行最大、列最小”的元素值；在此循环中又内嵌的两个循环与一个条件判断，其中：

第一个循环

```

    for(j=0;j<M;j++)
        if(max_min<a[i][j]) {max_min=a[i][j];k=j;}

```

完成了对第 i 行进行求最大元素并记录所在列的计算过程；

第二个循环

```

    for(j=0;j<M;j++)
        if(max_min>a[j][k]) break;

```

完成了对所求得的行最大元素 max_min 判断其所在的列中是否为最小的计算过程，若不为最小，立即中断此循环；

条件判断

```

    if(j==M)
    {
        flag=1;
        :
    }

```

中关键是 j 的值, 若上一个循环不是中断(执行 `break;` 语句)结束, 即 j 等于 M , 则意味着 `max_min` 在所处的列中是最小的, 因此, 给找到标志 `flag` 中赋 1(表示鞍点找到)并输出相关的行、列位置与鞍点值信息, 同时中断结束此鞍点的求解循环过程。

最后, 若鞍点的求解循环结束后, `flag` 中的值仍保持 0(不等于 1), 则表示在此二维数组中没有找到满足条件的鞍点, 故输出“没有满足条件的元素!”。

3.2.3 填空

1. 下列程序的功能是将字符串 s 中的所有空格进行压缩删除, 然后输出 s 。请填空完成程序。

```

#include <iostream.h>
void main( )
{
    char s[ ]="We are learning C++ language ";
    for(int i=0,j=0;s[i]!='\0';i++)
        if(s[i]!=' ') _____ (1) _____;
        _____ (2) _____;
    cout<< _____ (3) _____ <<endl;
}

```

参考解答:

① `s[j++]=s[i]`

② `s[j]='\0'或 s[j]=NULL`

③ `s 或 &s[0]` //关于 `&s[0]` 格式参阅第 6 章中的相关内容

分析 本程序中 `for` 循环的功能是对 s 数组中的所有字符逐个进行检查。如果被检查的字符不是空格, 则应保留在数组中, 否则就不保留。这里的所谓保留, 其实采用的是复制方法, 即 `s[j++]=s[i]`, 其中, i 表示被检查字符的下标, j 表示要复制的目标位置, 每次复制完毕后将 j 移到下一个待复制的位置, 即先使用, 后加一。循环结束后, 在 $s[j]$ 中还应补上字符串结束标志 `\0` 或 `NUL`。

使用 `cout<<` 来输出字符串, 可直接用字符数组名作为输出项。`cout` 将从 $s[0]$ 开始逐个输出字符, 直到遇到第 1 个 `\0` 为止。

2. 幼儿园有 n (<20) 个孩子围成一圈分糖果。老师先随机地发给每个孩子若干颗糖果, 然后按以下规则调整: 每个孩子同时将自己手中的糖果分一半给坐在他右边的小朋友; 如共有 8 个孩子, 则第 1 个将原来的一半分给第 2 个, 第 2 个将原有的一半分给第 3 个……

第 8 个将原有的一半分给第 1 个, 这样的平分动作同时进行; 若平分前, 某个孩子手中的糖果是奇数颗, 则必须从老师那里要一颗, 使他的糖果数变成偶数。小孩人数和每个小孩的初始糖果数由键盘输入。下面的程序可求出经过多少次上述调整, 使每个孩子手中的糖果一样多, 调整结束时每个孩子有糖果多少颗, 在调整过程中老师又新增发了多少颗糖果。

```
#include <iostream.h>
#define N 20
int allEqual(int a[],int n)
{
    int i;
    for(i=1;i<n;i++)
        if(a[0]!=a[i]) return 0;
    return 1;
}
int a[N],b[N];
void main( )
{
    int i,n,addk,loopc;
    cout<<"请输入小孩人数:n(<20)\n";
    cin>>n;
    cout<<"输入各小孩初始分配的糖果数: \n";
    for(i=0;i<n;i++)
    {
        cout<<"第 "<<i+1<<" 个小孩: ";
        cin>>a[i];
    }
    addk=0;
    _____ ① _____;
    while(_____ ② _____)
    {
        loopc++;
        for(i=0;i<n;i++)
        {
            if(a[i]%2) {a[i]++;_____ ③ _____;}
            if(i<n-1) b[i+1]=a[i]/2;
            else _____ ④ _____;
            a[i]/=2;
        }
        for(i=0;i<n;i++) _____ ⑤ _____;
    }
}
```

```
cout<<" 共调整 "<<loopc<<" 次数\n";  
cout<<" 调整过程中新增发 "<<addk<<" 颗糖果\n";  
cout<<" 每个孩子有 "<<a[0]<<" 颗糖果\n";  
}
```

参考解答:

- ① loopc=0
- ② allEqual(a,n)==0 或!allEqual(a,n)
- ③ addk++ 或 addk=addk+1
- ④ b[0]=a[i]/2 或 b[0]=a[n-1]/2
- ⑤ a[i]+=b[i] 或 a[i]=a[i]+b[i]

分析 本程序中设计了一个函数 allEqual (int a[],int n), 此函数比较简单, 主要实现判断数组各元素是否都相等, 若相等, 则返回 1 (真); 否则, 返回 0 (假)。调用此函数可判断出各小孩手中的糖果是否相等。显然, 判断式 “if(a[0]!=a[i])” 也可写为 if(a[i-1]!=a[i]), 请读者思考为什么?

在主函数中, 定义了几个整型变量: i, n, addk, loopc。通过对题意要求与程序的对照分析不难发现, i 只是用作一些循环控制变量, n 用来表示小孩的人数, addk 用来存储调整过程中新增发的糖果数, loopc 用来记录调整的次数。由此可见, 在①的位置, 应给 loopc 赋初值, 即 loopc=0, 而②的位置, 显然是当小孩手中糖果不等时应进行调整循环, 故应填 allEqual(a,n)==0 或!allEqual(a,n)的逻辑表达式。

在调整循环体过程中, 首先给记录调整次数的变量 loopc 加 1, 紧接着的 for 循环要处理三个问题: 第一是当小孩手中的糖果是奇数时, 即 if(a[i]%2)为真时, 应增补 1 个 (a[i]++), 使其成为偶数并统计新增发的糖果数, 故在③处应填 addk++或 addk=addk+1; 第二是将每个小孩手中糖果数的一半记录到 b 数组的对应元素 b[i+1]中, 以便将这一半加给下一个小孩 a[i+1], 但考虑到最后一个小孩的一半要加给第一个小孩, 所以当 i 等于 n-1(if(i<n-1)为假)时, 即为最后一个小孩的情况, 故④处应填 b[0]=a[i]/2 或 b[0]=a[n-1]/2; 第三是将每个小孩的一半糖果仍保留在自己手中, 即语句 “a[i]/=2;”。

调整循环过程中的第二个 for 循环自然是完成对每个小孩手中糖果数的重新计算, 故⑤处应填 a[i]+=b[i] 或 a[i]=a[i]+b[i]。

最后的几个输出语句, 完成了调整次数(loopc)、新增糖果数(addk)及最终每个小孩手中的糖果数 (a[0]) 的显示, 其中 a[0]可以用 a[1]~a[n-1]中的任一元素代替 (因为它们都相等)。

3.3 上机实验指导

3.3.1 实验目的

- (1) 学习一维数组、二维数组和字符数组的概念和使用方法。
- (2) 学习下标表达式的常用表示形式。

- (3) 学习将数组作为函数参数的方法。
- (4) 学习冒泡排序法。

3.3.2 实验内容与补充习题

1. 假设下列程序的输入数据是

7 10 5 4 6 7 9 8 3 24 12 15 7 11 21 -1

试分析程序的运行结果并上机验证。

```
#include <iostream.h>
void main( )
{
    int b[50],x,n=0;
    cin>>x;
    while(x>-1)
    {
        b[++n]=x;
        cin>>x;
    }
    for(int i=1,j=0;i<=n;i++)
        if(b[i]%3==0) b[++j]=b[i];
    for(i=1;i<=j;i++) cout<<b[i]<<"\t";
    cout<<endl;
}
```

运行结果为：

2. 分析并写出下列程序的运行结果，然后上机运行验证。

```
#include <iostream.h>
#define SIZE 12
void sub(char a[],int p1,int p2);
void main( )
{
    char s[SIZE];
    for(int i=0;i<SIZE;i++) s[i]='A'+i;
    sub(s,0,SIZE/2);
    sub(s,SIZE/2+1,SIZE-1);
    sub(s,0,SIZE-1);
    for(i=0;i<SIZE;i++) cout<<s[i]<<" ";
    cout<<endl;
}
```



```

}
void sub(char a[],int p1,int p2)
{
    char ch;
    while(p1<p2)
    {
        ch=a[p1];
        a[p1]=a[p2];
        a[p2]=ch;
        p1++;
        p2--;
    }
}

```

运行结果为：

3. 下列程序的功能是在数组 a 中查找与 x 值相同的元素所在位置，数据从元素 $a[1]$ 开始存放，请填空完成程序并上机运行验证。

```

#include <iostream.h>
#define MAX 10
void main( )
{
    int a[MAX+1],x;
    cout<<"请输入数组 a 的各元素:\n";
    for(____①____;i<=MAX;i++)
        cin>>a[i];
    cout<<"请输入要找的元素: ";
    cin>>x;
    a[0]=x;
    i=MAX;
    while(x!=____②____)____③____;
    if(____④____) cout<<x<<" 的位置在第 "<<i<<" 个元素\n";
    else cout<<"没有找到! \n";
}

```

4. 本程序采用筛选法求质数。程序用一个整数数组代表筛，它的每一位对应一个整数。约定数组按自然数的顺序依次对应 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... 程序首先将数组放入筛中，即将数组的全部位置都置成 1，并从筛中找出最小的质数，然后将该质数的倍数从筛中去掉，即将在数组中与它们对应的位置置成 0，则去掉的数是找到的质数的 1 倍、

2 倍、3 倍……反复上述过程，直至筛为空，程序就能求得指定范围(下面程序将范围指定在 100 以内)内的全部质数。要求每行输出 10 个。请填空完成程序并上机运行验证。

```
#include <iostream.h>
#define SIZE 100
#define PWIDTH 10
void main( )
{
    char sieve[SIZE+1];
    int i,n,printcol;
    for(i=0;i<=SIZE;i++)
        ①_____;
    sieve[0]=1;
    printcol=0;
    for(n=2;n<=SIZE;n++)
        if(②_____)
        {
            cout.width(5);           //设置输出域宽
            cout<<n;
            if(③_____)>=PWIDTH)
            {
                cout<<"\n";
                printcol=0;
            }
            for(i=n;i<=SIZE;i=④_____)
                sieve[i]=⑤_____;
        }
    if(printcol!=0)
        cout<<"\n";
}
```

5. 编写程序，通过函数调用将一个十进制正整数 x 转换成二进制数，其中二进制数的每一位放在一个一维数组中。要求在主函数中输入十进制数 x 的值，输出对应的二进制数。

6. 设计程序，用冒泡法对数组 a 中的 10 个数从小到大排序。

冒泡排序方法的基本思想是：

(1) 从元素 $a[0]$ 开始，将相邻的一对元素进行比较，如果前一个元素比后一个元素大，则将它们交换，使小的调到前面，如此向后两两逐对比较(需要则交换)，直到最后一对元素($a[8]$ 和 $a[9]$)。这样，使较小的数值像气泡一样浮到数组前面，较大的数值沉到数组后面，第一轮比较交换下来，最大的数值沉在数组最末一个元素 $a[9]$ 上(最大值到位，在以后的交换中它的位置就不变了)。

(2) 对余下的 9 个元素进行第二轮的两两比较和交换, 使第二大的数值沉在 $a[8]$ 上并使其到位, 再对余下的 8 个元素进行第三轮的两两比较交换, 使第三大的数值沉在 $a[7]$ 上并到位。

如此循环, 当第九大的数值沉在 $a[1]$ 上到位后, 余下的 $a[0]$ 中自然而然地放着最小的数值, 至此排序完毕, 总共进行了九轮循环。

提示 此算法可改进为当某轮两两比较后, 若无交换, 则排序完成。

*7. 编写程序, 对一字符串进行压缩处理, 其要求是: 先压缩掉字符串中多余的空格与相重的字符(只保留一个), 然后把各字符都转换为小写并按英文字典顺序输出显示。例如, 对字符串 “I am a teacher about computer.” 进行压缩处理后应为 “abcehimoprut.”。

*8. 已知某数列的前两项为 2 和 3, 其后继项根据当前最后两项的乘积按下列规则生成:

(1) 若乘积为一位数, 则该乘积即为数列的后继项;

(2) 若乘积为两位数, 则该乘积的十位数字和个位数字依次作为数列的两个后继项。

编写程序, 输出该数列的前 n 项以及它们的和, 要求输入的 n 值必须大于 2, 并且不超过给定的常数值 100。

例如, 若输入 n 值为 10, 则程序输出如下内容:

sum(10)=44

2 3 6 1 8 8 6 4 2 4

*9. 编写程序, 实现将自然数 $1, 2, \dots, N^2$ 按蛇形方式逐个顺序存入 N 阶矩阵。例如, 当 $N=3$ 和 $N=4$ 时的矩阵分别如下图所示。

N=3			N=4			
6	7	9	7	13	14	16
2	5	8	6	8	12	15
1	3	4	2	5	9	11
			1	3	4	10

从 a_{n0} 开始到 a_{0n} 为止 ($n=N-1$) 顺序填入自然数, 交替地对每一斜列从左上元素向右下元素或从右下元素向左上元素存数。

第4章 类和对象

4.1 基本知识点、内容概要与学习要求

4.1.1 基本知识点

1. 面向对象程序设计的概念
 - 结构化程序设计。
 - 面向对象程序设计。
2. 面向对象程序设计的基本特点
 - 抽象性。
 - 封装性。
 - 继承性。
 - 多态性。
3. 面向对象程序设计的语言与标记图
4. 类和对象
 - 类的声明。
 - 类成员的访问属性。
 - 类的成员函数。
 - 对象。
 - 对象数组。
 - 构造函数和析构函数。
 - 拷贝构造函数。
 - 类的组合。
5. 类模板

4.1.2 内容概要

1. 面向对象程序设计的概念

- 1) 结构化程序设计

所谓结构化程序设计，就是自顶向下、逐步求精，将整个程序结构划分成若干个功能相对独立的子模块，并要求这些子模块间的关系尽可能简单；子模块又可继续划分，直至

最简：每一个模块最终都可用顺序、选择、循环三种基本结构来实现。

2) 面向对象程序设计

对象在现实世界中是一个实例(体)或者一种事物的概念。

在面向对象程序设计中，从相同类型的具体对象(实例或事物)中，根据共同的性质与特点抽象出一种新型的数据结构——类。类是一种区别于其他各种一般数据类型的特殊类型，类的成员中不仅包含有描述类对象属性的数据，还包含对这些数据进行处理程序代码，称之为对象的行为（或操作）。

对象是类的实例，对象将其属性和行为封装在一起，并将其内部大部分的实现细节隐藏起来，仅通过一个可控的接口与外界交互。

将对象作为构成软件系统的基本单元而进行程序设计的方法称为面向对象程序设计。

2. 面向对象程序设计的基本特点

1) 抽象性

抽象是指从具体的实例中抽取出共同的性质并加以描述的过程。它一般包括数据抽象和行为抽象两个方面。数据抽象提供了对对象的属性和状态的描述，行为抽象则是对这些数据所需要进行的操作的抽象。

2) 封装性

把某些相关的数据和代码结合在一起，形成一个数据和操作的封装体，这个封装体向外提供一个可以控制的接口，其内部大部分的实现细节则对外隐藏，从而达到对数据访问权限的合理控制。

对象是面向对象程序设计中支持并实现封装的机制。对象中既包含有数据（即属性），又包含有对这些数据进行处理的操作代码（即行为），它们称为对象的成员——数据成员与成员函数。成员可以定义为公有的或私有的，私有成员对外被隐藏，即不能被该对象以外的程序直接访问；公有成员则提供了与外界接口，对象能通过这个接口与外界发生联系；由此实现了封装的两个目标——对数据和行为的包装和信息隐藏。

3) 继承性

继承使一个类(基类)的数据结构和操作被另一个类(派生类)重用，在派生类中只需描述其基类中没有的数据和操作。

(1) 继承关系的特性：

- 类间具有共享特征(包括数据和程序代码的共享)；
- 类间具有差别或新增部分(包括非共享数据和程序代码)；
- 类间具有层次关系。

(2) 继承的分类：

- 从继承源来划分，继承可分为单继承和多继承；
- 从继承内容上划分，继承可分为取代继承、包含继承、受限继承和特化继承。

4) 多态性

多态性是指不同的对象收到相同的消息时产生多种不同的行为方式。

C++中的多态性必须存在于继承环境之中，即在基类中定义的属性和操作被派生类继承之后，可能具有不同的数据类型或表现出不同的行为。

C++支持两种多态性——编译时的多态性和运行时的多态性。编译时的多态性通过重载来实现，运行时的多态性通过虚函数来实现。

3. 面向对象程序设计语言的特征与标记图

(1) 面向对象程序设计语言应具备以下特征：

- 支持对象的概念(包括对象的所有特性，如封装等)；
- 要求对象属于类；
- 提供继承机制。

(2) 面向对象标记图是将系统的构成更加直观地表述出来的一种图示方法，是进行面向对象程序设计的一种辅助工具。它应该能够准确清楚地描述以下四个问题：

- 类；
- 对象；
- 类及对象的关系；
- 类及对象之间的联系。

4. 类和对象

类是面向对象程序设计的基础和核心，也是实现抽象数据类型的工具。

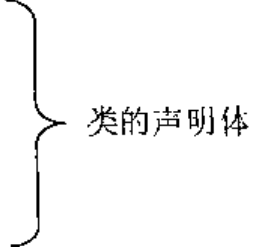
类实际是一种用户自定义的既包含有一组相关的数据，还包含能对这些数据进行处理的功能的特殊数据类型。

我们可以声明属于这种特殊数据类型(类)的变量，这种变量称之为类的对象。由此可见，类和对象的关系实际上是数据类型和具体变量的关系。

1) 类的声明

类的声明即类的定义，其一般形式如下：

```
class <类名>
{
    [private: ]
        私有数据成员的定义和私有成员函数的原型声明；
    protected:
        保护数据成员的定义和保护成员函数的原型声明；
    public:
        公有数据成员的定义和公有成员函数的原型声明；
};
:
<各个成员函数的实现> //成员函数的实现也可直接放在原型声明之后(形成内联函数)
:
```



2) 类成员的访问属性控制

C++规定有三种访问控制属性：

- (1) private: 表示成员是私有的，只允许本类的成员函数访问。
- (2) protected: 表示成员是受保护类型的，也允许其派生类的成员函数访问。
- (3) public: 表示成员是公有的，可以由程序中的任何函数访问。

3) 类的成员函数

一般在类的声明体内只给出成员函数的原型声明, 而将其定义体写在类的声明体之外(后)。一般成员函数定义的形式如下:

```
函数类型 类名::成员函数名([形参表])  
{  
    函数体;  
}
```

与成员函数对应产生了内联函数的概念(即在编译时, 内联函数的函数体被插入到每一次调用它的地方)。

将类中的成员函数定义为内联函数可采用如下两种方式:

(1) 将成员函数的函数体直接放在类声明体中;

(2) 使用 `inline` 关键字来确认, 即在类的声明体中说明函数原型时前面加上 `inline`, 或者在类声明体外定义这个函数时前面加上 `inline`。

4) 对象的定义与引用

(1) 对象的定义形式为

类名 对象名表;

其中, 各对象名之间用逗号“,”分隔。

(2) 对象的引用, 即对对象公有成员的引用。

创建了类的对象之后, 就可以用点运算符“.”来引用类的公有成员, 其一般形式为:

对象名.公有数据成员名

或 对象名.公有成员函数名([实参表])

5) 对象数组

由同一个类的若干对象构成的数组即为对象数组, 此时应注意数组中的各个元素是类对象, 不仅包含数据成员, 还包含成员函数。

声明对象数组的一般形式为

类名 数组名 [<常量表达式>] ...

其中, [<常量表达式>] ... 给出数组的维数和大小。

6) 构造函数、拷贝构造函数和析构函数

(1) 构造函数。构造函数是一种特殊的成员函数, 它主要用于为对象分配空间, 进行初始化。

构造函数作为类的一个成员函数, 具有一般成员函数的所有特性, 它可以访问类的所有数据成员, 可以是内联函数, 可以带有任意类型的参数, 还可以带默认的形参值, 也可以重载。此外, 构造函数还具有一些特殊的性质:

- 构造函数的名字必须与它所属的类名相同, 应声明为公有函数;
- 构造函数不能具有返回类型(值);
- 构造函数不能像其他函数那样被显式地调用, 它在定义对象时被编译系统自动调用,

一般格式为

类名 对象名(初始化实参表);

• 每个类都必须有构造函数，若没有给类定义构造函数，编译器会自动生成一个不带参数的缺省构造函数，其格式如下：

```
类名::类名() { }
```

(2) 拷贝构造函数。拷贝构造函数是重载构造函数的一种特殊形式，它的功能是使用已经存在的对象去创建一个同类的新对象，自定义拷贝构造函数的一般形式如下：

```
类名(const 类名 &形参对象)
{
    拷贝构造函数的函数体;
}
```

同样，每一个类都必须有一个拷贝构造函数，如果没有定义，系统会自动生成一个缺省的拷贝构造函数，该函数完成将一个对象的数据成员逐一复制到另一个对象中。

(3) 析构函数。析构函数也是一种特殊的成员函数。它的功能与构造函数正好相反，是用来释放分配给对象的内存空间的。析构函数有以下一些特点：

- 析构函数的名字是在类名前加一个波浪号“~”而构成。
- 析构函数没有参数，也没有返回值，而且不能重载，因此在一个类中只可能有一个析构函数。
- 析构函数可在程序中被显式调用，也可由系统自动调用。
- 如果一个类中没有定义析构函数，系统会自动生成一个缺省的析构函数，其形式如下：

```
类名::~~类名() { } //该析构函数是一个空函数，什么都不做
```

7) 类的组合

将一个已定义的类的对象作为另一个类的数据成员，称之为类的组合。

在此要强调的是，当声明一个含有对象成员的类时，首先要考虑各对象成员的创建，即构造函数对类内各对象的初始化问题。

组合类构造函数的一般形式如下：

```
类名::类名(形参表): 对象成员1(形参表1), 对象成员2(形参表2), ...
{
    类的初始化程序体;
}
```

5. 类模板

类模板又称为类属类或类生成类。它不是一个具体的、实际的类，而是代表着一个类的类(型)，它允许用户为类定义一种模式，使得类中的某些数据成员、某些成员函数的参数或返回值能以参数的形式取任意数据类型。

声明类模板的一般格式如下：

```
template <class Type>
class 类名
{
```

... //其中 Type 作为某些数据成员、某些成员函数的参数或返回值的数据类型符


```
}
```

其中, Type 是一个标识符, 称为模板参数, 代表类模板中参数化的类型名, 若有多个参数化的类型名, 中间用逗号隔开。

类模板的使用就是将类模板实例化为一个具体的类, 其格式如下:

类名<实际类型> 对象名;

4.1.3 学习要求

- 理解面向对象程序设计与结构化程序设计的概念与特点。
- 理解面向对象程序设计的抽象性、封装性、继承性、多态性等特性。
- 了解面向对象程序设计的语言特征与标记图方法。
- 掌握类的声明方法, 理解类成员的访问控制属性, 学会类成员函数的定义形式。
- 理解对象的概念, 掌握对象的定义方法, 了解对象数组等内容。
- 理解构造函数、拷贝构造函数和析构函数的作用及其使用方法。
- 了解类的组合及其构造函数的定义方式。
- 理解类模板的作用, 基本掌握类模板的使用方法。

4.2 典型例题 (或典型算法) 分析

4.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include <iostream.h>
   class Sample
   {
       int x,y;
   public:
       Sample() { x=y=1; }
       Sample(int a,int b) { x=a;y=b; }
       ~Sample()
       {
           if(x==y)
               cout<<"x=y"<<endl;
           else
               cout<<"x!=y"<<endl;
       }
       void disp()
       {
```

```
        cout<<"x="<<x<<" , y="<<y<<endl;
    }
};
void main( )
{
    Sample s1(2,3),s2;
    s2.disp( );
    s1.disp( );
    cout<<endl;
}
```

运行结果为:

```
x=1 , y=1
x=2 , y=3
x=y
y!=y
```

分析 这是一个反映构造函数重载与调用和析构函数调用次序的程序。在类的声明中, 包含了两个私有数据成员 `x`、`y`; 两个构造函数 `Sample()` 和 `Sample(int a,int b)`, 其中 `Sample()` 无参, 但用常量 1 对两个私有数据成员进行了初始化, `Sample(int a,int b)` 用两个参数对两个私有数据成员进行初始化; 一个析构函数 `~Sample()`, 它用于判断输出两个私有数据成员是否相等; 另外的成员函数 `disp()` 只用于输出数据成员信息。

在主函数 `main()` 中, 用语句“`Sample s1(2,3),s2;`”定义两个类对象 `s1` 和 `s2`, 其中先定义的 `s1` 含有两个实参, 而后定义的 `s2` 无参, 则 `s1` 用构造函数 `Sample(int a,int b)` 对其进行初始化, `s2` 用 `Sample()` 对其进行初始化; 之后由 `s2`、`s1` 各自调用 `disp()` 函数, 显示出相应的数据成员信息 `x=1, y=1` 与 `x=2, y=3`。注意析构函数“`~Sample()`”是在程序结束时自动调用执行的, 其释放对象的次序与定义对象的次序恰好相反, 因此, 先对 `s2` 调用 `~Sample()`, 显然判断输出结果为 `x=y`; 再对 `s1` 调用 `~Sample()`, 判断输出结果为 `x!=y`。

2. #include <iostream.h>

```
class CSample
{
    int i;
    static int k;
public:
    CSample( );
    CSample(int a);
    void Display( );
};
CSample::CSample( ) { i=0; k++; }
```

```

CSample::CSample(int a) { i=a; k=k+a; }
void CSample::Display( )
{
    cout<<"i="<<i<<" , k="<<k<<endl;
}
int CSample::k=0;
void main( )
{
    CSample s1,s2(9);
    s1.Display( );
    s2.Display( );
}

```

运行结果为：

i=0 , k=10

i=9 , k=10

分析 这是一个反映构造函数重载与静态成员变量特性的程序。在类的声明中，包含了一个 int 型私有数据成员 i 和一个静态 int 型数据成员 k；两个构造函数 CSample()和 CSample(int a)，在两个构造函数中不但对私有成员 i 进行了初始化，而且对静态成员 k 进行了运算；另外的成员函数 Display()用于输出数据成员信息。注意，在此题中将类成员函数的实现过程写在了类的声明之外。

在主函数 main()之前，先应对静态成员变量 k 赋初值，其形式为“int CSample::k=0;”。

在主函数 main()中，用语句“CSample s1,s2(9);”定义两个类对象 s1 和 s2，在用构造函数 CSample()对 s1 进行初始化的同时，执行 k++，使 k 的值变为 2；用“CSample(int a)”对 s2 进行初始化的同时，执行 k=k+a，则使 k 的值变为 10；之后由 s1、s2 各自调用 Display()函数，显示出相应的信息 i=0 ,k=10 与 i=9 ,k=10。注意各对象的静态成员变量共享同一拷贝(存储空间)，故 k 的值都为最后更新结果值 10。

3. #include <iostream.h>

```

class A
{
    int a,b;
public:
    A(int i,int j);
    void printA( );
};
class B
{
    A c;

```

```
    int n;
public:
    B(int i,int j,int k);
    void printB( );
};
B::B(int i,int j,int k):c(i,j) { n=k; }
void B::printB( )
{
    cout<<"n="<<n<<" , ";
    c.printA( );
}
A::A(int i,int j)
{
    a=i;b=j;
}
void A::printA( )
{
    cout<<"a="<<a<<" , b="<<b<<endl;
}
void main( )
{
    B m(7,8,2);
    m.printB( );
}
```

运行结果为:

n=2 , a=7 , b=8

分析 这是一个反映类的组合的程序,此题主要注意的是在类的组合情况下,其构造函数的形式与对象成员的初始化。在B类的声明中,不但有普通int型私有数据成员n,还包含了一个A类的对象c作为其中的私有成员,则B类的构造函数形式应为B::B(int i,int j,int k):c(i,j) { n=k; },即用两个形参i、j给其对象成员c进行初始化,k为其普通数据成员n进行初始化。

在主函数main()中,定义了一个B类对象m(7,8,2),在用构造函数“B(int i,int j,int k):c(i,j) { n=k; }”对其进行初始化时,对象成员m.c用7、8(对应i、j的实参)赋初值,普通成员m.n用2(对应k的实参)赋初值,从而使m.c.a=7, m.c.b=8, m.n=2。之后由m调用printB()成员函数,其中为输出对象成员m.c的信息,又嵌套调用“c.printA();”,故最后输出的信息为n=2 , a=7 , b=8。

```
4. #include <iostream.h>
   template <class T>
   class Sample
   {
       T n;
   public:
       Sample(T i) { n=i; }
       void add( );
       void disp( )
       {
           cout<<"n="<<n<<endl;
       }
   };
   template <class T>
   void Sample<T>::add( )
   {
       n+=1;
   }
   void main( )
   {
       Sample<char> s('a');
       Sample<double> t(96.87);
       s.add( );
       t.add( );
       s.disp( );
       t.disp( );
   }
```

运行结果为：

```
n=b
n=97.87
```

分析 这是一个类模板的应用程序。在类模板的声明中，T 代表类型参数，不但定义有 T 类型的私有数据成员 n，还定义了一个含 T 类型参数的构造函数。另外，由于 add() 是模板类的成员函数，在类的声明之外定义应采用“template <class T> void Sample<T>::add()”的形式，而成员函数 disp() 在类的声明之中，故直接定义即可。

在主函数 main() 中，定义了一个 char 类型的对象 s('a') 与一个 double 类型的对象 t(96.87)，且将数据成员 n 分别初始化为字符“a”和双精度实数“96.87”，各自调用成员函数 add() 后都增加 1，故最后的输出为 n=b 和 n=97.87。

4.2.2 程序设计与算法分析

1. 下面是一个类的测试程序, 根据运行结果设计出能使用该测试程序的类。

```
void main()  
{  
    Test a;  
    a.init(68,37);  
    a.show();  
}
```

运行结果为:

测试结果: 68-37=31

分析 本题是要设计 Test 类, 其设计方法很多, 在此只给出一种解法。显然, 可设计 Test 类包括两个(私有)数据成员 x、y, 并包括两个公有成员函数 init()和 show(), 前者用于给数据成员赋值, 后者用于 x、y 的减法运算和输出相应的结果。

```
#include <iostream.h>  
class Test  
{  
    int x,y;  
public:  
    void init(int,int);  
    void show();  
};  
void Test::init(int i,int j)  
{  
    x=i; y=j;  
}  
void Test::show()  
{  
    cout<<"测试结果: "<<x<<"-"<<y<<"="<<x-y<<endl;  
}
```

2. 编写一个程序, 对已有若干个学生数据(包括学号、姓名和成绩), 要求输出这些学生数据并计算输出平均分。

分析 设计一个学生类 Stud, 除了包括 no(学号)、name(姓名)和 chg(成绩)数据成员外, 还设计有两个静态变量 sum 和 num, 分别用于存放总分数和人数, 另有一个构造函数、一个普通成员函数 disp()和一个静态成员函数 agv(), 它们分别用于初始化学生信息对象、显示学生数据和计算平均分。本题程序如下:

```
#include <stdio.h>
#include <string.h>
class Stud
{
    int no;
    char name[10];
    int chg;
    static int sum,num;
public:
    Stud(int n,char na[],int d)
    {
        no=n;chg=d;
        strcpy(name,na);
        sum+=d;
        num++;
    }
    static double avg( )
    {
        return sum/num;
    }
    void disp( )
    {
        printf(" %-5d%-8s%3d\n",no,name,chg);
    }
};
int Stud::sum=0;
int Stud::num=0;
void main( )
{
    Stud s1[]={Stud(1,"Li",89),Stud(2,"Chen",78),Stud(3,"Wang",94)};
    printf("\n 学号   姓名   成绩\n");
    for(int i=0;i<3;i++)
        s[i].disp( );
    printf("平均分=%g\n\n",Stud::avg( ));
}
```

运行结果为:

学号	姓名	成绩
1	Li	89
2	Chen	78
3	Wang	94

平均分=87

3. 设计一个立方体类 Box, 它能计算输出立方体的体积和表面积。

分析 设计 Box 类, 包含三个私有数据成员 a、vol 和 area, 它们分别表示立方体的边长、体积和表面积; 两个构造函数以及一个用于设置立方体边长的函数 seta() (与构造函数赋初值进行比较)。另外, 还应设计计算体积、计算表面积和输出结果的成员函数, 它们分别为: getvol()、getarea() 和 disp()。程序代码如下:

```
#include <iostream.h>
class Box
{
    float a, vol, area;
public:
    Box() { };
    Box(float r) { a=r; }
    void seta(float r) { a=r; }
    void getvol() { vol=a*a*a; }
    void getarea() { area=6*a*a; }
    void disp()
    {
        cout<<"体积: "<<vol<<" , 表面积: "<<area<<endl;
    }
};
void main()
{
    Box obj1(4.5), obj2;
    obj2.seta(6.4);
    obj1.getvol();
    obj1.getarea();
    cout<<"obj1=>";
    obj1.disp();
    obj2.getvol();
    obj2.getarea();
    cout<<"obj2=>";
    obj2.disp();
}
```

完成程序后, 运行结果为:

obj1=>体积: 91.125 , 表面积: 121.5

obj2=>体积: 262.144 , 表面积: 245.76

4.2.3 填空

1. 以下是一个采用类结构的方式求 $n!$ 的程序，请填空完成程序。

```
#include <iostream.h>
class Factorial
{
    int n;
    int fact;
public:
    Factorial(int);
    void Calculate( );
    void Display( );
};
Factorial::Factorial(int val)
{
    n=val;
    ①_____ ;
}
void Factorial::Calculate( )
{
    int i=n;
    while(i>1)
        ②_____ ;
}
void Factorial::Display( )
{
    cout<<n<<"!="<<fact<<endl;
}
void main( )
{
    int n;
    cout<<"请输入 n 的值: ";
    cin>>n;
    ③_____ ;
    A.Calculate( );
    A.Display( );
}
```

完成程序后，试算运行结果为：

请输入 n 的值：6

6!=720

参考答案：

① fact=1

② fact*=i

③ Factorial A(n)

分析 本程序中首先声明了一个类结构 Factorial，其中，包含两个数据成员 n 与 fact，它们分别用于表示整数 n 与 n!；包含一个构造函数和两个成员函数 Calculate() 与 Display()，Calculate() 用于 n! 的计算过程，Display() 用于输出 n 与 n! 的信息。

在构造函数 Factorial(int val) 中，应完成对两个数据成员 n 与 fact 的初始化赋值，显然 n 由形式参数 val 进行传递，而 fact 是用于记录 n! 的，故知①处应填 fact=1；在 n! 的计算过程函数 Calculate() 中，由于 i 的初值等于 n，而 while() 循环的控制条件为 i>1，故知 i 是一个递减的过程，因此②处应填 fact*=i--；主函数 main() 中的第 3 个空③，显然应是对对象 A 的定义与初始化，故③处应填 Factorial A(n)。

2. 以下程序采用类结构的方式，可实现对英文单词的初始化，并可从前向后地查找某给定字符串中是否存在，若存在，则输出该单词并进行统计。试填空完成该程序。

```
#include <iostream.h>
#include <string.h>
static int n=0;           //静态全局变量 n 用来记录相匹配的单词个数
class Word
{
    char words[12];        //设每个单词不超过 12 个字符
public:
    Word(char wd[ ]);
    void lookup(char s[ ]);
};
Word::Word(char wd[ ])
{
    for(int i=0;wd[i]!='\0';i++)
        _____①_____ ;
    words[i]='\0';          //赋值字符串结束标识
}
void Word::lookup(char s[ ])
{
    int j;
    for(j=0;s[j]!='\0'&&words[j]!='\0'&&s[j]==words[j];j++);
```

```

        if( _____ ② _____ )
        {
            cout<<"      " <<words<<endl;
            _____ ③ _____;
        }
    }
}

void main( )
{
    Word wobj[5]=
    { Word("elapse"),Word("embody"),Word("embroider"),Word("elude"),Word("emulate")};
    char w[12];
    cout<<"请输入欲查找的字符串: ";
    cin>>w;
    cout<<"相匹配的单词有: \n";
    for(int i=0;i<5;i++)
        _____ ④ _____;
    cout<<"匹配的单词个数为: "<< _____ ⑤ _____ <<endl;
}

```

完成程序后，运行结果为：

请输入欲查找的字符串：em

相匹配的单词有：

```

    embody
    embroider
    emulate

```

匹配的单词个数为：3

参考答案：

- ① words[i]=wd[i]
- ② s[j]=='\0'
- ③ n++ 或 n=n+1
- ④ wobj[i].lookup(w)
- ⑤ n

分析 本程序中首先定义一个静态全局变量 `n` 用来记录相匹配的单词个数；然后设计一个 `Word` 类，包含一个存放英文单词的字符型私有数据成员 `words[12]` 和一个可实现对私有数据成员 `words` 进行初始化的构造函数；再设计一个用于查找与给定字符串相匹配的所有单词并可进行统计记录的成员函数 `lookup(char s[])`。

构造函数要通过形参数组 `wd[]` 实现对数据成员 `words[12]` 的初始化，故知①处应填 `words[i]=wd[i]`；②用来判断是否找到匹配字符串，显然只有当要查找字符串 `s[j]` 在上循环中等

于'\0'作为结束条件时才表示找到,故②处应填 `s[j]=='\0'`;找到时应该给记录匹配单词个数的变量 `n` 加 1,因此③处可填 `n++`或 `n=n+1`;主函数 `main()`中的 `for(int i=0;i<5;i++)`循环,是实现对对象数组 `wobj[5]`的逐个对象元素进行与给定字符串 `w[12]`的匹配查找,因此④处应填 `wobj[i].lookup(w)`;记录匹配单词个数的是静态全局变量 `n`,显然⑤处应填 `n`。

4.3 上机实验指导

4.3.1 实验目的

- (1) 学习类的声明方法,掌握类成员函数的定义形式。
- (2) 理解对象的概念,掌握对象的定义和使用方法。
- (3) 学习构造函数和析构函数的作用及其使用方法。
- (4) 学习将数组作为函数参数的方法。
- (5) 学习类的组合,基本掌握类模板的使用方法。

4.3.2 实验内容与补充习题

1. 试分析以下程序的运行结果并上机验证。

```
#include <iostream.h>
class Sample
{
    int x;
public:
    Sample() { };
    Sample(int a) { x=a; }
    Sample(Sample &a) { x=a.x+++10; }
    void disp(char *ob)
    {
        cout<<ob<<"x="<<x<<endl;
    }
};
void main()
{
    Sample s1(2),s2(s1),s3;
    s3=s2;
    s1.disp("s1.");
    s2.disp("s2.");
    s3.disp("s3.");
}
```

运行结果为：

```
_____  
_____  
_____
```

2. 试分析以下程序的运行结果并上机验证。

```
#include <iostream.h>  
class Sample  
{  
    int A;  
    static int B;  
public:  
    Sample(int a) { A=a;B+=a; }  
    static void func(Sample s);  
};  
void Sample::func(Sample s)  
{  
    cout<<"A="<<s.A<<" , B="<<B<<endl;  
}  
int Sample::B=0;  
void main( )  
{  
    Sample s1(2),s2(5);  
    Sample::func(s1);  
    Sample::func(s2);  
}
```

运行结果为：

```
_____  
_____
```

3. 以下是一个类中包含另一个类对象成员(类的组合)的例子，试分析并给出程序的运行结果。

```
#include <iostream.h>  
class Son          //定义类 Son  
{  
    int age;  
public:  
    Son( ) { age=1; }  
    Son(int i) { age=i; }
```

```

    void print( ) { cout<<"儿子的年龄是: "<<age<<endl; }
};

class Father           //定义类 Father
{
    int age;
    Son s1,s2;         //Son 类的对象
public:
    Father(int a1,int a2,int f):s2(a2),s1(a1) //带对象成员初始化表的构造函数
    {
        age=f;
    }
    void print( )
    {
        cout<<"父亲的年龄是: "<<age<<endl;
    }
    Son &gets1( ) { cout<<"第一个"; return s1; }      //访问对象成员 s1 的函数
    Son &gets2( ) { cout<<"第二个"; return s2; }      //访问对象成员 s2 的函数
};

void main( )
{
    Father f(10,5,38);
    f.print( );
    f.gets1().print( );
    f.gets2().print( );
}

```

运行结果为:

```

_____
_____
_____

```

4. 以下是用一个外部函数对两个类对象进行比较并求出最大值的程序。该程序一方面反映了对私有成员的访问过程, 另一方面示例了对象成员作为函数参数时的形式, 也表现了类成员函数同普通函数一样, 可以拥有各种类型的形参(不一定与类的数据成员有关)。试分析并给出其运行结果。

```

#include <iostream.h>

class Student
{
    int score;
public:

```

```

Student(int s) { score=s; }
void show(char xh[])
{
    cout<<xh<<"该学生成绩是: "<<score<<endl;
}
int getscore( );           //访问成绩的成员函数
};
int Student::getscore( ) { return score; }
int max(Student &s1,Student &s2)    //定义一用于求最大成绩外部函数
{
    return (s1.getscore( )>s2.getscore( ))?s1.getscore( ):s2.getscore( );
}
void main( )
{
    Student s1(95),s2(59);
    s1.show("s1");
    s2.show("s2");
    cout<<" 较高的成绩是: "<<max(s1,s2)<<endl;
}

```

运行结果为:

```

_____
_____
_____

```

5. 定义一个类 `score`, 它含有私有数据成员 `english_score`(英语分数)、公有成员函数 `setscore()`和 `printscore()`, 其中, `setscore()`用来设置 `english_score` 的值, `printscore()`用来输出 `english_score` 的值。在主程序中定义类 `score` 的两个对象 `stu1` 和 `stu2`, 其英语成绩分别为 85.5 和 93.5, 输出这两个分数。

6. 下面是一个计算器类的定义, 请完成该类的实现并在主函数中先将计算器给定初值 99, 然后进行二次加 1, 一次减 1, 最后显示计算器的值。

```

class counter
{
    int value;
public:
    counter(int number);
    void increment( );           //给原值加 1
    void decrement( );          //给原值减 1
    int getvalue( );             //取得计数器值
}

```

```
void display( );           //显示计算器值  
}
```

7. 试定义一个字符串类 `string`，使其至少具有内容(`contents`)和长度(`length`)两个数据成员，并具有显示字符串、求字符串长度和给原字符串后添加一个字符串等功能。

8. 用类结构的形式编写一个程序，使其可实现输入 `n` 个学生的姓名和出生日期，并按年龄从小到大进行排序输出。

第5章 程序结构

5.1 基本知识点、内容概要与学习要求

5.1.1 基本知识点

1. 作用域与可见性
2. 生存期
 - 静态生存期。
 - 局部生存期。
 - 动态生存期。
3. 局部变量和全局变量
4. 静态成员
 - 静态数据成员。
 - 静态函数成员。
5. 友元
 - 友元函数。
 - 友元类。
6. 常类型
 - 常引用。
 - 常对象。
 - 常成员函数。
 - 常数据成员。
7. 多文件结构和编译预处理

5.1.2 内容概要

1. 作用域与可见性

作用域规定了程序中变量或者对象的有效范围，它给变量或者对象的可见性提供了依据。标识符的可见性是指在程序的某个位置该标识符可以被有效地引用。所有的变量都有作用域和可见性。

C++的作用域有函数原型作用域、块作用域(局部作用域)和文件作用域。

2. 生存期

变量和对象从诞生到结束的这段时间称为它的生存期。在生存期内，对象将保持它的状态，即数据成员的值，若变量不被更新，将保持它的值不变。

生存期与存储区域关系密切，一般存储区域分为代码区(code area)、数据区(data area)、栈区(stack area)和堆区(heap area)。代码区用来存放程序代码，与其他存储区相对应的生存期分别为静态生存期、局部生存期和动态生存期。

1) 静态生存期

静态生存期与程序的运行期相同。如果要在函数内部的块作用域中声明具有静态生存期的变量，则要用关键字 `static` 说明。具有静态生存期的变量在固定的数据区域分配空间。如果具有静态生存期的变量未初始化，则自动初始化为 0。

全局变量、静态全局变量和静态局部变量都具有静态生存期。

2) 局部生存期

在块作用域中声明的变量具有局部生存期，此生存期诞生于声明点，终止于其作用域结束处。具有局部生存期的变量在内存的栈区分配空间。具有局部生存期的变量在初始化之前，内容不确定。

3) 动态生存期

动态生存期由程序中特定的函数(`malloc()`和`free()`)或由操作符(`new`和`delete`)创建和释放。具有动态生存期的变量在内存的堆区分配空间。

3. 局部变量和全局变量

1) 局部变量

局部变量具有局部作用域。因此，在不同函数体内的局部变量是互相不可见的，这就很好地实现了函数之间的数据隐蔽。局部变量包括自动(auto)变量、内部静态(static)变量和函数参数。程序中没有特别说明的变量都是自动变量。自动变量以堆栈方式占用内存空间。

2) 全局变量

全局变量具有文件作用域。在整个程序中，除了在定义有同名局部变量的块中之外，其他地方都可以直接访问全局变量。将数据存放在全局变量中，不同的函数可在不同的地方对同一个全局变量进行访问，以实现这些函数之间的数据共享。

4. 静态成员

静态成员为类内部封装的数据成员提供了在同一个类的函数成员之间进行数据共享的机制，这样的共享使数据的访问范围得到了严格的控制。使用静态成员可以在同一个类的不同对象之间共享数据。在类中，静态成员分为静态数据成员和静态函数成员。

1) 静态数据成员

静态数据成员是类的数据成员的一种特例，每个类只有一个拷贝，它由该类的所有对象共同维护和使用，以实现同一个类的不同对象之间的数据共享。静态数据成员具有静态生存期。

2) 静态函数成员

静态函数成员是使用 `static` 关键字声明的函数成员。同静态数据成员一样，静态函数成员也属于整个类，由同一个类的所有对象共同维护，为这些对象所共享。静态函数成员可以直接引用该类的静态数据和函数成员，但不能直接引用非静态数据成员，如果要引用，必须通过参数传递的方式得到对象名，然后再通过对象名来引用。

5. 友元

类的友元可以访问该类的所有数据成员，友元可以是普通函数、其他类的成员函数，也可以是其他类。友元在类之间、类与普通函数之间共享了内部封装的数据，但对类的封装性有一定的破坏。

1) 友元函数

如果友元是普通函数或其他类的成员函数，则称为友元函数。友元函数是在类声明中由关键字 `friend` 修饰的非成员函数。

普通函数声明为友元函数的格式如下：

```
friend <类型> <友元函数名>(<参数表>);
```

成员函数声明为友元函数的格式如下：

```
friend <类型> <类名>::<友元函数名>(<参数表>);
```

2) 友元类

如果友元是一个类，则称为友元类。友元类的所有成员函数都是友元函数。友元类的声明格式为

```
friend class <友元类名>;
```

6. 常类型

常类型是指使用 `const` 声明的类型，变量或对象被声明为常类型后，其值就不能被更新，因此，声明常类型时必须要进行初始化。虽然数据隐藏保证了数据的安全性，但各种形式的数据共享却又不同程度地破坏了数据的安全性，因此，对于既需要共享又需要防止改变的数据应该声明为常量以进行保护。

1) 常引用

常引用的声明格式为

```
const <类型说明符> &<引用名>
```

2) 常对象

常对象的声明格式为：

```
const <类名> <对象名>
```

或

```
<类名> const <对象名>
```

3) 常成员函数

常成员函数声明的格式为

```
<类型说明符> <函数名> (<参数表>) const;
```

4) 常数据成员

使用关键字 `const` 不仅可以说明成员函数, 还可以说明数据成员。如果在一个类中说明了常数据成员(包括常引用、常对象等), 由于常数据成员不能被更新, 因此, 只能用成员初始化列表的方式通过构造函数对该数据成员进行初始化。

7. 多文件结构和编译预处理

一个 C++ 源程序至少应该由类的声明(.h, 类的声明部分)、类的实现(.cpp, 类的成员函数体)和类的使用(.cpp, 主函数)三个部分组成。对于有多个类的程序, 每个类的声明和实现最好单独存放在一个文件中。这些文件相互之间用包含指令(include)联系在一起, 统一管理, 从而提高了程序编写、编译和连接的效率。

5.1.3 学习要求

- 理解作用域、可见性的重要意义。
- 了解各种生存期的异同之处。
- 了解静态成员在数据共享中的作用。
- 掌握友元的定义和使用方法。
- 了解常类型在数据共享中的保护作用。
- 了解多文件的组织结构和常用的编译预处理命令。

5.2 典型例题 (或典型算法) 分析

5.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

1. 程序代码如下:

```
#include <iostream.h>
int m;
void main( )
{
    void fun1( );
    int m=10;
    int n=0;
    for(int i=0;i<10;i++)
    {
        if(i%2)
            n++;
    }
    m=n/2;
```

```

    cout<<"在 main( )内的 m: "<<m<<endl;
    fun1( );
    n=i;
    cout<<"n="<<n<<endl;
}
void fun1( )
{
    cout<<"在 main( )外的 m: "<<m<<endl;
}

```

运行结果为:

```

在 main( )内的 m: 2
在 main( )外的 m: 0
n=10

```

分析 这是一个作用域及可见性的程序。在这个例子中，主函数 `main()` 内定义的都是局部变量，它们都具有块作用域。函数体 `main()` 是一个块，`for` 语句之后的循环体又是一个较小的块，变量 `n` 和 `i` 的作用域从声明处开始，到它所在的块，即整个函数体 `main()` 结束处为止。主函数之前声明的变量 `m` 具有文件作用域，它的有效作用范围是整个源代码文件；主函数内声明的变量 `m` 具有块作用域，它的作用范围在内层的花括号内，`m` 的块作用域被完全包含在 `m` 的文件作用域中。根据作用域及可见性的规则，在具有包含关系的作用域中声明同名标识符，外层标识符在内层不可见，程序的运行结果验证了这一点。

2. 程序代码如下:

```

#include<iostream.h>
class Test
{
private:
    int k;
public:
    static int n;
    Test(int kk){k=kk;n++;}
    static void Display( )
    {cout<<"n="<<n<<endl;}
};
int Test::n=0;
void main( )
{
    Test::Display( );
    Test t1(10);
}

```

```
t1.Display( );
Test t2(20);
t2.Display( );
}
```

运行结果为：

```
n=0
n=1
n=2
```

分析 这是一个有关静态成员的程序。类 `Test` 中，声明 `n` 为公有静态数据成员，用来给 `Test` 类的对象计数，每声明一个新对象，`n` 的值就相应加 1，同时，声明 `Display()` 为静态函数成员，用它来输出静态数据成员 `n` 的值。静态数据成员 `n` 的定义和初始化在类外进行，`n` 的值在类的构造函数中被引用。当 `t1` 对象生成时，调用构造函数 `Test()`，`n` 值由 0 变为 1；当 `t2` 对象生成时，又调用构造函数 `Test()`，这次 `n` 值由 1 变为 2。两次调用均访问的是 `t1` 和 `t2` 共同维护的该静态成员的拷贝，这样实现了在 `t1` 和 `t2` 两个对象间数据共享。这里特别要注意主函数的第一条语句“`Test::Display()`”，因为此时还没有任何对象生成，所以只能用类名调用 `Display()`，这正是静态成员有别于一般成员的地方。

3. 程序代码如下：

```
#include<iostream.h>
#include<string.h>
class student
{
private:
    char name[10];
    int age;
public:
    student(char in_name[ ],int in_age)
    {
        strcpy(name,in_name);
        age=in_age;
    }
    int get_age( ){return age;}
    char *get_name( ){return name;}
    friend int compare(student &s1,student &s2)
    {
        if(s1.age>s2.age)
            return 1;
        else if(s1.age==s2.age)
```

```

        return 0;
    else return -1;
}
};

void main( )
{
    student stu[] = { student("王红",18), student("吴伟",19), student("李丽",17) };
    int i,min=0,max=0;
    for(i=1;i<3;i++)
    {
        if(compare(stu[max],stu[i])==-1)
            max=i;
        else if(compare(stu[max],stu[i])==1)
            min=i;
    }
    cout<<"最大年龄: "<<stu[max].get_age()<<" , 姓名: "<<stu[max].get_name()<<endl;
    cout<<"最小年龄: "<<stu[min].get_age()<<" , 姓名: "<<stu[min].get_name()<<endl;
}

```

运行结果为:

最大年龄: 19, 姓名: 吴伟

最小年龄: 17, 姓名: 李丽

分析 这是一般函数作友元的程序。student 类中声明两个私有数据成员姓名(name[])和年龄(age), 其中, name 是字符数组, 用来接受学生的姓名。这里除定义得到年龄(get_age())和得到姓名(get_name())的函数外, 还定义了一个 compare()函数为友元, 用来比较学生的年龄。当第一个学生年龄大于第二个学生年龄时, 该函数返回值为 1, 当第一个学生年龄等于第二个学生年龄时, 该函数返回值为 0, 否则函数返回值为-1。

5.2.2 程序设计与算法分析

1. 用 C++面向对象的程序设计方法, 找出十个学生中成绩最高者(学生信息包括学号、姓名和成绩)。

分析 首先根据题目要求定义一个包含表示学生信息的数据成员 No(学号)、name[] (姓名)和 score(成绩)的 Sample 类, 一个用于给数据成员赋值的成员函数 setx()和显示学生信息的成员函数 disp(), 另外还应声明一个用于求成绩最高者学号的友元函数 fun()。程序代码如下:

```

#include <iostream.h>
#include <string.h>

```

```

class Sample
{
    int No;
    char name[10];
    int score;
public:
    Sample( ) { };
    void setx(int i,char na[],int sc)
    {
        No=i;
        strcpy(name,na);
        score=sc;
    }
    friend int fun(Sample B[],int n);
    void disp( )
    {
        cout<<"学号: "<<No<<" 姓名: "<<name<<" 成绩: "<<score<<endl;
    }
};

int fun(Sample B[],int n)
{
    int max=0,k=0;
    for(int i=0;i<n;i++)
        if(B[i].score>max) {max=B[i].score; k=i; }
    return k;
}

void main( )
{
    Sample A[10];
    int max;
    char *Arr[10]={"Li","Wang","Zhao","Feng","Ma","Gao","Liou","Hua","Qian","Dong"};
    int Scr[]={90,83,21,78,56,60,96,10,39,72};
    for(int i=0;i<10;i++)
        A[i].setx(i+1,Arr[i],Scr[i]);
    max=fun(A,10);
    cout<<"成绩最高者为: ";
    A[max].disp( );
}

```


运行结果为:

成绩最高者为: 学号: 7 姓名: Liou 成绩: 96

在友元函数“int fun(Sample B[],int n)”的定义中,形参“Sample B[]”用于接收表示若干学生信息的对象数组,n用于接收学生的人数,函数的返回值k表示成绩最大的学号。

在主函数 main()中,对于字符型指针数组“char *Arr[10]”的定义请参见第6章的相关内容,在此略过。现主要讨论对对象数组A[10]赋值的循环语句“for(int i=0;i<10;i++) A[i].setx(i+1,Arr[i],Scr[i]);”,显然其中的 i+1、Arr[i]和 Scr[i]分别作为对象 A[i]的成员函数 setx()的实参,将 i+1 作为学号、字符数组元素 Arr[i]作为学生姓名、整型数组元素 Scr[i]作为学生成绩分别赋给了第 i 个(注意: i 的变化范围是 0~9)学生对象 A[i]的数据成员 No、name[]和 score,然后通过调用函数 fun(A,10)将成绩最高者的学号返回赋给变量 max,最后由 A[max].disp()显示学号为 max 的相应信息。

2. 在编制 windows 程序时经常将类的声明部分、成员函数定义部分和其他程序部分分别存入不同的文件,一般的做法是:首先将类声明部分存入一个头文件(.h 文件),作为该类的对外接口,将类中的成员函数定义部分存入一个源文件(.cpp 文件),将其他程序部分存入另一个源文件(.cpp 文件,其中包含 main()函数);然后在每一个源文件中需要将类声明的头文件包含进来;最后分别进行编译后连接运行。

例如,包含三个文件形式的简单商品管理应用程序创建过程如下:

1) 头文件 Commodity.h

```
class Commodity
{
    int No;                //商品号
    float price;           //单价
    long number;           //库存量
public:
    void set(int n,float p,long num); //设置数据函数声明
    void print();             //显示函数声明
    void in(int m);           //进库函数声明
    void out(int m);          //出库函数声明
};
```

2) 类成员函数实现源文件 Commodity.cpp

```
#include <iostream.h>
#include "Commodity.h"
void Commodity::set(int n,float p,long num)
{
    No=n;
    price=p;
    number=num;
```

```
}  
void Commodity::print( )  
{  
    cout<<"商品号:"<<No<<" , 单价:"<<price<<" , 库存量:"<<number<<endl;  
}  
void Commodity::in(int m) { number+=m; }  
void Commodity::out(int m) { number-=m; }
```

3) 含 main()函数的源文件 main.cpp

```
#include <iostream.h>  
#include "Commodity.h"  
void main( )  
{  
    Commodity goods1,goods2;  
    goods1.set(101,15.6,100); //设置商品 goods1 的初始数据  
    goods2.set(103,18.9,20); //设置商品 goods2 的初始数据  
    goods1.in(50);  
    goods2.in(10);  
    goods1.out(30);  
    goods1.print( );  
    goods2.print( ),  
}
```

运行结果为:

商品号: 101 , 单价: 15.6 , 库存量: 120

商品号: 103 , 单价: 18.9 , 库存量: 30

5.2.3 填空

1. 下列程序是用递归的方法计算 $\text{fun}(5)*10$ 的值。其中 $\text{fun}(k)=\text{fun}(k-1)*k$, 当 $k=0$ 时, $\text{fun}(0)=3$ 。请根据运行结果填空完成程序, 并上机验证。

```
#include<iostream.h>  
int n=3;  
int fun(int);  
void main( )  
{  
    _____ ① _____;  
    cout<<fun(5)*n<<endl;  
}
```

```
int fun( ② )
{
    if(k==0)
        ③ ;
    return fun(k-1)*k;
}
```

运行结果为:

3600

参考答案:

- ① int n=10
- ② int k
- ③ return n

分析 这是一个递归的程序,其中用到了有关作用域的知识。在主函数 main() 的外面定义的 n 为全局变量,具有文件作用域。因为要求程序运行结果为 3600,所以第一个空要重新定义一个局部变量 n,并赋值为 10(当然,这不是惟一答案)。第二个空是函数参数,应填入 int k。第三个空是当递归终止条件满足时,返回 n 值。

2. 下列程序是有关友元类的。程序中将 A 类声明为 B 类的友元类, A 类中所有的成员函数都是 B 类的友元函数。请填空完成程序并上机运行验证。

```
#include<iostream.h>
class B;
class A
{
private:
    int x;
public:
    A(int xx) {x=xx;}
    int Set(B &);
    int Get() {return x;}
};
class B
{
private:
    int x;
public:
    B(int xx) { ① ; }
    friend ② ;
};
```

```

int A::Set(____③____)
{
    return x=b.x;
}
void main( )
{
    A a(10);
    B b(20);
    cout<<a.Get()<<" ";
    a.Set(b);
    cout<<a.Get()<<endl;
}

```

运行结果为:

10, 20

参考答案:

- ① $x=x$
- ② `class A`
- ③ `B &b`

分析 这是一个有关友元类的程序, 程序比较简单, 第一个空是 B 类构造函数的实现, 因此应填入 $x=x$; 第二个空是友元类的声明形式, 这一点要特别注意; 第三个空是定义 A 类中 Set() 函数的形参, 要写完整, 故需填入 `B &b`。

3. 设计一个直线类 Line (设直线方程为 $ax + bx + c = 0$), 其中, 包含三个数据成员 a、b、c, 一个显示数据成员的 disp 成员函数和一个求两直线交点的友元函数 setpoint, 并在 main() 中给出对象或数据进行测试。请填空完成程序并上机运行验证。

```

#include <iostream.h>
#include <math.h>
class point
{
    double x,y;
    public:
    void setvalue(____①____)
    { x=x1;y=y1; }
    void disp( )
    { cout<<"("<<x<<" "<<y<<"")<<endl; }
};
class line
{

```

```

int a,b,c;
public:
    line(int a1,int b1,int c1)
    { a=a1;b=b1;c=c1; }
    _____②_____ ;
    void disp( )
    { cout<<a<<"x+"<<b<<"y+"<<c<<"=0"<<endl;}
};
point secpoint(line l1,line l2)
{
    point p;
    _____③_____ ;
    x=(l1.b*l2.c-l2.b*l1.c)/(l1.a*l2.b-l2.a*l1.b);
    y=(l1.c*l2.a-l2.c*l1.a)/(l1.a*l2.b-l2.a*l1.b);
    p.setvalue(x,y);
    _____④_____ ;
};
void main( )
{
    point p;
    line a(2,3,-1).b(-1,4,6);
    cout<<"直线 L1 为: "; a.disp( );
    cout<<"直线 L2 为: "; b.disp( );
    p= _____⑤_____ ;
    cout<<"交 点 为: ";
    p.disp( );
}

```

运行结果为:

直线 L1 为: $2x+3y-1=0$

直线 L2 为: $-1x+4y+6=0$

交 点 为: (2, -1)

参考答案:

- ① double x1,double y1
- ② friend point secpoint(line l1,line l2)
- ③ double x,y
- ④ return p
- ⑤ secpoint(a,b)

分析 这是一个有关友元函数的应用程序,类 point 定义中的空①是成员函数 setvalue() 的参数表位置,根据数据成员 x、y 的类型,显然应填 double x1,double y1。注意到外部函数“point secpoint(line l1,line l2)”的返回类型是 point 类类型,形参是两个 line 类对象,因此此函数应是 line 类的友元函数,故空②处应填 friend point secpoint(line l1,line l2),而在此外部函数定义中的第③、④两空,根据上下语句的对应关系,发现变量 x、y 没有定义且无返回语句,故应分别填入 double x,y 和 return p。在主函数 main() 中的第⑤空,应计算两已知直线 a 与 b 的交点,因此填入函数调用 secpoint(a,b)。

5.3 上机实验指导

5.3.1 实验目的

- (1) 学习作用域、可见性和生存期的概念。
- (2) 学习如何使用静态成员。
- (3) 学习友元的定义和使用方法。

5.3.2 实验内容与补充习题

1. 下列是关于作用域、可见性和生存期的程序,程序中有错,试分析程序并改正。

```
#include<iostream.h>
void main( )
{
    void fun1( );
    int n;
    for(int i=0;i<5;i++)
    {
        int m;
        if(i%2)
            n++;
    }
    m=n/2;
    fun1( );
    n=i;
}
void fun1( )
{
    cout<<"i="<<i<<endl;
}
```

2. 分析并写出下列程序的运行结果，然后上机运行验证。

```
#include<iostream.h>
class Point
{
private:
    int X,Y;
    static int countP;
public:
    Point(int xx=0,int yy=0)
    {X=xx;Y=yy;countP++;}
    Point(Point &p);
    int GetX( ) {return X;}
    int GetY( ) {return Y;}
    static void GetC( )
    {cout<<" Object id="<<countP<<endl;}
};
Point::Point(Point &p)
{
    X=p.X;
    Y=p.Y;
    countP++;
}
int Point::countP=0;
void main( )
{
    Point::GetC( );
    Point A(4,5);
    cout<<"Point A,"<<A.GetX( )<<","<<A.GetY( );
    A.GetC( );
    Point B(A);
    cout<<"Point B,"<<B.GetX( )<<","<<B.GetY( );
    Point::GetC( );
}
```

运行结果为：

3. 下列是关于常引用的程序, 程序中有错, 试分析程序并改正。

```
#include<iostream.h>
void main( )
{
    int n=5;
    const int &m;
    m=10;
    cout<<"m="<<m<<endl;
}
```

4. 下列是用多文件结构的形式实现用友元函数计算两点之间距离的程序。分析程序的结构, 并上机运行验证。

文件 1, 头文件 point.h (类的声明)

```
#include<iostream.h>
#include<math.h>
class Point
{
private:
    double X,Y;
public:
    Point(double xx=0,double yy=0) {X=xx;Y=yy;}
    double GetX( ) {return X;}
    double GetY( ) {return Y;}
    friend double Dist(Point &p1,Point &p2);
},
```

文件 2, 友元函数的实现源文件 point.cpp

```
#include"point.h"
double Dist(Point &p1,Point &p2)
{
    return (sqrt((p1.X-p2.X)*(p1.X-p2.X)+(p1.Y-p2.Y)*(p1.Y-p2.Y)));
}
```

文件 3, 含主函数 main()的源文件 fmian.cpp

```
#include<iostream.h>
#include"point.h"
void main( )
{
    Point myp1(1,1),myp2(4,5);
```



```

        cout<<"The distance is:"<<Dist(myp1,myp2)<<endl;
    }

```

5. 首先声明一个 Cat 类,使其拥有记录 Cat 的个体数目的静态数据成员 HowManyCatst 和存取 HowManyCats 的静态成员函数 GetHowMany(); 然后设计程序测试这个类,体会静态数据成员和静态成员函数的用法。

6. 编写一个程序输入几个学生的姓名、英语和计算机成绩,然后按总分从高到低排序。(要求定义一个 student 类,用友元实现排序。)

7. 下列程序实现的是堆栈的压入和弹出。其中有两个类,一个是结点类,它包含结点值和指向上一个结点的指针;另一个类是堆栈类,数据成员为堆栈的头指针,它是结点类的友元。试分析程序,并说明堆栈的压入和弹出过程。

```

#include<iostream.h>
class stack;
class node
{
    int data;
    node *prev;
public:
    node(int d,node *n)
    {
        data=d;
        prev=n;
    }
    friend class stack;
};
class stack
{
    node * top;
public:
    stack( ){top=0;}
    void push(int i);
    int pop( );
};
void stack::push(int i)
{
    node *n=new node(i,top);
    top=n;
}
int stack::pop( )

```

```
{
    node *t=top;
    if(top)
    {
        top=top->prev;
        int c=t->data;
        delete t;
        return c;
    }
    return 0;
}

main( )
{
    int c;
    stack s;
    for(int j=0;j<10;j++)
    {
        cin>>c;
        s.push(c);
    }
    for(j=0;j<10;j++)
        cout<<s.pop( )<<" ";
    cout<<"\n";
    return 1;
}
```

*8. 编写一个队列操作类程序，然后用友元实现入队和出队。

第 6 章 指 针

6.1 基本知识点、内容概要与学习要求

6.1.1 基本知识点

1. 指针的概念

- 指针与内存地址的关系。
- 指针型变量的声明。
- 指针的基本操作。
- 指针变量的初始化。
- 指针变量的引用。

2. 指针的运算

- 指针的赋值运算。
- 指针的算术运算。
- 指针的关系运算。
- 常指针(const 修饰符与指针)。
- 引用型变量。

3. 指针与数组的关系

- 通过指针整体引用数组。
- 通过指针引用数组元素。
- 指向多维数组的指针。
- 指针数组与多级指针。

4. 指针与函数的关系

- 作为函数参数的指针。
- 函数的指针与指向函数的指针。
- 函数返回指针。

5. 指针与类、对象

- 类的指针。
- 对象的指针。
- this 指针。

6. 指针与字符串

- 字符串指针。
- 字符串标准库函数的使用。
- string 类。

7. 动态内存分配与 new 和 delete 运算符

- new 运算符。
- delete 运算符。
- 动态内存分配应用举例——链表。

6.1.2 内容概要

1. 指针的概念

指针就是指向内存中某个单元的地址值，与其对应的数据类型有关。

2. 指针的定义

指针变量的定义格式如下：

数据类型 *指针变量名；

其中，数据类型是指针变量的“基类型”，它规定了指针变量所能指向的数据变量类型。

一个指针变量只能按定义指向所规定的一种数据类型的变量，不能定义一个指针变量，一会指向整型变量，一会又指向双精度变量。

3. 指针的基本运算符

指针变量的基本运算符有以下两个：

(1) 取地址运算符“&”，其使用格式为

&变量名

它返回变量的首地址，也可以返回指针变量的首地址。

(2) 指向运算符(取指针所指的内容)“*”，其使用格式为

*指针变量名

它返回指针变量所指向的值。

&和*运算符都是单目运算符，其优先级高于所有双目运算符。

4. 指针变量的初始化

指针变量在定义中允许被初始化。若指针变量在定义中不带初始化项，则指针变量被初始化为 NULL，它的值为 0，此时指针不指向任何有效数据，也称其为空指针。因此，当调用一个要返回指针的函数时，常用返回值 NULL 来判断函数调用发生了某些错误。

5. 指针的运算

(1) 当指针变量 p 指向某一连续存储区中的某个存储单元时，可以通过+、-某个常量或++、--运算来移动指针。

(2) 一般情况下，当两个指针指向同一个数组时，可在关系表达式中对两个指针进行比较。若 $p==q$ 为真，则表示 p、q 指向数组的同一元素；若 $p<q$ 为真，则表示 p 所指向的数组元素在 q 所指向的数组元素之前。

(3) 当两个指针指向同一个数组时,可进行减法运算,相减的绝对值除以基类型长度表示两个指针所指对象之间的元素个数。

注意:不论指针变量指向何种数据类型,指针和整数 n 进行加、减运算时,实际加减的长度都应为 n 乘以基类型长度。

(4) 常指针(用 `const` 修饰符修饰的指针)。

(5) 引用型变量。

6. 指针与数组的关系

任何能由数组下标完成的操作都可用指针来实现,在程序中使用指针可使代码更紧凑、更灵活。

数组名本身就是该数组的指针,即该数组在内存中存储的开始地址(一般为连续的存储单元),因此,可以将数组名赋予指针变量,反过来,也可以把指针看成一个数组。

例如,若有定义 `int a[10], *p=a;`,则用指针 p 给出数组元素的地址和内容的几种形式为:

(1) $p+i$ 和 $a+i$ 均表示 $a[i]$ 的地址,或者说它们均指向数组第 i 号元素,即指向 $a[i]$ 。

(2) $*(p+i)$ 和 $*(a+i)$ 都表示 $p+i$ 和 $a+i$ 所指数组元素的内容,即为 $a[i]$ 。

(3) 指向数组元素的指针,也可以表示成数组的形式。也就是说,它允许指针变量带下标,如 $p[i]$ 与 $*(p+i)$ 等价。

二维数组元素 $a[i][j]$ 可表示成 $*(a[i]+j)$ 或 $*((a+i)+j)$,它们都与 $a[i][j]$ 等价,也可写成 $*(a+i)[j]$ 。

注意:指针是变量,可以作运算,而数组名是常量,不能进行运算;程序自动为数组分配所需的存储空间,而指针则是利用系统的动态分配功能为它分配存储空间或赋给它一个已分配的空间地址。

7. 指针数组与多级指针

1) 指针数组

用指向同一数据类型的指针构成的一个数组,就是指针数组。如

```
int*a[10];
```

其中, a 是一个包含 10 个元素的数组,每个元素是指向一个整型数的指针。

指针数组使对字符串的处理更加方便和灵活,使用二维数组对各行长度不等的文档的处理效率较低,而指针数组由于其中每个元素都为指针变量,因此通过地址运算来操作文档各行是十分方便的。

2) 多级指针

可以定义指向指针的指针变量,即二级指针。如

```
int **prd;
```

其中, prd 是一个指向指针的指针,被指向的指针指向一个整型数。

8. 指针与函数

1) 作为函数参数的指针

通过指针型参数,可以将一个变量的地址传递给函数,而通过这个地址,函数体中的语句就可以修改函数以外的变量的值。

2) 函数的指针与指向函数的指针

在 C++ 语言中, 指针变量除了可以保存数据的存储地址外, 还可以用于保存函数的存储首地址。函数的存储首地址又称为函数的执行入口地址。指针变量保存函数的入口地址时, 它就指向了该函数, 因此称这种指针为指向函数的指针, 简称为函数指针, 其说明形式如下:

数据类型 (*函数指针名)();

其中, 数据类型是指针所指向的函数的返回值的类型。函数指针名两边的圆括号不能省略, 它表示函数指针名先与 * 结合, 是指针变量, 然后再与后面的 () 结合, 表示该指针变量指向函数。例如

int (*funp)();

表示说明的函数指针 funp 是指向 int 型函数的。

C++ 语言程序中, 函数指针的作用主要体现在函数间传递函数上, 这种传递不是传递一般数据, 而是传递函数的入口执行地址, 或者说是传递函数的调用控制。当函数在两个函数间传递时, 调用函数的实参应该是被传递函数的函数名, 而被调用函数的形参应该是接收函数地址的函数指针。

3) 函数返回指针

当函数的返回值是地址时, 称为指针型函数。因为该类型函数的返回值是随返回参数的变化而变化的地址量, 而变化的地址值就是指针变量, 故称为指针型函数, 其说明的一般格式如下:

数据类型 *函数名(形参说明)

```
{  
    函数体;  
}
```

在指针型函数中, 使用 return 语句返回的可以是变量地址、数组的首地址或指针变量等指针类型。

9. 指针与类、对象

(1) 类的指针变量是一个用于保存该类对象在内存中存储空间首地址的指针型变量。

(2) 对象的指针指的是一个对象在内存中的首地址。

(3) this 指针是每个对象中隐藏的指针。this 指针是默认的, 当一个对象生成后, 这个对象的 this 指针就指向内存中保存该对象数据的存储空间的首地址。

10. 指针与字符串

字符串是以字符数组的形式进行存储与处理的, 而字符数组名就保存着字符串在内存的起始地址, 因此, 字符串实质上与 char* 型指针相对应。

(1) char* 型指针变量可以在定义时进行初始化。

(2) char* 型指针变量(或函数参数)既可以接收字符串常量, 也可以接收字符型数组(名)。

(3) 在使用字符串标准函数库中的函数时, 应在应用程序的开头添加包含 “string.h” 头文件的预处理命令 “#include <string.h>”。

(4) 先用 string 类将字符串定义为对象, 然后利用 string 类提供的字符串操作功能, 即

可实现对字符串的各种处理。与字符数组和字符指针处理字符串所不同的是, string 不一定要用 '\0' 来标识字符串的结束。

11. 动态内存分配与 new 和 delete 运算符

(1) new 运算符用于动态分配一块内存空间。其使用格式是

指针变量=new 数据类型[长度];

其中, 数据类型可以是 C++语言的标准数据类型, 也可以是结构体类型、共用体类型和类类型等, 长度表示该内存空间可以容纳该数据类型的数据个数; new 运算符返回一个指针, 指向所分配的存储空间的第 1 个单元。

注意: 如果当前存储器无足够的内存空间可分配, 则 new 运算符返回 0 (NULL)。

(2) 用 new 运算符获得的空间, 在使用完后要使用 delete 释放。delete 运算符的使用有以下两种格式:

delete 指针;

delete []指针;

其中, 不带中括号的 delete 运算符用于释放空间长度为 1 个单位的内存空间, 而带中括号的 delete 运算符用于释放空间长度大于 1 的内存空间。

6.1.3 学习要求

- 理解指针与内存地址的关系及利用指针进行“间接访问”的重要意义。
- 掌握指针型变量的声明和基本用法。
- 掌握指针的赋值运算、算术运算及比较运算。
- 了解常指针的作用与引用型变量的概念。
- 理解指针与数组的关系及利用指针进行数组元素访问的方法。
- 区别指针作为函数参数与一般变量作为函数参数的异同之处, 了解函数的指针与指向函数的指针的概念与用法。
- 了解指针与类、对象的关系和 this 指针的特点。
- 理解指针与字符串及字符数组之间的关系, 掌握一般字符串处理库函数的用法, 了解 string 类的特点与作用。
- 了解动态内存分配的特点, 掌握 new 和 delete 运算符的基本用法, 学会对简单链表的建立、插入及删除操作。

6.2 典型例题 (或典型算法) 分析

6.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include <iostream.h>
   void main( )
```

```
{
    int a[10];
    int i=5,*ptr=a;
    *(ptr+i)=10;
    cout<<a[i]<<endl;
}
```

运行结果为:

10

分析 此程序比较简单,首先是理解语句“int i=5,*ptr=a;”,即使 i 的值为 5、指针变量 ptr 指向数组 a,关键是理解*(ptr+i)等价于 a[i]。

2. #include <iostream.h>

```
void main( )
{
    void fun(int*,int*);
    int a=5,b=8;
    while(a!=b)
    {
        fun(&a,&b);
        cout<<a<<'\t'<<b<<endl;
    }
}

void fun(int *pa,int *pb)
{
    if(*pa>=*pb)
        *pa-=*pb;
    else
        *pb-=*pa;
}
```

运行结果为:

```
5    3
2    3
2    1
1    1
```

分析 本程序的特点是在调用函数 fun() 时用指针作为函数参数,其中形参是整型指针变量 pa 和 pb。在函数调用时形参用以接受由实参传来的整型变量 a 和 b 的地址(&a,&b)。这样实参和形参就指向相同的内存单元,在 fun() 内对 pa 和 pb 所指内容的改变,就意味着

对实参所指内容(即 a 和 b 值)的改变。在 while 循环中, 变量 a 和 b 的变化情况如表 6-1 所示, 在 a 与 b 的值相等时循环结束。

表 6-1 循环中 a 和 b 的变化情况

	a(即*pa)	b(即*pb)
初值	5	8
第 1 次循环后	5	3
第 2 次循环后	2	3
第 3 次循环后	2	1
第 4 次循环后	1	1

```

3. #include <iostream.h>
   void display(int*,int);
   void main( )
   {
       int a[10];
       for(int i=0;i<10;i++)
           a[i]=i*i;
       display(a,6);
   }
   void display(int *p,int n)
   {
       for(int i=n;i>0;i--)
           cout<<*(p+i-1)<<" ";
       cout<<endl;
   }

```

运行结果为:

25 16 9 4 0

分析 此程序的 display() 函数的参数中有一个指针类型, 其功能是根据 i 的变化显示指针所指的数据值*(p+i-1)。注意 i 从 n 到 0 递减变化。

在主函数 main() 中, 首先给数组 a 的各元素赋以其下标的平方值, 然后调用函数 display(a,6) 输出数组各元素值。注意理解实参数组名 a 与指针型形参 p 的对应关系。

```

4. #include <iostream.h>
   #include <string.h>
   char *stradd(char*.char*,int);
   char *pt; //用于保存连接后字符串的指针
   void main( )

```

```

{
    static char s[80]="1234",t[80]="abcd";
    stradd(s,t,0);
    cout<<pt<<endl;
    stradd(s,t,1);           //注意此时字符串 t 已发生变化
    cout<<pt<<endl;
}
char *stradd(char *s,char *t,int flag)
{
    char *ptr,*p;
    if(flag)
    {
        //t 连接到 s
        pt=s;                //保存连接后字符串的指针
        ptr=s+strlen(s);     //将 s 字符串的末尾元素指针计算出并赋给 ptr
        p=t;                 //记录被连接字符串的开始指针
    }
    else
    {
        //s 连接到 t, 其他注释与上相同
        pt=t;
        ptr=t+strlen(t);
        p=s;
    }
    for(;*p!='\0';p++,ptr++)
        *ptr=*p;
    *ptr='\0';               //给字符串加结束标记
    return pt;
}

```

运行结果为:

abcd1234

1234abcd1234

分析 此程序中函数 `*stradd(char *s,char *t,int flag)` 的作用是实现两个字符串的连接, 且返回连接后的字符串指针, 由参数 `flag` 控制 `s` 连接 `t`, 还是控制 `t` 连接 `s`, 关键是理解进行连接的循环语句 “`for(;*p!='\0';p++,ptr++) *ptr=*p;`” 的作用, 其中, 指针 `*ptr` 指向连接字符串的位置, 指针 `p` 指向被连接字符串的位置。

注意: 将指针变量 `pt` 定义为全局变量的目的, 一方面是为了防止函数调用结束后各局部变量被释放造成的错误, 另一方面是为了加快结果数据的传递速度。

```

5. #include <iostream.h>
   int a[] = {1,3,5,7,9};
   void main( )
   {
       int &sub(int);
       for(int i=0;i<4;i++)
           sub(i)*=sub(i+1);
       for(i=0;i<5;i++)
           cout<< a[i]<<" ";
       cout<<endl;
   }
   int &sub(int i)
   {
       return a[i];
   }

```

运行结果为:

3 15 35 63 9

分析 本题程序中的 sub() 是一个返回值是引用的函数, 其返回值是全局数组元素 a[i] 的引用。

这里我们可以将 sub(i) 看作 a[i] 的别名, 将 sub(i+1) 看作 a[i+1] 的别名, 表达式 sub(i)*=sub(i+1) 完成 a[i]=a[i]*a[i+1] 的功能。这里要注意的是, 函数返回值所引用的实体目标必须在函数返回后依然存在, 这里将 a 定义为全局数组就是因为全局数组的生命期是整个程序的运行期间。

```

6. #include <iostream.h>
   class A
   {
       int x;
   public:
       int y;
       void getx(int i) { x=i; }
       void gety(int i) { y=i; }
       void disp( ) { cout<<"x="<<x<<" , y="<<y<<endl; }
   };
   void main( )
   {
       A c,*pa;           //定义 pa 为指向 A 类的指针变量
       int A::*pt=&A::y;
       pa=&c;

```

```

    pa->getx(10);
    c.*pt=20;
    pa->disp( );
    void (A::*pfun)(int)=A::getx;
    (c.*pfun)(30);
    pfun=A::gety;
    (c.*pfun)(40);
    pa->disp( );
}

```

运行结果为:

x=10, y=20

x=30, y=40

分析 本题程序先定义了一个类 A, 在 main() 函数中的 “pa=&c;” 语句使指针 pa 指向 A 类对象 c, 则 “pa->getx(10);” 等价于 “c.getx(10);”。而语句 “int A::*pt=&A::y;” 定义了一个数据成员指针 pt, 并指向 y, 则 “c.*pt=20;” 等价于 “c.y=20”。因此第一次显示 x=10, y=20。

在 main() 函数中的 “void (A::*pfun)(int)=A::getx;” 语句定义了一个成员函数指针 pfun, 并指向 getx() 函数, 则 “(c.*pfun)(30);” 相当于 “c.getx(30);”。而语句 “pfun=A::gety;” 使函数指针 pfun 指向 gety(), 则 “(c.*pfun)(40);” 相当于 “c.gety(40);”。因此第二次显示 x=30, y=40。

6.2.2 程序设计与算法分析

1. 编写一个函数, 计算一维数组中的最大元素及其下标, 要求数组以指针方式传递。

分析 此题要求数组以指针方式进行传递, 因此将数组的起始地址与存放最大值下标的单元由指针形式的参数传递, 而数组中最大值元素的地址则由函数返回类型进行传递。程序代码段如下:

```

#include <iostream.h>
int max;
int *findmax(int *p,int n,int *t)
{
    int i;
    max=*p;
    for(i=1;i<n;i++)
        if(max<*(p+i))
        {
            max=*(p+i);
            *t=i;
        }
}

```

```

    }
    return (&max);
}
void main()
{
    int a[] = {0,3,19,27,5,42,8,17,11,1}, n=10, t=0;
    cout<<"最大值: "<<*findmax(a,n,&t)<<" ";
    cout<<"下标为: "<<t<<endl;
}

```

在此程序中, 函数 `*findmax(int *p, int n, int *t)` 的返回类型为指针类型, 将返回数组中元素最大值; 指针型形参 `p` 和 `t` 分别用于接收数组的起始地址和存放最大值下标的单元地址。

由于函数 `findmax` 返回的值 `max` 应在主函数中仍然有效, 故在程序中将 `max` 定义为全局变量(否则会出现警告错误)。注意返回语句“`return (&max);`”返回的是表示最大值变量 `max` 的地址, 因此, 在主函数中进行调用输出时使用的格式为“`*findmax(a,n,&t)`”。而表示最大值下标的变量 `t` 在主函数中是一般整数类型, 故在函数调用中的实参形式应为“`&t`”, 以与形参“`*t`”的指针类型匹配。

2. 要求函数的功能是在一维数组 `a` 中查找 `x` 的值, 若找到则返回所在的下标值, 否则返回 0。分析下列各函数, 找出不符合要求的函数 (C)。

A. `funa(int *a, int n, int x)`

```

{
    *a=x;
    while(a[n]!=x) n--;
    return n;
}

```

B. `funb(int *a, int n, int x)`

```

{
    int k;
    for(k=1; k<=n; k++)
        if(a[k]==x) return k;
    return 0;
}

```

C. `func(int *a, int n, int x)`

```

{
    int *k;
    a[0]=x;
    k=a+n;
    while(*k!=x) k--;
    return k-n;
}

```

```

D. fund(int *a,int n,int x)
{
    int k=0;
    do
        k++;
    while((k<n+1)&&(a[k]!=x));
    if(a[k]==x)
        return k;
    else
        return 0;
}

```

分析 对函数 funa(), 关键是要理解语句 “*a=x;” 的作用, 因为数组名 a 存放着数组的开始地址, 也就是数组第一个元素 a[0] 的地址, 故语句 “*a=x;” 等价于 “a[0]=x;”。

函数 funb() 比较好理解, 由 for 循环控制进行逐一核对查找, 找到即返回 k, 若都核对完仍没有找到, 则返回 0。

函数 func() 的错误一方面是返回类型为指针, 而函数没有声明返回类型(默认为 int), 另一方面是即使加上返回类型, 当找不到 x 时, 返回值也不会为 0。

函数 fund() 的关键是对循环结束条件 “(k<n+1)&&(a[k]!=x)” 的理解, 若在 k<n+1 的情况下, 当 a[k] 不等于 x 为假时结束循环(即 a[k] 等于 x), 此时当然是找到了 x; 否则, 若以 k<n+1 为假时结束循环, 自然是没找到 x。

3. 有 N 个人围成一圈, 顺序编号, 从第一个人开始按 1、2、3 顺序报数, 凡报到 3 的人退出圈子, 然后从出圈的下一个人开始重复此过程。输出出圈序列。

```

#include <iostream.h>
#define N 100
void main( )
{
    int a[N], i, k, n, m, *p;
    cout<<"请输入人数(<100):";
    cin>>n;
    for(i=0; i<n; i++)
        a[i]=i+1;          //为每个人编号
    p=a;
    i=0;                    //下标计数变量
    k=0;                    //1,2,3 报数时的计数变量
    m=0;                    //退出人数计数器变量
    while(m<n)
    {
        if(*(p+i)!=0) k++;
    }
}

```

```

        if(k==3)
        {
            cout<<" "<<*(p+i);
            *(p+i)=0;          //将退出的编号置为 0
            k=0;
            m++;
        }
        i++;
        if(i==n) i=0;          //报数到尾后, i 应恢复为 0
    }
}

```

分析 程序的前半部分比较容易读懂, 通过给变量 n 输入值来确定实际人数(<100), 由数组 $a[N]$ 的前 n 个元素代表每个人, 并进行顺序编号。理解此程序主要是分析指针变量 p 与整型变量 i 、 k 、 m 之间的相互关系与作用。

p 的初始值是数组 a 的首地址, i 、 k 、 m 初始值均为 0。

在 `while()` 循环结构中, 由循环控制条件 $m < n$ 即可知 m 是用于记录出圈人数的, 当出圈人数等于实际人数 n 时循环应结束。在循环体中, 首先是理解将某人的编号置为 0, 即表示此人退出圈; 条件语句 “`if(*(p+i)!=0) k++;`” 就是控制只有编号不为 0 的才参与报数, 若某人报到 3 (即 `if(k==3)` 为真时), 则需进行四項工作: ① 输出此人的编号 (`cout<<" "<<*(p+i);`), ② 将此人的编号置为 0 (`*(p+i)=0;`), ③ 重新开始报数 (`k=0;`), ④ 出圈人数加 1 (`m++;`)。

另外应理解 i 的作用, 它是用于表示数组下标的, 当 i 变化到 n 时, 应从头再对编号不为 0 的人员进行报数, 故当 `if(i==n)` 为真时 $i=0$ 。

理解此程序还有一点应注意, 就是 `while()` 循环执行的次数问题, 并不是只循环 n 次, 因为显然并不是每次 m 都会加 1。

4. 实现单链表的逆置, 并输出逆置前后的结果 (设链表结点只有一个整数元素)。

分析 链表的逆置就是将一个已有链表的尾作为头、头作为尾的变化过程, 又称为链表的反转。程序代码如下:

```

#include <iostream.h>
struct Node
{
    int num;
    Node *next;
};
class Chain
{
    Node *head,*tail;      //定义一个含头、尾指针的链表
public:
    Chain() {head=NULL;}
    void Input(int i);

```

```

    void Output(Node *outhead);
    Node *Gethead( ) {return head; }
};

void Chain::Input(int i) //链表建立函数
{
    Node *pt;
    pt=new(struct Node);
    pt->num=i;
    if(hcad==NULL) head=pt,tail=pt;
    tail->next=pt;
    tail=pt;pt->next=NULL;
}

void Chain::Output(Node *outhead) //链表输出函数
{
    for(Node *pt=outhead;pt!=NULL;cout<<"    "<<pt->num,pt=pt->next);
}

Node *turnback(Node *head) //链表反转函数
{
    Node *p1,*p2,*newp,*newhead=NULL;
    do
    {
        p2=NULL;
        p1=head;
        while(p1->next!=NULL)p2=p1,p1=p1->next;
        if(newhead==NULL)newhead=p1,newp=newhead->ncxt=p2;
        newp=newp->ncxt=p2;
        p2->next=NULL;
    }while(head->next!=NULL);
    return (newhead);
}

void main( )
{
    Chain A;
    int arr[]={5,2,8,1,4,3,9,7,6};
    for(int i=0;i<9;i++)
        A.Input(arr[i]);
    cout<<"\n 原来表: ";
    A.Ouput(A.Gethead( ));
    cout<<"\n 反转表: ";
}

```



```

    A.Output(turnback(A.Gethead()));
    cout<<endl;
}

```

在程序中，我们先定义了一个表示结点的结构体类型 `Node`，其成员只有两个，一个是整型，它表示结点的数据，另一个是指针类型，它存放下一个结点的指针。然后定义了一个表示链表的类结构 `Chain`，它包含了链表的头指针与尾结点指针，还包含了链表的建立函数(`void Input(int i);`)与输出函数(`void Output(Node *outhead);`)，对这两个函数的理解请读者参考其他章节或书籍自行完成。

在链表的逆置函数(`Node *turnback(Node *head)`)中，首先应注意其返回类型为结构指针类型，即实现将逆置后的链表首指针返回；其次是理解链表的逆置处理过程，即对循环结构进行分析：在 `do` 循环结构中，`p1`、`p2` 及 `newp` 是进行反转的中介指针变量，`p1` 开始时指向被反转链表的头指针，而循环“`while(p1->next!=NULL) p2=p1,p1=p1->next;`”的作用是 `p1` 指向链表的尾结点，`p2` 指向倒数第二结点，如图 6-1 所示。

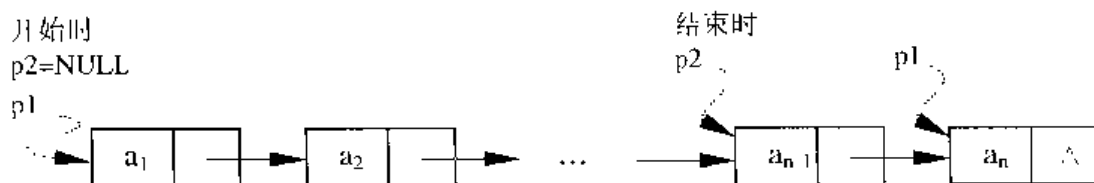


图 6-1 查找链表尾结点指针

如果是第一次，即 `newhead==NULL` 为真，自然应将 `p1` 赋给反转后新链表的头指针 `newhead`，并将下一结点的地址存入其指针成员中(`newhead->next=p2;`)，使尾结点与倒数第二结点相链接，然后记录 `p2` 的指针值(即语句 `newp=newp->next=p2;` 中的 `newp=p2` 功能)，最后再给 `p2` 的指针成员中存入 `NULL` (`p2->next=NULL;`)。这样既形成了只有两个结点的反转新链表，同时保留了其余 `n-1` 个结点的原链表，此处应注意指向其分隔结点的指针变量“`newp`”及其作用。

在以后的各次循环过程中对保留的原链表进行类似的操作，惟一不同的是此时只要将 `p2` 所指的结点链入反转的新链表中，直至将原链表头结点的指针成员变为 `NULL`，即表示反转完成。

6.2.3 填空

1. 下列程序的功能是从键盘输入变量 `i` 的值，然后将 `i` 的值赋给变量 `j`，将 `j` 的值输出，以上操作要求通过指针变量 `p` 进行。

```

#include <iostream.h>
void main( )
{
    int i,j;
    _____ ① _____;
    cin>>*p;
}

```

```

    _____②_____;
    p=&j;
    cout<<_____③_____;
}

```

参考答案:

① `int *p=&i`

② `j=*p`

③ `*p`

分析 首先应考虑题目要求通过指针变量 `p` 进行操作,但程序中没有对 `p` 的定义与初始化语句,因此在①的位置上应填 `int *p=&i`;然后通过指针 `p` 实现将 `i` 的值赋给变量 `j`,故②的位置上应填 `j=*p`;最后利用指针 `p` 将 `j` 的值输出,即③的位置上应填 `*p`。

2. 下列程序的功能是将 `n` 个整数循环右移 `m` 位,即把这 `n` 个数顺序向右移 `m` 位,最后 `m` 个数变成最前面 `m` 个数,例如,10 个数“1 2 3 4 5 6 7 8 9 10”循环右移 4 位变成“7 8 9 10 1 2 3 4 5 6”。

```

#include <iostream.h>
void move(int,int,int*); //移动函数声明
void main( )
{
    const int n=10;
    int m,a[n];
    for(int i=0;i<n;i++)
        cin>>a[i];
    cin>>m;
    move(_____,①_____);
    for(i=0;i<n;i++)
        cout<<a[i]<<" ";
}

void move(int n,int m,int *p)
{
    int i,j,temp;
    for(i=1;i<=m;i++)
    {
        temp=_____②_____;
        for(j=n-2;_____③_____j--)
            *(p+j+1)=_____④_____;
        _____⑤_____;
    }
}

```

参考答案:

- ① n,m,a
- ② *(p+n-1)
- ③ j>=0
- ④ *(p+j)
- ⑤ *p=temp

分析 在函数 `move()` 的定义中, 形参 `p` 被定义为指针变量, 用来指向数组的元素, 因此其对应的实参应为数组名 `a`。这样由于形参和实参都指向数组 `a`, 在 `move()` 中对数组元素循环右移的结果将被带回主函数。

在 `move()` 中共执行 `m` 次循环, 每一次循环将 `n` 个数组元素值向右移动 1 位, 右端移出的数据再放回到数组的左端(称为循环右移 1 次), 其具体操作可分为以下三步:

(1) 将右端元素 `a[n-1]` 的值, 即 `*(p+n-1)` 保存到临时变量 `temp` 中。

(2) 从右端第 2 个元素 `a[n-2]`, 即 `*(p+n-2)` 开始直到左端元素 `a[0]` 依次右移 1 位, 也就是说, 将 `a[n-2]` 的值移到 `a[n-1]`, `a[n-3]` 的值移到 `a[n-2]`, 直到将 `a[0]` 的值移到 `a[1]`, 注意在程序中这样的移动通过指针形参 `p` 来实现。

(3) 将保存在 `temp` 中的 `a[n-1]` 的值放回到左端的 `a[0]`。

3. 下面程序中的函数 `maxword()` 实现从给定的两个由英文单字(词)组成的字符串 `s` 和 `t` 中, 找出其中都包含的最长的相同单字(同一字母的大小写视为不同字符)。约定单字全由英文字母组成, 单字之间由一个或多个空白符分隔。

分析 程序采用以下算法: 自左至右顺序扫视字符串 `s`, 逐个找出单字(单字开始位置和单字长度), 当该单字的长度比已找到的单字更长时, 就从头至尾扫视字符串 `t`, 在从 `t` 中找出与该单字长度相等、字符相同的单字后, 记录该单字的开始位置和长度并回到 `s`, 在其中找一个更长的单字, 上述寻找过程直至字符串 `s` 扫视结束, 最后输出找到的单字。

```
#include <iostream.h>
#include <string.h>
void maxword(char *s,char *t)
{
    char *res,*temp,chs,cht;
    int i,j,found,maxlen=0;
    while(*s!='\0')
    {
        while(*s==' ') s++;           //跳过单字(词)之间的空格
        for(i=0;_____①_____;i++);
        if(i>maxlen)
        {
            chs=s[i];_____②_____;temp=t;found=0;
            while(*temp!='\0'&&!found)
            {
```

```

        while(*temp==' ')temp++;
        for(j=0;_____③_____;j++);
        if(j==i)
        {
            cht=temp[j];_____④_____;
            if(strcmp(s,temp)==0)
            {
                _____⑤_____=i; res=s; found=1;
            }
            temp[j]=cht;
        }
        temp=&temp[j];
    }
    s[i]=chs;
}
s=&s[1];
}
if(maxlen==0)
    cout<<"There is no same word.\n";
else
{
    chs=res[maxlen];
    res[maxlen]='\0';
    cout<<res<<endl;
    res[maxlen]=chs;
}
}
char s[] ="This is C++ progrkmming test",t[] ="This is a test for C++ programming";
void main( )
{
    maxword(s,t);
}

```

参考答案:

- ① s[i]!=' '&& s[i]!='\0'
- ② s[i]='\0'
- ③ temp[j]!=' '&& temp[j]!='\0'
- ④ temp[j]='\0'
- ⑤ maxlen

注意：问题①所在的循环没有具体的循环体，但应实现对单词字符个数的统计，并由 i 记录其统计值，因此①处应填入 `s[i]!=''&&s[i]!='\0'`（因为单词由空格分隔，字符串由 '\0' 结束）。若此时 i 比假定的最长的相同单词长 (`i>maxlen` 为真)，考虑到此单词要与另一字符串中的单词比较，为了临时标识此单词，在②处应添入 `s[i]='\0'`。

由于每次 (`i>maxlen` 为真时) 都要对另一字符串 t 进行一遍从头到尾的扫视，因此不能破坏 t 的开始指针，而将在 t 中找到的单词由指针变量 temp 来进行表示。同上分析可知在③、④处应分别填入 `temp[j]!=''&&temp[j]!='\0'` 与 `temp[j]='\0'`。

问题⑤处比较好理解，因为当 `i>maxlen` 为真，且在 t 字符串中又找到了相等的单词，即字符串比较函数 `strcmp(s,temp)==0` 为真时，应修改记录假定最长相同单字长度的变量 maxlen，因此⑤处应填 maxlen。

4. 下列程序先接收若干用户的姓名和电话号码，按姓名的字典顺序排列后，再输出用户的姓名和电话号码。

```
#include <iostream.h>
#include <string.h>
#include <iomanip.h>
const int N=5;    //设有 5 个用户
class person
{
    char name[10];
    char num[10];
public:
    void getdata(____①____) { strcpy(name,na); strcpy(num,nu); }
    friend void getsort(person pn[N]);
    friend void outdata(person pn[N]);
};
void getsort(person pn[N]) //对对象数组进行选择法排序
{
    int i,j,k;
    person temp;
    for(i=0;i<N-1;i++)
    {
        k=1;
        for(j=i+1;j<N;j++)
            if(strcmp(____②____)>0) k=j;
        temp=pn[k];
        pn[k]=pn[i];
        pn[i]=temp;
    }
}
```

```

}
void outdata(person pn[N]) //输出姓名和电话号码
{
    int i;
    cout<<"姓名      电话号码\n";
    cout<<"-----\n";
    cout.setf(ios::left);
    for(i=0;i<N;i++)
    {   cout.width(10);
        cout<<pn[i].name;
        cout.width(10);
        cout<<pn[i].num<<endl;
    }
}
void main( )
{
    char *na[5]={"Li","Zhang","Ma","Chen","Gao"};
    char *nu[5]={"8765781","5583901","8688080","8079500","9855321"};
    person obj[5];
    for(int i=0;i<5;i++)
        _____③_____;
    person _____④_____;
    getsort(pt);
    outdata(pt);
}

```

参考答案:

- ① char *na,char *nu 或 char na[],char nu[]
- ② pn[k].name,pn[j].name
- ③ obj[i].getdata(na[i],nu[i])
- ④ *pt=obj

分析 ①空位于类成员函数 getdata()的形参表中,根据其后的函数体内容和数据成员的类型,易知①处应填 char *na,char *nu 或 char na[],char nu[]。②空位于进行排序的友元函数 getsort(person pn[N])的比较语句中,根据函数的参数和题意要求,可知此处是对对象数组 pn[N]的 name 成员进行比较,故填 pn[k].name,pn[j].name。在主函数中,注意到定义了对象数组 obj[N],但没有被赋值,因此③处要填 obj[i].getdata(na[i],nu[i]),实现对各对象元素的赋值。空④处应注意到下两函数调用中的实参为 pt,而 pt 又没有被定义且应为指向类对象数组的指针,故此处填*pt=obj。

6.3 上机实验指导

6.3.1 实验目的

- (1) 学习并理解指针的概念和使用方法。
- (2) 学习利用指针表示数组及其元素的常用形式。
- (3) 掌握指针作为函数参数的使用方法。
- (4) 学习并掌握利用指针处理字符串的方法。
- (5) 学会对链表进行建立、插入、删除等操作。

6.3.2 实验内容与补充习题

1. 试分析程序的运行结果并上机验证。

```
#include <iostream.h>
void main( )
{
    int i;
    int a[10],*p=a;
    for(i=0;i<10;i++)
        *p++=i*i;
    for(i=0;i<10;i++)
        cout<<a[i]<<" ";
    cout<<endl;
}
```

运行结果为：

2. 分析下面程序的运行结果及功能，并上机运行验证。

```
#include <iostream.h>
#define M 15
#define N 3
void main( )
{
    static a[M];
    const int j=M-1,*p2=&a[j];
    int i,n,*p1=&a[j];
    for(n=0;n<j;++n)
```

```

{
    for(i=0;i<N;++i)
        while(1)
        {
            if(++p1>p2)    p1=a;
            if(!*p1) break;
        }
        *p1=-1;
    }
    for(i=0;i<M;++i)
        if(!a[i]) cout<<i<<endl;
}

```

运行结果为：

提示 此题实际就是本章程序设计 with 算法分析中第 3 题的变化形式，称为瑟夫问题。

3. 下列程序是用选择排序算法对 10 个整数按增序排序的程序。选择排序算法是先在所有数中比较出最小的数，放置在最前面；然后再比较余下的数据，找出次小的数，放置在次之的位置上；依此类推，直至所有的数都排序为止。请填空完成程序并上机运行验证(要求用指针形式完成)。

```

#include <iostream.h>
#define N 10
void main( )
{
    static a[N];
    int i,*p1,*min,temp;
    int *p2=&a[N-1];
    cout<<"请输入"<<N<<"个整数(之间用空格分隔):\n";
    for(p1=a;_____①_____;++p1)
        cin>>*p1;
    for(i=1;i<N;++i)
    {
        for(_____②_____,min=p1-1;p1<=p2;p1++)
            if(*p1<*min)
                _____③_____=p1;
        temp=_____④_____;
        a[i-1]=*min;
        *min=temp;
    }
}

```



```

    }
    for(i=0,p1=a;i<N;++i,++p1)
        cout<<*p1<<" ";
    cout<<endl;
}

```

4. 下列程序是用指针形式判断输入的字符串是不是回文(若一个字符串顺读与倒读相同,则该字符串为回文,例如 LEVEL 是回文,LEVAL 不是回文),若是则显示 YES,否则显示 NO。请填空完成程序并上机运行验证。

```

#include <iostream.h>
#include <____(1)____>
class huiwen
{
    char *pc;
    int n;
public:
    void getpc(char *c) { pc=c; ____ (2) ____; }
    int sub( );
};
int huiwen::sub( )
{
    char *pi=&pc[0];
    char *pj=&pc[n-1];
    while(*pi==' ') pi++;          //使 pi 指向字符串的第 1 个非空格字符
    while(*pj==' ') pj--;          //使 pj 指向字符串的倒数第 1 个非空格字符
    while((pi<pj)&&(*pi==*pj))
    {
        ____ (3) ____; ____ (4) ____;
    }
    if( ____ (5) ____ ) return 0;
    return 1;
}
void main( )
{
    char s[80];
    cout<<"请输入字符串:";
    cin>>s;
    huiwen hw;
    hw.getpc(s);
}

```

```
    if(hw.sub( ))
        cout<<"YES\n";
    else
        cout<<"NO\n";
}
```

5. 已知 head 指向单链表的第一个结点, 以下函数完成向降序单向的链表中插入一个结点, 插入后链表仍有序。请编主程序调用该函数进行验证。

```
#include <iostream.h>
#include <string.h>
struct student
{
    int info;
    struct student *link;
};
struct student *insert(struct student *head, struct student *stud)
{
    struct student *p0, *p1, *p2;
    p1=head;
    p0=stud;
    if(head==NULL)
    {
        head=p1=p0;
        p0->link=NULL;
    }
    else
    {
        while((p0->info<p1->info)&&(p1->link!=NULL))
        {
            p2=p1; p1=p1->link;
        }
        if(p0->info>=p1->info)
        {
            if(head==p1)
            {
                p0->link=head;
                head=p0;
            }
        }
    }
}
```

```

        else
        {
            p2->link=p0;
            p0->link=p1;
        }
    }
    else
    {
        p1->link=p0;
        p0->link=NULL;
    }
}
return (head);
}
#define M 10
void main( )
{
    //应补充填入程序的位置
}

```

6. 设计程序,使其同时具有 add(求和)、sub(求差)、mul(求积)、div(求商)和 mod(求余)的功能,要求使用函数指针数组来进行处理。

7. 编写一个程序,用随机数方式生成 a[N][M]二维数组的元素值(其中 N、M 由#define 语句指定),然后计算每行的平均值并输出。

8. 用指针方式编写对字符串进行压缩处理的程序,其要求是:压缩掉字符串中多余的空格与相重的字符(只保留一个)。例如,对字符串“I am a teacher about computer.”进行压缩处理后应为:“Iamtechrboup。”

9. 编写程序,将一个英文句子中的单词进行分离并存入一维指针数组中,然后按词典顺序输出这些单词。

10. 编写程序,将某个位数不确定的正整数进行三位分节后输出,例如,输入 1234567,应输出 1,234,567。

* 11. 用指针方式编写一个包括入栈和出栈成员函数的栈操作类程序,然后入栈一组数据,再让数据出栈并显示出栈顺序。

* 12. 编写程序,实现一边读入整数,一边构造一个从大到小顺序链接的链表,直至从键盘读入 '0',然后顺序输出链表上各表元的整数值。要求主函数每读入一个整数,就调用函数 insert(),函数 insert()将还未出现的链表上的整数按从大到小的顺序插入到链表中。(为了插入方便,链表在表首有一个辅助表元。)

第7章 继承与派生

7.1 基本知识点、内容概要与学习要求

7.1.1 基本知识点

1. 继承与派生
 - 派生类的声明。
 - 派生类生成过程。
2. 多继承
 - 多继承的声明。
 - 类族。
3. 类的继承方式
 - 公有继承。
 - 私有继承。
 - 保护继承。
4. 派生类的构造函数和析构函数
5. 派生中成员的标识与访问
 - 作用域分辨。
 - 基类私有成员的访问。
 - 引入派生类后的对象指针。
6. 虚基类
 - 虚基类的声明。
 - 虚基类及其派生类的构造函数。

7.1.2 内容概要

1. 继承与派生

所谓继承，是指新的类从已有类那里得到已有的特性。从另一个角度来看，从已有类产生新类的过程就是类的派生。已有的类称为基类或父类，产生的新类称为派生类或子类。

1) 派生类的声明

```
class 派生类名:[继承方式]基类名
{
    派生类成员声明;
};
```

2) 派生类生成过程

派生类生成过程为：吸收基类成员，改造基类成员，添加新的成员。

2. 多继承

在派生类的声明中，基类可以有一个，也可以有多个。如果只有一个基类，则这种继承方式称为单继承；如果基类有多个，则这种继承方式称为多继承。

多继承的声明格式如下：

```
class 派生类名:[继承方式]基类名 1, [继承方式]基类名 2, ..., [继承方式]基类名 n
{
    派生类成员声明;
};
```

在派生过程中，派生出来的新类也同样可以作为基类再继续派生新的类，此外，一个基类可以同时派生出多个派生类，这样就形成了一个相互关联的类的家族，称为类族。在类族中，直接参与派生出某类的基类称为直接基类；基类的基类甚至更高层的基类称为间接基类。

3. 类的继承方式

类的继承方式有 `public`(公有)、`private`(私有)和 `protected`(保护)继承三种，不同的继承方式，会导致基类成员原来的访问属性在派生类中有所变化。

1) 公有继承

当类的继承方式为 `public` 继承时，基类的 `public` 和 `protected` 成员的访问属性在派生类中不变，而基类的 `private` 成员仍保持私有属性。

2) 私有继承

当类的继承方式为 `private` 继承时，基类中的 `public` 成员和 `protected` 成员都以私有成员身份出现在派生类中，而基类的 `private` 成员在派生类中不可访问。

3) 保护继承

保护继承中，基类的 `public` 和 `protected` 成员都以保护成员的身份出现在派生类中，而基类的 `private` 成员不可访问。

4. 派生类的构造函数和析构函数

基类的构造函数和析构函数是不能被继承的。如果基类没有定义构造函数，派生类也可以不定义构造函数，全部采用默认的构造函数，这时新增成员的初始化工作可以用其他公有函数来完成；如果基类定义了带形参表的构造函数，派生类就必须加入新的构造函数，提供一个将参数传递给基类构造函数的途径，保证在基类进行初始化时能够获得必要的数据。而派生类的析构函数与基类相对独立，可根据需要自行定义。

(1) 派生类构造函数声明的语法形式为:

```
派生类名::派生类名(参数总表):基类名 1(参数表 1), ..., 基类名 n(参数表 n),  
    内嵌对象名 1(内嵌对象参数表 1), ..., 内嵌对象名 m(内嵌对象参数表 m)  
{  
    派生类新增成员的初始化语句;  
}
```

派生类构造函数的执行顺序一般是先祖先(基类), 再客人(内嵌对象), 后自己(派生类本身)。

(2) 派生类析构函数的定义方法与没有继承关系的类中析构函数的定义方法完全相同, 但它的执行顺序和构造函数正好严格相反, 是先自己(派生类本身), 再客人(内嵌对象), 后祖先(基类)。

5. 派生中成员的标识与访问

在派生类的访问中, 有两个问题需要解决: 第一是惟一标识问题, 第二是可见性问题。对于在不同的作用域声明的标识符, 可见性原则是: 若存在两个或多个具有包含关系的作用域, 则对于外层声明的标识符, 如果在内层没有声明其同名标识符, 那么它在内层仍可见; 如果内层声明了其同名标识符, 则它在内层不可见。后一现象实质上是内层变量覆盖了外层同名变量, 故称之为同名覆盖。

1) 作用域分辨符

作用域分辨符的使用形式为:

类名::成员名;

类名::成员函数名(参数表);

注意: 在多继承中, 如果某个派生类的部分或全部直接基类是从另一个共同的基类派生而来的, 则在这些直接基类中, 从上一级基类继承来的成员就拥有相同的名称。因此, 派生类中就会产生同名现象, 对这种类型的同名成员要使用作用域分辨符来惟一标识, 而且必须用直接基类来进行限定。

2) 基类私有成员的访问

- 在类定义体中增加保护段。
- 将需访问基类私有成员的派生类成员函数声明为基类的友元。

3) 引入派生类后的对象指针

任何被说明为指向基类对象的指针都可以指向它的公有派生类对象。

6. 虚基类

虚基类用来解决多继承中同名成员的惟一标识问题。

1) 虚基类的声明

虚基类是在派生类的声明过程中声明的, 其语法形式为

```
class 派生类名:virtual 继承方式 基类名
```

2) 虚基类及其派生类的构造函数

虚基类的初始化与一般的多继承的初始化在语法上是一样的, 但构造函数的调用次序

不同，它的调用规则为：

- 虚基类的构造函数在甘虚基类之前调用；
- 若同一层次中包含多个虚基类，则这些虚基类的构造函数按它们说明的次序调用；
- 若虚基类由非虚基类派生而来，则仍然先调用基类构造函数，再调用派生类的构造函数。

7.1.3 学习要求

- 理解类层次的概念及实现类层次的方法。
- 掌握单继承的定义格式及成员访问控制的方法。
- 掌握多继承的定义方式和向基类构造函数传递参数的方法。
- 了解继承中二义性问题及解决的方法。
- 了解虚基类的作用，掌握虚基类的用法。

7.2 典型例题（或典型算法）分析

7.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

1. 程序代码如下：

```
#include <iostream.h>
#include <math.h>
class Point
{
private:
    float X,Y;
public:
    void InitP(float xx=0, float yy=0) {X=xx;Y=yy;}
    void Move(float xOff,float yOff) {X+=xOff;Y+=yOff;}
    float GetX(){return X;}
    float GetY(){return Y;}
};
class Rectangle:public Point
{
private:
    float W,H;
public:
    void InitR(float x,float y,float w,float h)
```

```
{InitP(x,y);W=w;H=h;}  
float GetH( ){return H;}  
float GetW( ){return W;}  
};  
void main( )  
{  
    Rectangle rect;  
    rect.InitR(2,3,20,10);  
    rect.Move(3,2);  
    cout<<"The data of rect(X,Y,W,H):"<<endl,  
    cout<<rect.GetX( )<<","  
        <<rect.GetY( )<<","  
        <<rect.GetW( )<<","  
        <<rect.GetH( )<<endl;  
}
```

运行结果为：

```
The data of rect(X,Y,W,H):  
5,5,20,10
```

分析 这是一个从 Point 类派生出新的 Rectangle(矩形)类的程序。Point 类由坐标(x,y)构成，可以通过 Move()、GetX()和 GetY()移动和显示点的坐标。矩形是由一个点加上长、宽构成，矩形的点具备了 Point 类的全部特征，同时，矩形自身也有一些特点，这就需要在继承 Point 类的同时添加新的成员。Point 类和 Rectangle 类的派生关系可以用图 7-1 来描述

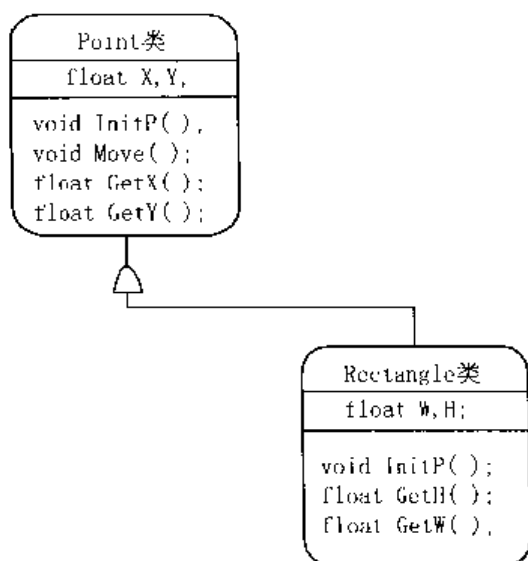


图 7-1 Point 类和 Rectangle 类的派生关系

这里首先声明了基类 `Point`，派生类 `Rectangle` 继承了 `Point` 类的全部成员(默认的构造和析构函数除外)，因此，在派生类中，实际所拥有的成员就是从基类继承过来的成员与派生类新声明的成员的总和。继承方式为公有继承，这时基类中的公有成员在派生类中访问属性保持原样，派生类的成员函数及对象可以访问到基类的公有成员(例如，在派生类函数成员 `InitR` 中直接调用基类的函数 `InitP()`)，但是无法访问基类的私有数据(例如基类的 `X`、`Y`)。基类原有的外部接口(例如基类的 `GetX()` 和 `GetY()` 函数)变成了派生类外部接口的一部分。当然，派生类自己新增的成员之间都是可以互相访问的。

`Rectangle` 类继承了 `Point` 类的成员，也就实现了代码的重用，同时，通过新增成员，加入了自身的独有特征，实现了程序的扩充。

在主函数 `main()` 中，首先声明了一个派生类的对象 `rect`，对象生成时调用了系统所产生的默认构造函数，这个函数什么都没做；然后通过派生类的对象，访问派生类的公有函数 `InitR()` 和 `Move()` 等，也访问了派生类从基类继承来的公有函数 `GetX()` 和 `GetY()`。这样我们看到，一个基类以公有方式产生了派生类之后，派生类的成员函数以及派生类的对象是如何访问从基类继承的公有成员的。

2. 程序代码如下：

```
#include <iostream.h>
class base1
{
public:
    int n;
    void fun() {cout<<"This is base1,n="<<n<<endl;}
};
class base2
{
public:
    int n;
    void fun() {cout<<"This is base2,n="<<n<<endl;}
};
class Derive:public base1,public base2
{
public:
    int n;
    void fun() {cout<<"This is Derive,n="<<n<<endl;}
};
void main()
{
    Derive obj;
    obj.n=1;
```

```

obj.fun();
obj.base1::n=2;
obj.base1::fun();
obj.base2::n=3;
obj.base2::fun();
}

```

运行结果为：

This is Derive,n=1

This is base1,n=2

This is base2,n=3

分析 这是一个在多继承中使用作用域分辨符来标识成员的程序。程序中声明了基类 base1 和 base2，由基类 base1 和 base2 共同公有派生产生了新类 Derive。两个基类中都定义了成员数据 n 和成员函数 fun()，在派生类中新增了同名的两个成员，这时的 Derive 类共含有六个成员，而这六个成员只有两个名字。类的派生关系及派生类的结构如图 7-2 所示。

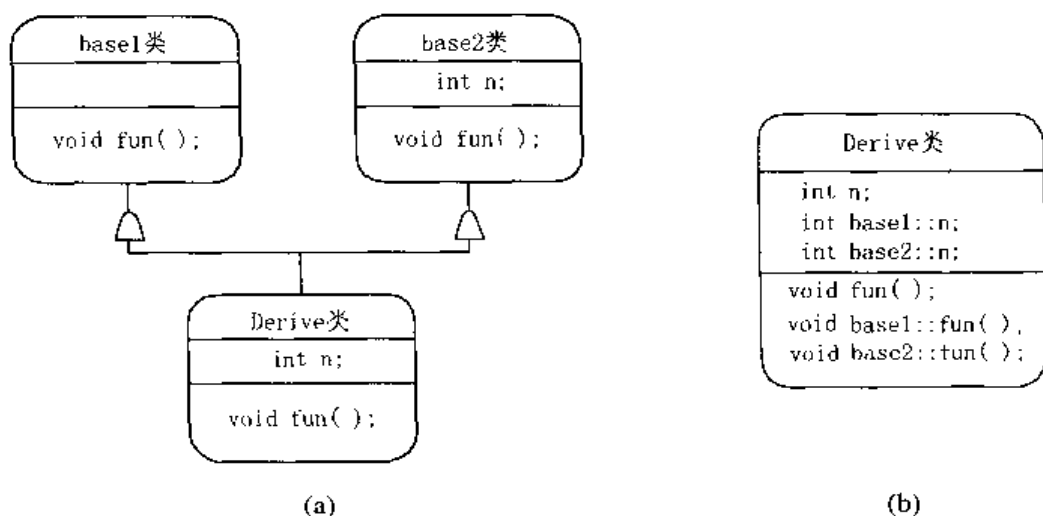


图 7-2 多继承情况下派生类与基类的结构图

(a) 继承关系；(b) 成员关系

从结果来看，使用作用域分辨符也能很清楚地标识从不同基类继承过来的成员，达到访问成员的目的，从而解决了同名覆盖情况下的成员屏蔽问题。

在多继承中，派生类成员将覆盖所有基类的同名成员。这时，使用“对象名.成员名”方式可以惟一标识和访问派生类新增成员，基类的同名成员可以使用作用域分辨符访问。

但是，在上述例题中，如果派生类没有声明同名成员，那么使用“对象名.成员名”就无法确定访问成员，这时从 base1、base2 两个不同基类继承过来的成员具有相同的名称，同时具有相同的作用域。系统仅仅根据这些信息根本无法判断到底是调用哪个基类的成员，因此必须通过作用域分辨符来标识成员。

3. 程序代码如下:

```
#include <iostream.h>
class base
{
public:
    base(){cout<<"this is base class!"<<endl;}
};
class base1
{
public:
    base1(){cout<<"this is base1 class!"<<endl;}
};
class level1:public base1,virtual public base
{
public:
    level1(){cout<<"this is level1 class!"<<endl;}
};
class level2:public base1,virtual public base
{
public:
    level2(){cout<<"this is level2 class!"<<endl;}
};
class toplevel:public level1,virtual public level2
{
public:
    toplevel(){cout<<"this is toplevel class!"<<endl;}
};
void main( )
{
    toplevel topobj;
    return;
}
```

运行结果为:

this is base class!

This is base1 class!

This is level2 class!

This is base1 class!

This is level1 class!

This is toplevel class!

分析 这是在一个含有虚基类的派生类中，说明构造函数的执行顺序的程序。程序中无论是基类 base 与 base1，还是派生类 level1、level2 和 toplevel，都有自己的构造函数。建立 toplevel 类对象 topobj 时，因为 level2 是虚基类，所以先执行虚基类 level2 的构造函数；而 level2 是 base 与 base1 的派生类，base 又是 level2 的虚基类，因此，实际上是先分别执行 base、base1、level2 的构造函数，然后才执行 level2 的构造函数。同理，对 level1 构造函数的执行也是通过分别执行 base1、level1 的构造函数来实现的。这充分证明了虚基类的构造函数是在非虚基类之前调用的，若同一层次中包含多个虚基类，则这些虚基类的构造函数按它们说明的次序调用。

4. 程序代码如下：

```
#include <iostream.h>

class B
{
    int x1,x2;
public:
    void Init(int n1,int n2){x1=n1;x2=n2;}
    int inc1(){return ++x1;}
    int inc2(){return ++x2;}
    void disp(){cout<<"B.x1="<<x1<<" ,x2="<<x2<<endl;}
};

class D1:B
{
    int x3,
public:
    D1(int n3){x3=n3;}
    void Init(int n1,int n2) { B::Init(n1,n2); }
    int inc1() { return B::inc1(); }
    int inc2() { return B::inc2(); }
    int inc3() { return ++x3; }
    void disp() { cout<<"D1.x3="<<x3<<endl; }
};

class D2:public B
{
    int x4;
public:
    D2(int n4) { x4=n4; }
    int inc4() { return ++x4; }
```

```
void disp( ) { cout<<"D2,x4="<<x4<<endl; }  
};  
void main( )  
{  
    B b;  
    b.Init(-2,-2);  
    b.disp( );  
    D1 d1(3);  
    d1.Init(5,5);  
    d1.inc1( );  
    d1.disp( );  
    D2 d2(6);  
    d2.Init(-4,-4);  
    d2.disp( );  
    d2.inc1( );  
    d2.inc2( );  
    d2.disp( );  
    d2.B::inc1( );  
    d2.disp( );  
}
```

运行结果为:

```
B: x1=-2,x2=-2  
D1: x3=3  
D2: x4=6  
D2: x4=6  
D2: x4=6
```

分析 这是一个关于私有继承和公有继承比较的程序, 特别要注意其中各成员的访问属性和访问方式。这里 B 是基类, D1 是基类 B 的私有派生类, 因此 B 类的所有公有成员函数都成为 D1 类的私有成员, D1 类的对象不能访问它们, 但 D1 类的公有成员函数可以访问。为了使 D1 类的对象可以调用 B 类的函数 inc1() 和 inc2(), 在 D1 中对 inc1() 和 inc2() 重新进行定义, 间接地进行调用。D2 是基类 B 的公有派生类, 因此 B 类的所有公有成员都被继承下来, 仍作为公有成员使用。

7.2.2 程序设计与算法分析

1. 设计父亲类 father、母亲类 mother 和子女类 child, 其主要数据是姓名、年龄与民族, 子女继承父亲的姓、母亲的民族, 最后输出子女及其父母的姓名、年龄和民族信息。

分析 首先根据题意要求, 设计一个 father 类和 mother 类, 其中包含私有属性数据成

员 age(年龄), 保护属性数据成员 fname(姓)、sname(名)和 nation(民族), 成员函数 getage()、getfname()、getnation()、show()等, 它们分别用于年龄访问、姓氏传递、民族传递及数据显示, 还定义相应的构造函数, 用于初始化其类对象; 然后再从 father 类和 mother 类以派生的方法定义 child 类。程序代码如下:

```
#include <iostream.h>
#include <string.h>
class father
{
    int age;                //父年龄
protected:
    char fname[4];          //父姓
    char sname[10];         //父名
    char nation[4];         //父民族
public:
    father( ) { };
    father(char *fn,char *sn,int n,char *na)
    {
        strcpy(fname,fn);
        strcpy(sname,sn);
        age=n;
        strcpy(nation,na);
    }
    void getage(int n) {age=n; }
    char *getfname( ) { return fname; }
    void show( )
    {
        cout<<fname<<sname<<","<<age<<"岁,"<<nation<<"族"<<endl;
    }
};
class mother
{
    int age;                //母年龄
protected:
    char fname[4];          //母姓
    char sname[10];         //母名
    char nation[4];         //母民族
public:
    mother( ) { };
    mother(char *fn,char *sn,int n,char *na)
    {
```

```

        strcpy(fname,fn);
        strcpy(sname,sn);
    age=n;
        strcpy(nation,na);
    }
    char *getnation( ) { return nation; }
    void show( )
    {
        cout<<fname<<sname<<" " <<age<<"岁", "<<nation<<"族"<<endl;
    }
};

class child:public father,public mother
{
private:
    father *myfather;
    mother *mymother;
public:
    child(father &fa,mother &mo,char *na,int n):myfather(&fa),mymother(&mo)
    {
        strcpy(father::fname,fa.getfname( ));
        strcpy(father::sname,na);
        father::getage(n);
        strcpy(father::nation,mo.getnation( ));
    }
    void show( )
    {
        cout<<" 姓名: ";father::show( );
        cout<<" 父亲: ";myfather->show( );
        cout<<" 母亲: ";mymother->show( );
    }
};

void main( )
{
    father fa1("李","国强",45,"汉"),fa2("王","伟",28,"川");
    mother mo1("呼","晓丽",41,"满"),mo2("许","英霞",26,"汉");
    child ch1(fa1,mo1,"盛男",17),ch2(fa2,mo2,"燕萍",3);
    ch1.show( );
    ch2.show( );
}

```

运行结果为:

姓名: 李盛男, 17 岁, 满族

父亲: 李国强, 45 岁, 汉族

母亲: 呼晓丽, 41 岁, 满族

姓名: 王燕萍, 3 岁, 汉族

父亲: 王伟, 28 岁, 回族

母亲: 许英霞, 26 岁, 汉族

分析 程序中关于 father 类和 mother 类的定义比较好理解, 现主要对派生类 child 进行分析。在 child 类中, 定义了两个私有的 father 类和 mother 类指针型对象 *myfather 与 *mymother, 用于传递父亲与母亲的相关信息, 而在其构造函数 “child(father &fa, mother &mo, char *na, int n):myfather(&fa),mymother(&mo)” 中, 以 father 类的成员作为其相应的派生成员, 并从 father 类继承姓氏 “strcpy(father::fname,fa.getfname());”, 从 mother 类继承民族 “strcpy(father::nation,mo.getnation());”。由于从 father 类派生的成员 age 具有基类私有属性, 因此要用 father::getage(n)的方式为其赋值。在 show()函数中, 因为指明以 father 类的成员作为 child 类的派生成员, 所以可以用 “father::show();”的方式输出 child 类对象(孩子)的相关信息。

2. 分析程序, 解答问题。根据基类指针在指向其派生类对象时的关系, 设计基类指针的应用程序如下:

```
#include <iostream.h>
class base
{
    int x;
public:
    void Setx(int i) { x=i; };
    int Getx( ) {return x; }
};
class Derived:public base
{
    int y;
public:
    void Sety(int i) { y=i; }
    int Gety( ) { return y; }
};
void main( )
{
    base *p;
    base b_obj;
    Derived d_obj;
```



```

p=&b_obj;
p->Setx(11);
cout<<"base object x:"<<p->Getx()<<endl;
p=&d_obj;
p->Setx(55);
cout<<"Derived object x:"<<p->Getx()<<endl;
d_obj.Sety(99);
cout<<"Derived object y:"<<d_obj.Gety()<<endl;
}

```

运行结果为:

base object x: 11

Derived object x: 55

Derived object y: 99

问题 在程序中, 语句表达式“d_obj.Sety(99);”与“d_obj.Gety();”可否用“p->Sety(99);”和“p->Gety();”代替?

分析 不可以。因为基类指针虽然可以指向其派生类的对象, 但是, 只可以访问派生类对象中从该基类继承下来的那些成员(如 Setx()、Getx()等), 基类指针指向的对象只认识基类的成员, 它不知道有关派生类新声明的成员信息。

7.2.3 填空

1. 下列程序是一个从 A 类公有派生一个新类 B 的程序。请根据图 7-3 所示的派生类 B 对象 b 的构成情况及运行结果, 填空完成程序并上机运行验证。

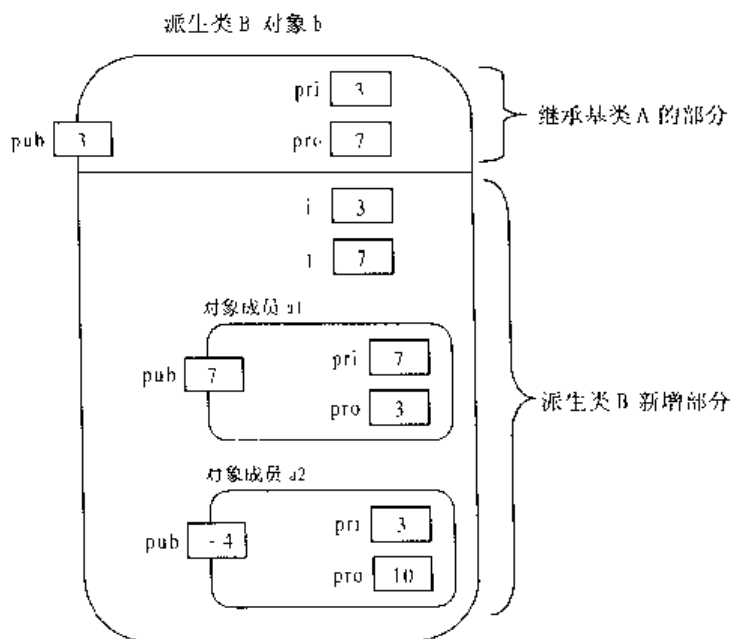


图 7-3 派生类 B 对象 b 的构成示意

程序代码如下:

```
#include <iostream.h>
class A
{
    int pri;
protected:
    int pro;
public:
    int pub;
    A(int i,int j,int k) { pri=i;pro=j;pub=k; }
    void print( )
    {
        cout<<pri<<"\t"<<pro<<"\t"<<pub<<endl;
    }
};
class B:public A
{
    int i,j;
    A a1,a2;
public:
    B(int x,int y): _____①_____ { i=x; j=y; }
    void print( )
    {
        A::print( );
        cout<<i<<"\t"<<j<<endl;
        a1.print( );
        a2.print( );
    }
};
void main( )
{
    B b(3,7);
    cout<<sizeof(b)<<"\n";
    _____②_____;
}
```

运行结果为:

```
_____③_____
3    7    3
3    7
```

```
7   3   7
3   1  -4
```

参考答案:

- ① A(x,y,x),a1(y,x,y),a2(x,x+y,x-y)
- ② b.print()
- ③ 44

分析 这里首先声明了基类 A。因为继承方式为公有继承，派生类 B 就继承了 A 类的全部成员(构造函数和析构函数除外)，且基类中的公有成员在派生类中访问属性保持原样。由于在基类 A 中定义了带有形参表的构造函数，派生类 B 就必须加入新的构造函数，又注意到派生类 B 新增定义了两个 A 类对象成员 a1、a2，故在①处应提供将参数传递给基类 A 的构造函数与对象成员 a1、a2 进行初始化的途径，再根据运行结果，可知①处应填 A(x,y,x), a1(y,x,y),a2(x,x+y,x-y)。

主函数 main()中首先声明了一个派生类 B 的对象 b，然后输出对象 b 所占的内存单元数，由于派生类 B 对象从基类 A 中继承了 3 个整型数据成员 pub、pri 和 pro，新增了 2 个整型数据成员 i 和 j 以及 2 个 A 类对象成员 a1 和 a2，而每个 A 类对象都包含 3 个整型数据成员，因此派生类 B 对象 b 共有 11 个整型数据成员，占内存 44 个单元(字节)。

②处是派生类 B 对象 b 调用其成员函数 b.print()，通过嵌套调用 A::print()、a1.print()、a2.print()及执行“cout<<i<<“\t”<<j<<endl;”语句，输出对象 b 中的 11 个数据成员的值。

2. 下列程序中声明一个圆类 circle 和一个桌子类 table，另外声明一个圆桌类 roundtable，它是由 circle 和 table 两个类派生的，要求声明一个圆桌类对象，并输出圆桌的高度、面积和颜色。请填空完成程序并上机运行验证。

```
#include <iostream.h>
#include<string.h>
class circle
{
    double radius;
public:
    circle(double r){ radius=r;}
    double get_area( ){return _____①_____};
};
class table
{
    double height;
public:
    table(double h){ height=h;}
    double get_height( ){return height;}
```

```
};  
class roundtable: public table,public circle_  
{  
    char *color;  
public:  
    roundtable(double h,double r,char c[ ]):_____②_____  
    {  
        color=new char[strlen(c)+1];  
        _____③_____;  
    }  
    char *get_color(){return color;}  
};  
void main( )  
{  
    roundtable rt(0.8,1.0,"白色");  
    cout<<"圆桌数据: "<<endl;  
    cout<<"圆桌高度: "<<rt.get_height()<<endl;  
    cout<<"圆桌面积: "<<rt.get_area()<<endl;  
    cout<<"圆桌颜色: "<<rt.get_color()<<endl;  
}
```

参考答案:

- ① $3.1416 * radius * radius$
- ② `circle(r),table(h)`
- ③ `strcpy(color,c)`

分析 这是一个多继承的程序,roundtable类是由table类和circle类公有派生的,这里特别要注意的是派生类构造函数的实现。①比较简单,是填入圆面积的计算公式。②是派生类的构造函数,填入的内容应是对两个基类分别初始化。③是通过字符串复制函数将c的内容复制到color中。

7.3 上机实验指导

7.3.1 实验目的

- (1) 学习使用继承方式派生新类的方法。
- (2) 进一步加深对访问权限的理解。
- (3) 掌握派生类构造函数和析构函数的调用次序。

7.3.2 实验内容与补充习题

1. 下列程序中, 基类 base 和派生类 d1、d2 中都含有私有、保护和公有成员, d1 类是 base 的派生类, d2 是 d1 的派生类。试分析下列程序的访问权限。

```
#include <iostream.h>
class base
{
private:
    int n1;
protected:
    int k1;
public:
    base() {n1=0;k1=1;}
    void fun1() {cout<<n1<<k1<<endl;}
};
class d1:public base
{
private:
    int n2;
protected:
    int k2;
public:
    d1() {n2=10;k1=11;}
    void fun2() {cout<<n1<<k1<<endl; cout<<n2<<k2<<endl;}
};
class d2:public d1
{
private:
    int n3;
protected:
    int k3;
public:
    d2() {n3=20;k3=21;}
    void fun3() {cout<<n1<<k1<<endl;cout<<n2<<k2<<endl; cout<<n3<<k3<<endl;}
};
void main()
{
    base baseobj;
```

```

    d1 d1obj;
    d2 d2obj;
    baseobj.fun1();
    d1obj.fun2();
    d2obj.fun3();
}

```

(1) 回答下列问题:

① 派生类 d1 中成员函数 fun2() 能否访问基类 base 中的成员 fun1()、n1 和 k1?

② 派生类 d1 的对象能否访问基类 base 中的成员 fun1()、n1 和 k1?

③ 派生类 d2 中成员函数 fun3() 能否访问直接基类 d1 中的成员 fun2(), n2 和 k2? 能否访问基类 base 中的成员 fun1(), n1 和 k1?

④ 派生类 d2 的对象能否访问直接基类 d1 中的成员 fun2()、n2 和 k2? 能否访问基类 base 中的成员 fun1()、n1 和 k1?

(2) 以上程序有错, 请改正并上机验证。

2. 分析并写出下列程序的运行结果, 然后上机运行验证。

```

#include <iostream.h>
class person
{
    char *name;
    int age;
    char *add;
public:
    person() {cout<<"person class constructor."<<endl;}
    ~person() {cout<<"person class destructor."<<endl;}
};
class student:public person
{
    char *department;
    int level;
public:
    student() {cout<<"student class constructor."<<endl;}
    ~student() {cout<<"student class destructor "<<endl;}
};
class teacher:public person
{
    char *major;
    float salary;
}

```

```

public:
    teacher() {cout<<"teacher class constructor."<<endl;}
    ~teacher() {cout<<"teacher class destructor."<<endl;}
};

void main()
{
    student s1;
    teacher t1;
}

```

运行结果为:

```

_____
_____
_____
_____
_____
_____
_____
_____
_____

```

3. 下列程序是一个有关虚基类及其派生类的初始化的程序。如果虚基类定义有非默认形式的(即带形参)构造函数,在整个继承结构中,直接或间接继承虚基类的所有派生类,都必须在构造函数的成员初始化表中列出对虚基类的初始化。分析程序,写出程序运行结果并回答问题。

```

#include <iostream.h>
class B1
{
public:
    int n1;
    B1(int in_n1){n1=in_n1;cout<<"B1,n1="<<n1<<endl;}
};
class B21:virtual public B1
{
public:
    int n21;
    B21(int a):B1(a){n21=a;cout<<"B21,n21="<<n21<<endl;}
};
class B22:virtual public B1
{

```

```
public:
    int n22;
    B22(int a):B1(a){n22=a;cout<<"B22,n22="<<n22<<endl;}
};
class B3:public B21,public B22
{
public:
    int n3;
    B3(int a):B1(a),B21(a),B22(a){n3=a;cout<<"B3,n3="<<n3<<endl;}
};
void main()
{
    B3 obj(5);
}
```

运行结果为:

```
_____
_____
_____
_____
```

问题分析:

(1) 如果程序运行结果为:

B1,n1=5

B22,n22=5

B21,n21=5

B3,n3=5

上述程序应该怎么改?

(2) 如果将 “B3(int a):B1(a),B21(a),B22(a){n3=a;cout<<"B3,n3="<<n3<<endl;}” 中的 B21(a)和 B22(a)的位置调换,程序的运行结果是否会有变化?

4. 编写程序,声明一个哺乳动物 Mammal 类,再由此派生出狗 Dog 类,要求类中必须包含输出信息的构造函数与析构函数;声明一个 Dog 类的对象,运行程序,观察基类与派生类的构造函数与析构函数的调用顺序。

5. 编写程序,声明一个 Shape 基类,再派生出 Rectangle 和 Circle 类,二者都有 GetArea() 函数计算对象的面积。使用 Rectangle 类创建一个派生类 Square。

6. 编写一个能输入和输出学生和教师数据的程序,学生数据有编号、姓名、班号和成绩;教师数据有编号、姓名、职称和部门。要求声明一个 person 类,并作为学生数据操作类 student 和教师数据操作类 teacher 的基类。

*7. 编写一个对字符串操作的程序，其中有一个简单的串类 `string`，包含设置字符串、返回字符串长度及内容等功能。另外，设计一个具有编辑功能的 `edit_string` 类，它是 `string` 的派生类，在其中设置一个光标，使其能支持在光标处的插入、替换和删除等编辑功能。

*8. 编写一个小型公司人员管理的程序，要求存储公司人员的编号、姓名、级别和月工资。该公司主要有四类人员：经理、兼职技术人员、销售员和销售经理。不同人员月工资的计算方法为：经理固定月薪 8000 元，兼职技术人员按每小时 100 元领取月薪，销售人员按当月销售额的 5% 提成，销售经理月薪为固定月薪 5000 元加销售总额的 5%。

第8章 多态性

8.1 基本知识点、内容概要与学习要求

8.1.1 基本知识点

1. 多态性的概念
2. 运算符重载
 - 运算符重载的规则。
 - 运算符重载为成员函数。
 - 运算符重载为友元函数。
3. 虚函数
 - 虚函数的定义及使用。
 - 虚函数的限制。
4. 抽象类
 - 纯虚函数。
 - 抽象类。

8.1.2 内容概要

1. 多态性的概念

多态性是一种实现“一种接口，多种方法”的技术，是面向对象程序的重要特性。从实现的角度来讲，多态性可以分为静态多态性和动态多态性两种。静态多态性在编译的过程中确定了同名操作的具体操作对象，而动态多态性则在程序运行过程中才动态地确定操作所针对的具体对象。函数重载和运算符重载就属于静态多态性。虚函数是通过动态联编完成的。

2. 运算符重载

1) 运算符重载的规则

(1) C++中的运算符除了少数几个(类属关系运算符“.”、作用域分辨符“::”、成员指针运算符“*”、“sizeof”运算符和三目运算符“?:”)之外，全部可以重载，而且只能重载C++中已有的运算符，不能臆造新的运算符。

(2) 重载之后运算符的优先级和结合性都不能改变,也不能改变运算符的语法结构,即单目运算符只能重载为单目运算符,双目运算符只能重载为双目运算符。

(3) 运算符重载后的功能应当与原有功能相类似。

(4) 重载运算符含义必须清楚,不能有二义性。

2) 运算符重载的形式

运算符重载的形式包括重载为类的成员函数和重载为类的友元函数。

3) 运算符重载为成员函数

运算符重载实质上就是函数重载,当运算符重载为成员函数后,它就可以自由地访问本类的数据成员了。在程序设计过程中,总是通过该类的某个对象来访问重载的运算符。

运算符重载为类的成员函数的语法形式为:

函数类型 operator 运算符(形参表)

```
{
    函数体;
};
```

(1) 双目运算 oprd1@oprd2。

对于双目运算符@,如果要重载@为类的成员函数,使之能够实现表达式 oprd1@oprd2,那么,若 oprd1 为 A 类的对象,则应当把@重载为 A 类的成员函数,该函数只有一个形参,形参的类型是 oprd2 所属的类型。经过重载之后,表达式 oprd1@oprd2 就相当于函数调用 oprd1.operator @(oprd2)。

(2) 单目运算。

• 前置单目运算@oprd

对于前置单目运算符@,如“-”(负号)、“++”等,如果要将它们重载为类的成员函数,用来实现表达式@oprd,那么,若 oprd 为 A 类的对象,则@应当重载为 A 类的成员函数,且没有形参,经过重载之后,表达式@oprd 相当于函数调用 oprd.operator @()。

• 后置单目运算 oprd@

对于后置运算符@,如“++”和“-”,如果要将它们重载为类的成员函数,用来实现表达式 oprd ++或 oprd --,则若 oprd 为 A 类的对象,那么运算符@就应当重载为 A 类的成员函数,这时函数要带有一个整型(int)形参。重载之后,表达式 oprd ++ 和 oprd -- 就相当于函数调用 oprd.operator ++(0)和 oprd.operator --(0)。

4) 运算符重载为友元函数

运算符也可以重载为类的友元函数,这样它就可以自由地访问该类的任何数据成员。这时,运算所需要的操作数都需要通过函数的形参来传递。但是,有些运算符不能重载为友元,如“=”、“()”、“[]”和“->”。

运算符重载为类的友元函数的语法形式为:

friend 函数类型 operator 运算符(形参表)

```
{
    函数体;
}
```

(1) 双目运算 `oprd1@oprd2`。

对于双目运算符`@`，如果`oprd1`为A类的对象，则应当把`@`重载为A类的友元函数，该函数有两个形参，其中至少一个形参(`oprd1`)的类型是A类。经过重载之后，表达式`oprd1@oprd2`就相当于函数调用`operator @(oprd1, oprd2)`。

(2) 单目运算。

• 前置单目运算`@oprd`

对于前置单目运算符`@`，如“`-`”(负号)、“`++`”等，如果要实现表达式`@oprd`，那么，若`oprd`为A类的对象，则`@`可以重载为A类的友元函数，函数的形参为A类的对象。经过重载之后，表达式`@oprd`相当于函数调用`operator @(oprd)`。

• 后置单目运算`oprd@`

对于后置运算符`@`，如“`++`”和“`--`”，如果要实现表达式`oprd ++`或`oprd --`，那么，若其中`oprd`为A类的对象，则运算符就可以重载为A类的友元函数，这时函数的形参有两个，一个是A类的对象`oprd`，另一个是整型(int)形参。重载后，表达式`oprd ++`和`oprd --`就相当于函数调用`operator ++(oprd, 0)`和`operator --(oprd, 0)`。

3. 虚函数

虚函数是引入了派生概念以后，用来表现基类和派生类的成员函数之间的一种关系的，它是动态联编的基础。

1) 虚函数的定义

虚函数的定义是在基类中进行的，当基类中的某个成员函数被声明为虚函数后，此虚函数就可以在一个或多个派生类中被重新定义。在派生类中重新定义时，其函数原型(包括返回类型、函数名、参数个数、参数类型以及参数的顺序都必须与基类中的原型)完全相同。

一般虚函数的定义语法是：

```
virtual 函数类型 函数名(形参表)
{
    函数体;
}
```

2) 虚函数的限制

- (1) 只有成员函数才能声明为虚函数。
- (2) 虚函数必须是非静态成员函数。
- (3) 内联函数不能声明为虚函数。
- (4) 构造函数不能声明为虚函数。
- (5) 析构函数可以声明为虚函数。

其中，虚析构函数的声明语法为：

```
virtual ~类名
{
    析构函数体;
}
```

4. 抽象类

抽象类是带有纯虚函数的类。主要作用是通过它为一个类族建立一个公共的接口，使它们能够更有效地发挥多态特性。其目的就是为了通过它多态地使用各类中的成员函数。

(1) 纯虚函数。一个抽象类至少带有一个纯虚函数。纯虚函数是一个在基类中说明的虚函数，它在该基类中没有定义具体的操作内容，要求各派生类根据实际需要定义自己的实现内容。

纯虚函数的声明格式为

virtual 函数类型 函数名(参数表)=0

(2) 使用抽象类时需注意以下几点：

① 抽象类只能用于其他类的基类，不能建立抽象类对象。抽象类处于继承层次结构的较上层，一个抽象类自身无法实例化，而只能通过继承机制生成抽象类的非抽象派生类，然后再实例化。

② 抽象类不能用作参数类型、函数返回值或显式转换的类型。

③ 可以声明一个抽象类的指针和引用。通过指针或引用，就可以指向并访问派生类对象，也可以访问派生类的成员。

8.1.3 学习要求

- 了解静态联编和动态联编的异同之处。
- 了解多态性实现的方法。
- 熟练掌握常用运算符重载的方法，包括用成员函数重载和用友元函数重载。
- 掌握虚函数和纯虚函数的概念。
- 掌握抽象类的概念及使用方法。

8.2 典型例题（或典型算法）分析

8.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

1. 程序代码如下：

```
#include <iostream.h>
class complex
{
private:
    double real;
    double imag;
public:
```

```
    complex(double r=0.0,double i=0.0){real=r;imag=i;}
    complex operator + (complex c2);
    complex operator - (complex c2);
    void display( );
};
complex complex::operator + (complex c2)
{
    complex c=(real+c2.real,imag+c2.imag);
    return c;
}
complex complex::operator - (complex c2)
{
    return  complex(real-c2.real,imag-c2.imag);
}
void complex::display( )
{
    cout<<"(" <<real<<","<<imag<<")"<<endl;
}
void main( )
{
    complex c1(5,4),c2(2,10),c3,c4;
    cout<<"c1=";
    c1.display( );
    cout<<"c2=";
    c2.display( );
    c3=c1-c2;
    cout<<"c3=c1-c2=";
    c3.display( );
    c4=c1+c2;
    cout<<"c4=c1+c2=";
    c4.display( );
}
```

运行结果为:

c1=(5,4)

c2=(2,10)

c3=c1-c2=(3,-6)

c4=c1+c2=(7,14)

分析 这是一个双目运算符重载为成员函数的程序。复数的加减法所遵循的规则是实部和虚部分别相加减，运算符的两个操作数都是复数类对象。因此，可以把“+”、“-”运算重载为复数类的成员函数，重载函数只有一个形参，类型也是复数类对象。

从本例可以看出，除了在函数声明及实现的时候使用了关键字 `operator` 之外，运算符重载成员函数与类的普通成员函数没有区别。在使用的时候，可以直接通过运算符、操作数的方式来完成函数调用。这时，运算符“+”、“-”原有的功能都不改变，对整型数、浮点数等基本类型数据的运算仍然遵循 C++ 预定义的规则，同时，添加了新的针对复数运算的功能。“+”运算符作用于不同的对象就会导致完全不同的操作行为，具有更广泛的多态特征。

2. 程序代码如下：

```
#include <iostream.h>
template <class T>
class Sample
{
    T n;
public:
    Sample(T i) { n=i; }
    void operator++( );
    void disp( )
    {
        cout<<"n="<<n<<endl;
    }
};
template <class T>
void Sample<T>::operator++( )
{
    n+=1; //不能用 n++的形式，因为对 double 型会出现问题
}
void main( )
{
    Sample<char> s('a');
    Sample<double> t(96.87);
    s++;t++;
    s.disp( );
    t.disp( );
}
```

运行结果为：

n=b

n=97.87

分析 这是一个单目运算符“++”重载为模板类成员函数的程序。其中要注意“n+=1”不能写成n++的形式，因为n++的形式对double类型的数据会出现问题。

3. 程序代码如下：

```
#include<iostream.h>
class A
{
public:
    virtual void disp( ){cout<<"class A!"<<endl;}
};
class B:public A
{
public:
    void disp( ){cout<<"class B!"<<endl;}
};
class C:public A
{
public:
    void disp( ){cout<<"class C!"<<endl;}
};
void main( )
{
    A a,*ptr;
    B b;
    C c;
    a.disp( );
    b.disp( );
    c.disp( );
    ptr=&b;
    ptr->disp( );
    ptr=&c;
    ptr->disp( );
}
```

运行结果为：

```
class A!
class B!
class C!
class B!
class C!
```


分析 这是一个比较简单的有关虚函数的程序。在基类 A 中对 `void disp()` 进行了虚函数声明 `virtual`, 在其派生类中可以重新定义它。在派生类 B 和 C 中分别重新定义了 `void disp()` 函数。在 `void disp()` 函数被重新定义时, 其函数的原型与基类中的函数原型必须完全相同。在 `main()` 函数中, 分别定义了 A、B 和 C 类的三个对象, 并且还定义了一个指向基类 A 的指针。在执行过程中, 当采用“对象名.函数名”的方式调用函数 `disp()` 时, 分别调用 A、B 和 C 类中定义的 `disp()` 函数。当指向基类 A 的指针指向不同的派生类的对象时, 虽然都是 `ptr->disp()` 语句, 但 `ptr->disp()` 却能调用不同的版本。由此可见, 使用虚函数可充分体现多态性。

4. 程序代码如下:

```
#include<iostream.h>
class B0
{
public:
    virtual void display()=0;
};
class B1:public B0
{
public:
    void display(){cout<<"B1::display()"<<endl;}
};
class D1:public B1
{
public:
    void display(){cout<<"D1::display()"<<endl;}
};
void fun(B0 *ptr)
{
    ptr->display();
}
void main()
{
    B0 *p;
    B1 b1;
    D1 d1;
    p=&b1;
    fun(p);
    p=&d1;
    fun(p);
}
```

运行结果为:

```
B1::display()
```

```
D1::display()
```

分析 这是一个关于抽象类的程序。在基类 B0 中将成员函数 display() 声明为纯虚函数, 这样, 基类 B0 就是一个抽象类, 我们无法声明 B0 类的对象, 但是可以声明 B0 类的指针和引用。B0 类经过公有派生产生了 B1 类, B1 类作为新的基类又派生出 D1 类。若使用抽象类 B0 类型的指针, 则当它指向某个派生类的对象时, 就可以通过它访问该对象的虚成员函数。程序中类 B0、类 B1 和类 D1 属于同一个类族, 抽象类 B0 通过纯虚函数为整个类族提供了通用的外部接口语义。通过公有派生而来的子类给出了纯虚函数的具体函数实现, 因此是非抽象类。可以定义派生类的对象, 同时, 根据赋值兼容规则, 抽象类 B0 类型的指针也可以指向任何一个派生类的对象, 通过基类 B0 的指针可以访问到正在指向的派生类 B1 和 D1 类对象的成员, 这样就实现了对同一类族中的对象进行统一的多态处理。

同时, 程序中派生类的虚函数并没有用关键字 virtual 显式说明 (因为它们与基类的纯虚函数具有相同的名称、参数及返回值), 而是由系统自动默认其为虚函数。

8.2.2 程序设计与算法分析

1. 重载下标运算符“[]”。

分析 下标运算符“[]”通常用于取数组中的某个元素, 通过下标运算符重载, 可以实现数组下标的越界检测等。下标运算符“[]”的重载关键是将下标值作为一个操作数, 它的实现其实非常简单, 就是用字符指针的首地址加下标值, 然后将相加后的地址返回, 这样就实现了读取数组中某个元素的作用。

```
#include<iostream.h>
#include<string.h>
class word
{
private:
    char *str;
public:
    word(char *s)
    {
        str=new char[strlen(s)+1];
        strcpy(str,s);
    }
    char &operator [] (int k)
    {
        return *(str+k);
    }
}
```

```

void disp( )
{
    cout<<str<<endl;
}
};
void main( )
{
    char *s="china";
    word w(s);
    w.disp( );
    int n=strlen(s);
    while(n>=0)
    {
        w[n-1]=w[n-1]-32;
        n--;
    }
    w.disp( );
}

```

运行结果为:

```

china
CHINA

```

2. 重载运算符 new 和 delete。

分析 通过重载 new 和 delete, 可以克服 new 和 delete 的不足, 使其按要求完成对内存的管理。这是一个很简单的重载 new 和 delete 的程序, 其中, new 通过 malloc() 函数实现, 其操作数是申请内存单元的字节个数; delete 通过 free() 函数实现, 其操作数是一个指针, 指针指向哪儿即告诉系统释放哪里的单元。

```

#include <iostream.h>
#include<malloc.h>
class rect
{
private:
    int length,width;
public:
    rect(int l,int w)
    {
        length=l;
        width=w;
    }
}

```

```

    }
    void *operator new(size_t size)
    {
        return malloc(size);
    }
    void operator delete(void *p)
    {
        free(p);
    }
    void disp()
    {
        cout<<"area:"<<length*width<<endl;
    }
};
void main()
{
    rect *p;
    p=new rect(5,9);
    p->disp();
    delete p;
}

```

运行结果为:

area:45

3.3 重载逗号运算符“,”。

分析 逗号运算符是一个双目运算符,和其他运算符一样,也可以通过重载逗号运算符来达到期望的结果。逗号运算符构成的表达式为“左操作数,右操作数”,该表达式返回右操作数的值。

```

#include <iostream.h>
#include<malloc.h>
class point
{
private:
    int x,y;
public:
    point(){};
    point(int xx,int yy)
    {

```

```
x=xx;
y=yy;
}
point operator , (point r)
{
    point t;
    t.x=r.x;
    t.y=r.y;
    return t;
}
point operator + (point r)
{
    point t;
    t.x=x+r.x;
    t.y=y+r.y;
    return t;
}
void disp( )
{
    cout<<"area:"<<x*y<<endl;
}
};
void main( )
{
    point p1(1,2),p2(3,4),p3(5,6);
    p1.disp( );
    p2.disp( );
    p3.disp( );
    p1=(p1,p2+p3,p3);
    p1.disp( );
}
```

运行结果为:

area:2

area:12

area:30

area:30

8.2.3 填空

1. 下列程序是一个含有比较运算符和赋值运算符重载的程序。请填空完成程序并上机运行验证。

```
#include <iostream.h>
class point
{
private:
    float x,y;
public:
    point(float xx=0, float yy=0) {x=xx;y=yy;}
    point(point &);
    ~point(){}
    bool operator == (point);
    bool operator != (point);
    point operator += (point);
    point operator -= (point);
    float get_x() {return x;}
    float get_y() {return y;}
};
point::point(point &p)
{
    _____ ① _____;
}
bool point::operator == (point p)
{
    if(_____ ② _____)
        return 1;
    else
        return 0;
}
bool point::operator != (point p)
{
    if(x!=p.get_x() && y!=p.get_y())
        return 1;
    else
        return 0;
}
point point::operator += (point p)
```

```

{
    this->x+=p.get_x( );
    this->y+=p.get_y( );
    return ③;
}
point point::operator += (point p)
{
    this->x+=p.get_x( );
    this->y+=p.get_y( );
    return *this;
}
void main( )
{
    point p1(1,2),p2(3,4),p3(5,6);
    cout<<"p1==p2? " <<(p1==p2)<<endl;
    cout<<"p1!=p2? " <<(p1!=p2)<<endl;
    p3+=p1;
    cout<<"p3+=p1,p3: " <<p3.get_x( )<<","<<p3.get_y( )<<endl;
    p3-=p1;
    cout<<"p3+=p1,p3: " <<p3.get_x( )<<","<<p3.get_y( )<<endl;
}

```

运行结果为:

```

p1==p2? 0
p1!=p2? 1
p3+=p1,p3: 6,8
p3+=p1,p3: 5,6

```

参考答案:

- ① x=p.x;y=p.y
- ② x==p.get_x()&&y==p.get_y()
- ③ *this

分析 这是一个含有比较运算符和赋值运算符重载的程序。从程序可以看出,这两种运算符的重载很容易实现,其中①处是 point 类的拷贝构造函数定义;②处是比较两个点是否相等,故可知②处应填 `x==p.get_x()&&y==p.get_y()`,表示 x 值和 y 值都相等时,返回 1,否则返回 0;③处是返回 this 指针的内容。

2. 下列程序中声明一个哺乳动物 Mammal 类,再由此派生出 Dog 类,二者都定义 Speak() 成员函数,基类中定义 Speak() 为虚函数。主程序中分别声明一个 Mammal 类和 Dog 类的对象,再分别用“对象名.函数名”和指针的形式调用 Speak() 函数。请填空完成程序并上机运行验证。

```
#include<iostream.h>
class Mammal
{
public:
    _____①_____ void Speak( ){cout<<"This is a Mammal!"<<endl;}
};
class Dog:public Mammal
{
public:
    void Speak( ){cout<<"This is a dog!"<<endl;}
};
void main( )
{
    Mammal m,_____②_____;
    Dog d;
    m.Speak( );
    d.Speak( );
    ptr=&m;
    ptr->Speak( );
    _____③_____;
    ptr->Speak( );
}
```

参考答案:

① virtual

② *ptr

③ ptr=&d

分析 这是一个含有虚函数的程序。首先要注意的是虚函数必须在基类中定义,因此①处应填 virtual;其次在用指针对基类和派生类对象操作时,指针应定义为基类的指针,如果定义为派生类的指针,则该指针只能指向派生类的对象,而不能指向基类对象,因此②处应填*ptr,定义一个基类指针;③处填入 ptr=&d,是让指针指向派生类对象。

8.3 上机实验指导

8.3.1 实验目的

- (1) 学习运算符重载的方法。
- (2) 学习虚函数的使用方法。
- (3) 学习抽象类的使用方法。

8.3.2 实验内容与补充习题

1. 用运算符重载为友元函数的方法重作复数加、减法运算。请填空完成程序并上机运行验证。(注意: 这时必须把操作数全部通过形参的方式传递给运算符重载函数。)

```
#include <iostream.h>
.
class complex
{
private:
    double real;
    double imag;
public:
    complex(double r=0.0,double i=0.0){_____①_____;}
    friend complex operator + (complex c1,complex c2);
    friend complex operator - (complex c1,complex c2);
    void display( );
};

complex operator + (complex c1,complex c2)
{
    return _____②_____;
}

complex operator - (complex c1,complex c2)
{
    return _____③_____;
}

void complex::display( )
{
    cout<<"(" <<real<<"," <<imag<<")"<<endl;
}

void main( )
{
    complex c1(5,4),c2(2,10),c3,c4;
    cout<<"c1=";
    c1.display( );
    cout<<"c2=";
    c2.display( );
    c3=c1 + c2;
    cout<<"c3=c1+c2=";
    c3.display( );
```

```
c4=c1+c2;
cout<<"c4=c1+c2=";
c4.display( );
}
```

2. 以下是一个计算正方体、球体和圆柱体的面积及体积的程序。试分析程序并写出程序的运行结果，然后上机运行验证。

```
#include<iostream.h>
class container
{
protected:
    double radius;
public:
    container(double radius)
    {container::radius=radius;}
    virtual double surface_area( )=0;
    virtual double volume( )=0;
};
class cube:public container
{
public:
    cube(double radius):container(radius){};
    double surface_area( ){return 6*radius* radius;}
    double volume( ){return radius* radius* radius;}
};
class sphere:public container
{
public:
    sphere(double radius):container(radius){};
    double surface_area( ){return 4*3.1416*radius*radius;}
    double volume( ){return 3.1416*radius*radius*radius*4/3;}
};
class cylinder:public container
{
    double height;
public:
    cylinder(double radius, double height):container(radius)
    {cylinder::height=height;}
    double surface_area( ){return 2*3.1416*radius*(radius+height);}
```

```

        double volume( ){return 3.1416*radius*radius*height;}
    };
void main( )
{
    container *ptr;
    cube obj1(5);
    sphere obj2(6);
    cylinder obj3(3,4);
    ptr=&obj1;
    cout<<"the surface area of cube is :"<<ptr->surface_area( )<<endl;
    cout<<"the volume of cube is :"<<ptr->volume( )<<endl<<endl;
    ptr=&obj2;
    cout<<"the surface area of sphere is :"<<ptr->surface_area( )<<endl;
    cout<<"the volume of sphere is :"<<ptr->volume( )<<endl<<endl;
    ptr=&obj3;
    cout<<"the surface area of cylinder is :"<<ptr->surface_area( )<<endl;
    cout<<"the volume of cylinder is :"<<ptr->volume( )<<endl;
}

```

运行结果为:

```

_____
_____
_____
_____
_____
_____

```

3. 下列是一个重载 new 和 delete 的程序, 试分析程序并写出程序的运行结果, 然后上机运行验证。

```

#include<iostream.h>
#include<stdio.h>
const maxpoints=512;
static struct block
{
    int x,y;
    block * next;
}block1[maxpoints];
class point
{

```

```

    int x,y;
    static block * freelist;
    static int used;
public:
    point(int vx=0,int vy=0){x=vx,y=vy;}
    void * operator new(size_t);
    void operator delete(void *ptr);
    void print( ){cout<<x<<" "<<y<<"\n";}
};
void *point::operator new(size_t size)
{
    block *res=freelist;
    if(used<maxpoints)
        res=&block l[used++];
    else if(res!=0)
        freelist=freelist->next;
    return res;
}
void point::operator delete(void *ptr)
{
    ((block *)ptr)->next=freelist;
    freelist=(block *)ptr;
}
block * point::freelist=0;
int point::used=0;
main( )
{
    point *pt=new point(5,7);
    pt->print( );
    return 1;
}

```

运行结果为：

4. 编写程序声明一个三维点 Point 类，重载运算符“+”、“-”和“=”，实现三维点的加、减和赋值。

5. 声明一个 Shape 抽象类，在此基础上派生出 Rectangle 和 Circle 类，二者都由 GetArea() 函数计算对象的面积，由 GetPerim() 函数计算对象的周长。

*6. 编写一个程序，先设计一个链表 `list` 类，再从链表类派生出一个集合类 `set`，在集合类中添加一个记录元素个数的数据项。要求可以实现对集合的插入、删除、查找和显示。

7. 编写一个程序，声明一个矩阵类，通过重载 “+”、“-” 和 “”，实现矩阵的相加、相减和相乘。

第9章 流类库与输入/输出

9.1 基本知识点、内容概要与学习要求

9.1.1 基本知识点

1. I/O 流的概念
2. 磁盘文件的输入/输出
 - 文本文件的读写。
 - 二进制文件的读写。
3. 格式化输入/输出
 - ios 标志位的使用。
 - 输入输出成员函数的使用。
 - 控制符的使用。
4. 输入/输出运算符的重载
 - 插入符的重载。
 - 提取符的重载。

9.1.2 内容概要

1. I/O 流的概念

C++中把数据之间的传输操作称作流。流既可以表示数据从某个载体或设备传送到内存缓冲区变量中(即输入流),也可以表示数据从内存传送到某个载体或设备中(即输出流)。

使用流以后,程序用流统一对各种计算机设备和文件进行操作,使程序与设备、程序与文件无关,提高了程序设计的通用性和灵活性。也就是说,无论与流相联系的实际物理设备差别有多大,流都采用相同的方式运行,这种机制使得流可以跨越物理设备平台,实现透明运作。例如,往显示器上输出字符和向磁盘文件或打印机输出字符,尽管接收输出的物理设备不同,但具体操作过程是相同的。

2. 磁盘文件的输入/输出

在对文件进行 I/O 操作时,要先进行打开操作,使流和文件发生联系,建立联系后的文件才允许数据流入或流出。输入或输出结束后,使用关闭操作使文件与流断开联系。

1) 流对象的声明

流可以分为三类：输入流、输出流以及输入/输出流。相应地，必须将流说明为 `ifstream`、`ofstream` 以及 `fstream` 类的对象。

2) 文件的打开与关闭

说明了流对象之后，可使用函数 `open()` 打开文件。文件的打开即在流与文件之间建立一个联系。

3) 文件的读写

通过打开文件可建立 I/O 流与文本文件的联系，之后便可以进行文件的读写操作。文件读写方法主要有使用流运算符直接读写和使用流成员函数等。

3. 格式化输入/输出

1) 使用 `ios` 成员函数

每一个 C++ 流都有自己当前的数据格式控制状态，这些状态用一个长整数表示，即 `ios` 类的数据成员 `x_flags`，称为格式化标志字。这些格式化标志字在 `ios` 类中定义为公有的枚举量。

2) 使用 I/O 控制符

除 `ios` 类成员函数之外，C++ 的流类库还提供了另一种更方便的 I/O 格式化方法，这种方法使用一种称为控制符的特殊函数。控制符的特点是可以直接包含在 I/O 表达式中。

所有不带形参的控制符都定义在头文件 `iostream.h` 中，而带形参的控制符则定义在头文件 `iomanip.h` 中，因而使用相应的控制符就必须包含相应的头文件。

4. 关于文件流类

由文件流类不是标准设备，因此其没有 `cout` 那样预先定义的全局对象。文件流类支持对磁盘文件的操作。要定义一个文件流类对象，需要指定文件名和打开方式。

例如，`ofstream` 类用于执行文件输出。如果需要一个只输出的磁盘文件，就可以构造一个 `ofstream` 类的对象。在打开文件之前或之后可以指定 `ofstream` 对象接受二进制或文本模式数据。很多格式化选项和成员函数可以作用于它，包括基类 `ios` 和 `ostream` 的所有功能。

构造输出文件流的常用方法如下：

(1) 先使用默认构造函数，然后调用 `open` 成员函数，例如：

```
ofstream sFile;           //声明一个静态输出文件流对象
sFile.open("filename",iosmode); //打开文件，使对象与文件建立联系
ofstream * dFile=new ofstream; //建立一个动态的输出文件流对象
dFile->open("filename",iosmode); //打开文件，使对象与文件建立联系
```

(2) 在调用构造函数时，指定文件名和模式为

```
ofstream myFile("filename",iosmode);
```

5. 关于输入/输出运算符重载

可以重载提取运算符和插入运算符以执行自定义类型的 I/O 操作。

1) 重载“>>”运算符

在 C++ 中, “>>”运算符称为提取运算符, 对它进行重载的函数称为提取符重载函数。这个函数接收流的输入信息。其格式如下:

```
friend istream& operator >> (istream& stream,<类名>&<类引用名>)
{
    函数体;
    return stream;
}
```

该提取符重载函数是以友元方式说明的。其中第一个参数是 istream 类对象的一个引用, 即 stream 必须是一个输入流, <类引用名>接收输入对象的引用。该函数返回 istream 的一个引用 stream。<函数体>中给出实现该提取符重载的目的代码。

2) 重载“<<”运算符

在 C++ 中, “<<”运算符称为插入运算符。当重载“<<”运算符用于输出时, 相当于创建一个插入符函数。该函数的格式如下:

```
friend ostream& operator << (ostream& stream,<类名>&<类引用名>)
{
    函数体;
    return stream;
}
```

该插入符函数是以友元方式说明的。其中第一个参数是 ostream 类对象的一个引用, 即 stream 必须是一个输出流, <类引用名>接收待输出对象的引用。该函数返回 ostream 的一个引用 stream。<函数体>中给出实现该插入符重载的目的代码。

9.1.3 学习要求

- 理解输入/输出流的概念。
- 理解输入/输出流类的层次结构。
- 熟练掌握磁盘文件的输入/输出。
- 掌握使用串流对字符串的输入/输出。
- 熟练掌握输出格式的控制:
 - ① 能够用 ios 成员函数控制输出的格式;
 - ② 能够用 I/O 控制符控制输出的格式。
- 掌握常用输入输出成员函数的使用。
- 理解如何输入/输出用户自定义类型的对象。

9.2 典型例题(或典型算法)分析

9.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include<iostream.h>
   #include<fstream.h>
   void main( )
   {
       char ch;
       fstream out;
       out.open("text.txt",ios::in|ios::out);
       out<<"abcdefg";
       out.seekp(0); //定位于文件头
       out.get(ch);
       while(ch!=EOF)
       {
           cout<<ch;
           out.get(ch);
       }
       cout<<endl;
   }
```

运行结果为:

abcdefg

分析 该程序首先建立一个输入输出文件流对象 out, 打开文件 text.txt, 并向其中写入 "abcdefg", 定位于文件头。然后从该文件中读出字符并显示。

```
2. #include<iostream.h>
   class Sample
   {
       int x,y;
   public:
       Sample(int a,int b){x=a;y=b;}
       friend ostream& operator<<(ostream& out,Sample& s);
   };
```

```
ostream& operator<<(ostream& out,Sample& s)
{
    cout<<"x="<<s.x<<"y="<<s.y<<endl;
    return out;
}
void main( )
{
    Sample c1(10,20);
    cout<<c1;
}
```

运行结果为:

x=10,y=20

分析 该程序采用“<<”运算符重载函数设计方法,在 main()中执行“cout<<c1;”语句时(注意:c1是sample类对象),自动调用“<<”重载运算符函数,而不是普通的“<<”运算符功能。其中,设计“<<”运算符重载函数的格式如下:

```
ostream & operator<<(ostream & out,<类名>&<对象名>)
{
    //放置代码
    return out;
}
```

9.2.2 程序设计与算法分析

1. 编写程序,实现将一个文本文件复制到另一个文件中。

分析 该题可以用多种方法实现,首先可以一个字符一个字符的进行复制。程序代码如下:

```
#include<iostream.h>
#include<fstream.h>
void main( )
{
    fstream file1,file2;
    char fn1[10],fn2[10],ch;
    cout<<"输入源文件名: ";
    cin>>fn1;
    cout<<"输入目标文件名: ";
    cin>>fn2;
    file1.open(fn1,ios::in);
    file2.open(fn2,ios::out);
```

```
while((ch=file1.get( ))!=EOF)
{
    cout<<ch;
    file2.put(ch);
}
file1.close( );
file2.close( );
}
```

该程序以一个字符一个字符的形式实现文件的复制，其关键实现语句为 while 循环。另外还可以一行字符一行字符的进行复制。程序代码如下：

```
#include<iostream.h>
#include<fstream.h>
void main( )
{
    fstream file1,file2;
    char fn1[10],fn2[10],buf[100];
    cout<<"输入源文件名： ";
    cin>>fn1;
    cout<<"输入目标文件名： ";
    cin>>fn2;
    file1.open(fn1,ios::in);
    file2.open(fn2,ios::out);
    while(!file1.EOF( ))
    {
        file1.getline(buf,100);
        file2<<buf;
        file2<<"\n";
    }
    file1.close( );
    file2.close( );
}
```

该段程序以一行字符一行字符的形式实现文件的复制，该方法首先开辟了一段内存缓冲区 buf，将从源文件中读出的内容先放在 buf 中，再从 buf 输出到目标文件中去。注意：在 buf 之后没有“\n”，要专门添加。当然我们还可以使用下面的方法实现一行一行的复制：

```
#include<iostream.h>
#include<fstream.h>
void main( )
```

```
{
    fstream file1,file2;
    char fn1[10],fn2[10],buf[100];
    cout<<"输入源文件名: ";
    cin>>fn1;
    cout<<"输入目标文件名: ";
    cin>>fn2;
    file1.open(fn1,ios::in);
    file2.open(fn2,ios::out);
    while(!file1.EOF())
    {
        file1.getline(buf,100);
        file2.write((char *)&buf,sizeof(buf));
    }
    file1.close();
    file2.close();
}
```

该程序生成的文件是二进制格式,因为 write() 函数是按二进制格式写的,前两个程序都是文本方式。

2. 编写一个程序,实现以下功能:

- (1) 输入一系列的数据(学号、姓名、成绩)存放在文件 std.dat 中。
- (2) 从该文件中读出这些数据并显示出来。

分析 可以先设计一个 Std 类,其中包括一个学生数据的操作,然后建立两个普通函数 func1()、func2(),分别实现功能(1)和(2)。程序代码如下:

```
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
class Std
{
    int no;
    char name[10];
    int degree;
public:
    void gdata()
    {
        cout<<"(学号 姓名 成绩): ";
        cin>>no>>name>>degree;
    }
}
```

```
void display( )
{
    cout<<setw(6)<<no<<setw(10)<<name
        <<setw(6)<<degree<<endl;
}
};

void func1( )
{
    ofstream out;
    out.open("std.dat");
    Std s;
    int n;
    cout<<"输入数据"<<endl;
    cout<<"  学生人数:  ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cout<<"  第"<<i+1<<"个学生:";
        s.gdata( );
        out.write((char *)&s,sizeof(s));
    };
    out.close( );
}

void func2( )
{
    ifstream in;
    in.open("std.dat");
    Std s;
    cout<<"输入数据"<<endl;
    cout<<"  学号  姓名  成绩"<<endl;
    in.read((char *)&s,sizeof(s));
    while(in)
    {
        s.display( );
        in.read((char *)&s,sizeof(s));
    }
    in.close( );
}

void main( )
```

```

{
    int n;
    do
    {
        cout<<"选择(1: 输入数据 2: 输出数据 其他: 退出): ";
        cin>>n;
        switch(n)
        {
            case 1:func1( );break;
            case 2:func2( );break;
        }
    }while(n==1||n==2);
}

```

运行结果为:

选择(1: 输入数据 2: 输出数据 其他: 退出): 1

输入数据

学生人数: 3

第 1 个学生(学号 姓名 成绩):1 wang 78

第 2 个学生(学号 姓名 成绩):2 chen 89

第 3 个学生(学号 姓名 成绩):3 li 87

选择(1: 输入数据 2: 输出数据 其他: 退出): 2

输出数据

学号	姓名	成绩
1	wang	78
2	chen	89
3	li	87

选择(1: 输入数据 2: 输出数据 其他: 退出): 3

3. 编写一个程序, 采用“<<”运算符重载函数的设计方法显示一个数组 (3×4) 中的元素。

分析 在“<<”运算符重载函数中, 可以使用标准“<<”运算符将数组的各个元素直接显示在屏幕上。程序代码如下:

```

#include<iostream.h>
#include<iomanip.h>
const int M=3;
const int N=4;
class Array
{

```

```

int A[M][N];
public:
    Array(int b[M][N])
    {
        for(int i=0;i<M;i++)
            for(int j=0;j<N;j++)
                A[i][j]=b[i][j];
    }
    friend ostream& operator<<(ostream& out,Array& a);
};
ostream& operator<<(ostream& out,Array& a)
{
    cout<<"输出结果"<<endl;
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
            cout<<setw(3)<<a.A[i][j];
        cout<<endl;
    }
    return out;
}
void main( )
{
    int a[M][N]={ {1,3,5,9},{11,13,15,17},{19,21,23,25}};
    Array A(a);
    cout<<A;
}

```

运行结果为:

```

1    3    5    9
11   13   15   17
19   21   23   25

```

9.2.3 填空

下面的程序用于统计文件 abc.txt 中的字符个数，请填空完成程序。

```

#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>

```

```
void main( )
{
    fstream file;
    file.open("abc.txt",ios::in);
    if(____①____)
    {
        cout<<"abc.txt cannot open"<<endl;
        abort( );
    }
    char ch;
    int i=0;
    while(!file.eof( ))
    {
        ____②____;
        ____③____;
    }
    cout<<"文件字符个数: "<<i<<endl;
    ____④____;
}
```

参考答案:

- ① !file
- ② file.get(ch)
- ③ i++
- ④ file.close

分析 该程序首先建立一个输入/输出文件流类的对象 file，并通过该对象调用 open() 成员函数打开文件 abc.txt，程序中的 if 语句对文件打开的成功与否进行验证，即条件!file 是否成立，而 while 语句是具体对文件中的字符进行逐个的统计。在程序最后，还要关闭该文件。

9.3 上机实验指导

9.3.1 实验目的

- (1) 学习 C++ I/O 流类的操作。
- (2) 学习文件作为一种流的操作。

9.3.2 实验内容与补充习题

1. 分析以下程序的运行结果并上机验证。

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
void main( )
{
    fstream file;
    file.open("text.dat",ios::in|ios::out);
    if(!file)
    {
        cout<<"text.dat cannot open"<<endl;
        abort( );
    }

    char textline[]="123456789\nabcdefghi\0";
    for(int i=0;i<sizeof(textline);i++)
        file.put(textline[i]);
    file.seekg(0);
    char ch;
    while(file.get(ch))
        cout<<ch;
    file.close( );
}
```

2. 分析以下程序的运行结果并上机验证。

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
void main( )
{
    fstream outfile,infile;
    outfile.open("text.dat",ios::out);
    if(!outfile)
    {
        cout<<"text.dat cannot open!"<<endl;
        abort( );
    }
}
```

```

outfile<<"1234567890"<<endl;
outfile<<"aaaaaaaaa"<<endl;
outfile<<"*****"<<endl;
outfile.close( );
infile.open("text.dat",ios::in);
if(!infile)
{
    cout<<"text.dat cannot open"<<endl;
    abort( );
}
char textline[80];
int i=0;
while (!infile.eof( ))
{
    i++;
    infile.getline(textline,sizeof(textline));
    cout<<i<<": "<<textline<<endl;
}
infile.close( );
}

```

3. 分析以下程序的运行结果并上机验证。

```

#include<iostream.h>
class Sample
{
    int x,y;
public:
    Sample( ){ }
    friend istream& operator>>(istream&,Sample&);
    void display( )
    {
        cout<<"x="<<x<<"y="<<y<<endl;
    }
};
istream& operator>>(istream& input,Sample& a)
{
    input>>a.x>>a.y;
    return input;
}

```

```
void main( )
{
    Sample obj;
    cin>>obj;
    obj.display( );
}
```

4. 下面是一个向一个文件中写入一些内容,然后再把该内容显示出来的程序。请填空完成该程序。

```
#include<iostream.h>
#include<fstream.h>
void main( )
{
    char str[100];
    fstream out;
    _____ ① _____
    out<<"abcdefg";
    out.put('\n');
    out<<"1234567";
    out.seekg(0);
    while ( _____ ② _____ )
    {
        out.getline(str,100);
        _____ ③ _____;
    }
    cout<<endl;
}
```

5. 设计程序将数据存入磁盘文件。

基本要求: 将数 0~255, 按十进制、八进制和十六进制的表示形式(数与数之间相隔 5 个字符)一一输出到文件 data.dat 中。

将下列格式化的数据输出到文件 data.dat 中。

名字	年龄	编号	工资
li	29	111-22-333	330.00
chen	32	333-22-111	425.00
peng	41	11-22-0000	650.00

建立数据结构:

```
struct
{
    char name[10];
```

```
int age;  
char snum[20];  
float wage;  
}employees[3];
```

输出到文件中的例子：

```
ofstream out;  
out.open("data.dat");  
out<<setiosflags(ios::left)<<setw(10)<<employees[0].name;  
out<<setw(8)<<employees[0].age;  
out<<setw(20)<<employees[0].snum;  
out<<setiosflags(ios::showpoint|ios::fixed);  
out<<setprecision(2)<<setw(8)<<employees[0].wage<<endl;
```

回答下列问题：

- (1) 如果文件在打开时出错怎么办？
- (2) 如果要读取这个文件，应该怎么做？
- (3) 如果 `employees` 结构数组是一个包含自己的输出处理成员函数的类，那么原来的程序将做怎样的修改？
6. 编写程序，使之能够一屏一屏简单地浏览指定的文本文件。
7. 编写程序，使之能统计文件 `text.txt` 的字符个数。
8. 编写程序，使之能统计文件 `text.txt` 的行数。
9. 编写程序，使之能给一个文本文件加上行号后存储到另一个文件中。
10. 编写程序，使之能输入一系列的数据（学号、姓名、成绩）存放在文件 `std.dat` 中（只显示其中 80~89 分数段的学生数据）。
11. 设计一个点类 `Point`，包括 `x`、`y`、`z` 三个私有数据成员，通过重载“<<”和“>>”运算符实现数据的输入输出。
12. 编写程序，使之能通过重载“<<”运算符将 `Std` 类的对象数据存放在文件 `std.dat` 中，然后在屏幕上显示这些数据。
13. 编写程序，实现在二进制文件 `data.dat` 中写入 3 个记录，显示其内容，然后删除第 2 个记录，而且要显示删除记录后的文件内容。

第10章 异常处理

10.1 基本知识点、内容概要与学习要求

10.1.1 基本知识点

1. 异常处理机制
2. 异常处理的实现
 - 异常处理的语法。
 - 异常处理的执行过程。
3. 异常处理中的构造函数与析构函数

10.1.2 内容概要

1. 异常处理机制

C++语言异常处理机制的基本思想是将异常的检测与处理分离。当在一个函数体中检测到异常条件存在，但却无法确定相应的处理方法时，该函数将引发一个异常，由函数的直接或间接调用者捕获这个异常并处理这个错误。如果程序始终没有处理这个异常，最终它会被传到C++运行系统那里，运行系统捕获异常后，通常只是简单地终止这个程序。

由于异常处理机制使得异常的引发和处理不必在同一函数中，因此底层的函数可以着重解决具体问题而不必过多地考虑对异常的处理。上层调用者可以在适当的位置设计对不同类型异常的处理。

2. 异常处理的语句

(1) throw 语句格式如下：

throw <表达式>;

当某段程序发现了自己不能处理的异常，就可以使用 throw 语句将这个异常抛掷给调用者。

(2) try 语句格式如下：

```
try
{
    try 语句块;    //一般为复合语句
}
```

try 语句后的复合语句是代码的保护段。如果预料某段程序代码(或对某个函数的调用)有可能发生异常,就将其放在 try 语句之后。如果这段代码(或被调函数)运行时真的遇到异常情况,其中的 throw 表达式就会抛掷这个异常。

(3) catch 语句格式如下:

```
catch(异常类型 1 参数 1)
{
    针对异常类型 1 的处理语句;
}
catch(异常类型 2 参数 2)
{
    针对异常类型 2 的处理语句;
}
:
catch(异常类型 n 参数 n)
{
    针对异常类型 n 的处理语句;
}
```

catch 语句后的复合语句是异常处理程序,捕获由 throw 表达式抛掷的异常。异常类型声明部分指明语句所处理的异常类型,它与函数的形参相类似,可以是某个类型的值,也可以是引用。这里的类型可以是任何有效的数据类型,包括 C++ 的类。当异常被抛掷以后,catch 语句便依次被检查,若某个 catch 语句的异常类型声明与被抛掷的异常类型一致,则执行该段异常处理程序。如果异常类型声明是一个省略号(...),catch 语句便处理任何类型的异常,但这段处理程序必须是 try 块的最后一段处理程序。

3. 异常处理的执行过程

(1) 控制通过正常的顺序执行到达 try 语句,然后执行 try 块内的保护段。

(2) 如果在保护段执行期间没有引起异常,那么跟在 try 块后的 catch 语句就不执行,程序从异常被抛掷的 try 块后跟随的最后一个 catch 语句后面的语句继续执行下去。

(3) 如果在保护段执行期间或在保护段调用的任何函数中有异常被抛掷,则从通过 throw 创建的对象中创建一个异常对象。

(4) 如果匹配的处理程序未找到,则 terminate() 将被自动调用,而函数 terminate() 的默认功能是调用 abort 终止程序。

(5) 如果找到了一个匹配的 catch 处理程序,且它通过值进行捕获,则其形参通过拷贝异常对象进行初始化。如果它通过引用进行捕获,则参量被初始化为指向异常对象。在形参被初始化之后,“循环展开栈”的过程开始,这包括对那些在与 catch 处理器相对应的 try 块开始和异常丢弃地点之间创建的(但尚未析构的)所有自动对象的析构。析构与构造相反的顺序进行,然后 catch 处理程序被执行,接下来程序跳转到跟随在处理程序之后的语句。

4. 异常处理中的构造与析构

C++ 异常处理在异常抛掷前为构造的所有局部对象自动调用析构函数。在程序中,找到

一个匹配的 catch 异常处理后, 如果 catch 语句的异常类型声明是一个值参数, 则其初始化方式是复制被抛掷的异常对象。如果 catch 语句的异常类型声明是一个引用, 则其初始化方式是使该引用指向异常对象。

当 catch 语句的异常类型声明参数被初始化后, 栈的展开过程便开始了。这包括从对应的 try 块开始到异常被抛掷处之间对构造(且尚未析构)的所有自动对象进行析构。析构的顺序与构造的顺序相反, 然后程序从最后一个 catch 处理之后开始恢复执行。

10.1.3 学习要求

- 了解异常处理机制。
- 掌握异常处理的语法的使用方法。
- 了解异常处理的执行过程。
- 了解异常处理中的构造与析构。

10.2 典型例题 (或典型算法) 分析

10.2.1 分析程序运行结果

分析下列程序运行后的输出结果并上机验证。

```
1. #include <iostream.h>
   #include <math.h>
   void Root(int a,int b,int c);
   void main( )
   {
       int a,b,c;
       try
       {
           cout<<"请输入一元二次方程的系数(a b c): ";
           cin>>a>>b>>c;
           Root(a,b,c);
       }
       catch(int)
       {
           cout<<"此方程无实数解!"<<endl;
       }
   }
   void Root(int a,int b,int c)
   {
```

```

double x1,x2;
int D;
D=b*b-4*a*c;
if(D<0)
    throw 0;
x1=(-b+sqrt(D))/(2*a);
x2=(-b-sqrt(D))/(2*a);
cout<<"此方程的根为: ";
cout<<"    x1="<<x1<<" , x2="<<x2<<endl;
}

```

程序试运行结果为:

- (1) 请输入一元二次方程的系数(a b c): 1 -5 4
此方程的根为: x1=4, x2=1
- (2) 请输入一元二次方程的系数(a b c): 1 2 3 4
此方程无实数解!

分析 这是一个判断一元二次方程 $ax^2 + bx + c = 0 (a \neq 0)$ 有无实数根的程序。当判别式 $d = b^2 - 4ac$ 小于零时, 求根函数 “void Root(int a,int b,int c)” 将抛掷一个异常信息 “throw 0”, 异常捕获处理函数 “catch(int)” 捕获此异常信息后, 将输出处理信息: 方程无实数解!。

2. #include <iostream.h>

```

int a[5]={1,2,3,4,5};
int fun(int);
void main( )
{
    for(int i=0;i<=5;i++)
    {
        try { cout<<"a["<<i<<"]="<<fun(i)<<endl; }
        catch(char *)
        {
            cout<<"数组下标超界错误!"<<endl;
        }
    }
}

int fun(int x)
{
    if(x>=5)
        throw "error";
    return a[x];
}

```



```
}
```

运行结果为:

```
a[0]=1
```

```
a[1]=2
```

```
a[2]=3
```

```
a[3]=4
```

```
a[4]=5
```

数组下标超界错误!

分析 这是一个防止数组下标超界的程序。在主函数 `main()` 中, 通过循环结构“`for(int i=0;i<=5;i++)`”输出全局数组 `a[5]` 时, 为防止数组下标超界, 在返回数组元素的被调用函数 `fun(i)` 中, 设置了异常信息抛掷语句“`if(x>=5) throw "error";`”, 即当下标参数 `x` 的值超过 4 时, 抛掷一字符型异常信息“`error`”, 异常捕获处理函数“`catch(char *)`”捕获到此异常信息后, 将输出反馈信息“数组下标超界错误!”。

10.2.2 程序设计与算法分析

1. 设计一个程序, 采用异常处理的方法, 在输入学生类对象的数据时检验成绩输入是否正确。

分析 根据题目设计要求, 应先定义一个相应的类结构, 其中除包含表示学生信息的数据成员外, 还应设计一个成员函数, 用于输入学生数据并且当输入的成绩大于 100 或小于 0 时抛出一个异常。程序代码如下:

```
#include <iostream.h>
#include <iomanip.h>
class Stud
{
protected:
    int no;
    char name[10];
    int score;
public:
    Stud() { };
    void getdata()
    {
        cin>>no>>name>>score;
        if(score>100||score<0)
            throw name;
    }
}
```

```

void disp( )
{
    cout<<" "<<setw(4)<<no<<setw(10)<<name<<setw(16)<<score<<endl;
}
};
void main( )
{
    Stud st[5];
    cout<<"输入学生数据: "<<endl;
    cout<<"学号, 姓名, 成绩: "<<endl;
    for(int i=0;i<5;i++)
    {
        try { st[i].getdata( ); }
        catch(char *s)
        {
            cout<<" "<<s<<"成绩输入错误! "<<endl;
        }
    }
    cout<<"输出学生数据: "<<endl;
    for(i=0;i<5;i++)
        st[i].disp( );
}

```

在成员函数 `getdata()` 中, 我们设计了一个条件异常抛掷语句 “`if(score>100||score<0) throw name;`”, 当学生成绩数据 `score` 大于 100 或小于 0 时抛出一个字符型异常信息 (该学生的姓名)。而在主函数 `main()` 中用 “`st[i].getdata()`” 为各学生对象进行数据输入时, 若某学生的成绩大于 100 或小于 0, 则其后的异常捕获处理函数 “`catch(char *s)`” 将输出信息 “XXX 成绩输入错误!” (其中 XXX 表示某学生的姓名)。

2. 编写一个程序, 设计一个 `Teacher` 类, 其中包含一个成员函数 `test()`, 它可根据职称检测年龄是否正确。其标准是: 教授年龄应大于 35 岁, 副教授年龄应大于 30 岁, 讲师年龄应大于 25 岁, 助教年龄应大于 20 岁。

分析 根据题意, 先定义一个 `Person` (一般人) 类, 然后在此类的基础上派生 `Teacher` (教师) 类, 其中应包含一个检测年龄是否正确的成员函数, 不符合假定标准时, 抛出一个异常。程序代码如下:

```

#include <iostream.h>
#include <string.h>
class Person
{
    protected:

```

```
char name[10];
int age;
public:
    Person(char na[],int a)
    {
        strcpy(name,na);
        age=a;
    }
};
class Teacher:public Person
{
    char prof[20];
public:
    Teacher(char na[],int a,char pr[]):Person(na,a)
    {
        strcpy(prof,pr);
    }
    void test( )
    {
        if(strcmp(prof,"教授")==0)
        {
            if(age<35)
                throw name;
        }
        else if(strcmp(prof,"副教授")==0)
        {
            if(age<30)
                throw name;
        }
        else if(strcmp(prof,"讲师")==0)
        {
            if(age<25)
                throw name;
        }
        else if(strcmp(prof,"助教")==0)
        {
            if(age<20)
                throw name;
        }
    }
};
```

```
    }  
    void disp( )  
    {  
        cout<<" "<<name<<"，年龄:"<<age<<"岁，职称："<<prof<<endl;  
    }  
};  
void main( )  
{  
    Teacher te[]={Teacher("李华",45,"教授"),Teacher("张明",23,"副教授"),  
                  Teacher("王军",28,"讲师")};  
    for(int i=0;i<3;i++)  
    {  
        try  
        {  
            te[i].test( );  
            te[i].disp( );  
        }  
        catch(char *s)  
        {  
            cout<<" "<<s<<"的年龄输入有误！"<<endl;  
        }  
    }  
}
```

在检测年龄是否正确的成员函数 test() 中，我们设计了一个条件异常抛掷语句“if(age < 年龄) throw name;”，当某教师的年龄 age 不符合假定职称标准时，抛出一个字符型异常信息（该教师的姓名）。而在主函数 main() 中用“te[i].test();”并根据职称对各位教师的年龄进行检测，当不符合标准时，异常捕获处理函数“catch(char *s)”将输出信息“XXX 的年龄输入有误!”（其中 XXX 表示某教师的姓名）。

运行结果为：

李华, 年龄: 45 岁, 职称: 教授

张明的年龄输入有误!

王军, 年龄: 28 岁, 职称: 讲师

10.2.3 填空

以下是一个采用异常处理的方法，能检测指定文件不存在时显示出错信息“XXX 文件不存在”（其中 XXX 表示文件名），且当文件存在时显示其大小的 C++ 程序。请填空完成程序。

```
#include <iostream.h>
#include <fstream.h>
void main( )
{
    char FileName[20];
    cout<<"请输入文件名: ";
    cin>>FileName;
    ifstream file(FileName,ios::nocreate|ios::in);
    try
    {
        if(file.fail( ))
            _____①_____ ;
    }
    catch(char *)
    {
        cout<< _____②_____ <<"文件不存在"<<endl;
        return;
    };
    file.seekg(0,ios::end);
    cout<<FileName<<"文件大小: "<<_____③_____ <<" Bytes"<<endl;
}
```

参考答案:

① throw "error" (其中 error 可为任意字符串)

② FileName

③ file.tellg()

分析 本程序除使用了 C++ 的异常处理机制外还涉及到较多的文件处理功能, 其中“if(file.fail())”用来判断文件是否存在, 当“file.fail()”为真时表示文件不存在, 故①处应填异常抛掷语句 throw "error" (其中 error 可为任意字符串); 在②处显然应填相对应的文件名, 即 FileName; ③处是当文件存在时要输出文件的大小, 因此填 file.tellg() (其中 tellg() 函数用于返回相应文件的字节数)。

10.3 上机实验指导

10.3.1 实验目的

- (1) 学习并掌握异常处理的语句功能及使用方法。
- (2) 了解异常处理的执行过程。

10.3.2 实验内容与补充习题

1. 分析并写出下列程序的运行结果，然后上机运行验证。

```
#include <iostream.h>
void Test(char *)
{
    try
    { throw "Testing throw and catch"; }
    catch(char *)
    {
        cout<<"catch a character in Test( )" <<endl;
        throw 1;
    }
    catch(int)
    { cout<<"catch a interger in Test( )" <<endl; }
}
void main( )
{
    char *c;
    c="Test";
    try
    { Test(c); }
    catch(int)
    { cout<<"catch a interger in main( )" <<endl; }
    catch(char *)
    {
        cout<<"catch a character in main( )" <<endl;
    }
}
```

运行结果为：

2. 设计一个直线类 Line (设直线方程为 $ax + bx + c = 0$)，其中包含三个数据成员 a、b、c，一个显示数据成员的 disp 成员函数和一个求两直线交点的友元函数 setpoint，要求考虑当两直线平行或交点坐标的绝对值大于等于 10^8 时，可抛出异常信息，并进行相应的处理。

3. 编写一个 C++ 程序，可演示当程序通过运算符 new 循环请求分配动态空间而产生内存溢出时的异常处理。

第 11 章 Visual C++ 环境下 Windows 程序开发概述

11.1 基本知识点、内容概要与学习要求

11.1.1 基本知识点

1. 项目和项目工作空间
2. Windows 编程基础
 - Windows API。
 - Windows 基础。
 - Windows 消息处理机制。
3. MFC 基础
 - MFC 类库。
 - MFC 应用程序框架。

11.1.2 内容概要

1. 项目和项目工作空间

Visual C++ 程序的核心是项目，而项目位于工作空间之上。每一个 Visual C++ 的工作空间中可以容纳多个项目。一个项目的开发步骤为：

- (1) 创建项目。
- (2) 使用工作空间窗口及其 ClassView、FileView 和 ResourceView 选项卡分别对项目中的类、文件和资源进行以下编辑操作：
 - 在项目中添加或删除文件；
 - 在项目中编辑源代码和资源。
- (3) 编译项目。
- (4) 纠正编译或链接中的错误。
- (5) 执行并测试可执行文件。
- (6) 调试项目。
- (7) 配置和优化代码。

2. Windows 编程基础

1) Windows API

API 是大量函数加上数字常量、宏、结构、类型以及其他相关项的集合。可以从 C++、Visual Basic、汇编语言、Fortran、Pascal 以及其他编程语言中调用这些函数。

Windows API 函数分为三类:

- 窗口管理函数: 实现窗口的创建、移动和修改功能。
- 图形设备 (GUI) 函数: 实现与设备无关的图形操作功能。
- 系统服务函数: 实现与操作系统有关的多种功能。

2) Windows 编程基础

• 事件: 当用户按下键、移动鼠标或单击鼠标按钮时, 就称为触发了一个事件, 它主要由事件的种类、发生的时间和发生的位置(如坐标值)等要素组成。Windows 系统等待用户的动作(既触发某事件)以作出相应的响应, 这种系统也称为事件驱动系统。

• 消息: 当 Windows 捕获一条事件后, 它会产生相应的消息码, 将其放入一个消息队列中, 然后由程序系统将此消息根据某种时序规则发送给相应的消息处理函数(程序)。

消息循环的形式如下:

```
While(GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

• 消息处理函数: 用于处理特定消息的一些代码。收到消息的应用程序会做些什么, 取决于应用程序本身。程序员可以编写相应的处理函数以处理消息; 如果程序员没有为该消息编写处理函数, 需要把消息传递给 Windows, 让 Windows 对消息进行默认处理。

• 绘图及设备环境: Windows 具有图形用户界面(GUI), 它包括菜单、工具栏、滚动条和对话框中的按钮等元素。这些元素都是被绘制出来的, 当窗口的任何部分被破坏或无效时, 需要重新绘制它们。在要求重新绘制时, 一般由 Windows 负责绘制窗口的框架, 而程序员负责绘制客户区。与客户区对应有一个 Windows 对象称为设备环境(device context, 通常缩写为 DC)。设备环境其实是 Windows 的一个数据结构, 包含该区域(窗口)的信息、当前背景色或区域图案、区域的无效部分等等。设备环境中最重要的对象是位图, 它是所绘图形的逻辑接口。

• 对象与句柄: 对设备环境的访问是通过 Windows 返回的句柄进行的。用于描述设备环境句柄的变量类型是 HDC。事实上, Windows 提供了多种类型的句柄, 如窗口和字体等。

• 坐标系: Windows 默认的坐标系原点在窗口客户区的左上角, x 向右为正, y 向下为正, 没有负的坐标。

3) Windows 消息处理机制

Windows 程序用一个 WinMain() 函数建立应用程序的主窗口, 再通过操作系统发送的消息来处理用户输入的操作, 程序的主窗口中需要包含处理 Windows 所发送消息的代码。

如果使用 Visual C++ 开发环境提供的向导, 只需要确定哪些消息是需要响应的, 并为之编写消息处理函数, 而不必担心消息与处理函数代码之间是如何联系起来的, 因为自动生成的应用程序框架会处理这些问题。

Windows 程序要处理的消息种类如下:

- Windows 消息: 以 WM_ 开头(WM_COMMAND 例外), 通常由窗口和视图来处理, 这些消息常常带有参数以决定处理该消息的方式。
- 由控件和其他子窗口发送给父窗口的 WM_COMMAND 消息: 当用户在编辑框中键入文本或进行修改时, 就会向系统发送一个带 EN_CHANGE 通知码的 WM_COMMAND 消息。
- 来自于用户界面对象的 WM_COMMAND 消息: 这些用户界面对象包括菜单、工具栏按钮和快捷键等。

3. MFC 基础

类库是一个可以在应用程序中使用的相互关联的 C++ 类的集合。与一般类库不同, Microsoft 基本类库(MFC 库)是一个 Windows 应用程序框架, 它定义了应用程序的结构并实现了标准的用户接口。MFC 提供了管理窗口、菜单、对话框的代码, 可实现基本的输入/输出和数据存储。

(1) MFC 类库: (略)。

(2) MFC 应用程序框架: 应用程序框架包含用于生成应用程序所必需的各种面向对象的组件的集合。

(3) MFC AppWizard 生成的应用程序包括的主要要素如下:

- WinMain 函数

Windows 要求应用程序必须有一个 WinMain 函数。

- 应用程序类 CmyFirstApp

该类的每一个对象代表一个应用程序。程序中默认定义一个全局 CMyFirstApp 对象, 即 theApp。

- 应用程序启动

启动应用程序时, Windows 调用应用程序框架内置的 WinMain 函数, WinMain 寻找由 CWinApp 派生出的全局构造的应用程序对象。

- 成员函数: CmyFirstApp::InitInstance

当 WinMain 函数找到应用程序对象时, 它调用伪成员函数 InitInstance, 这个成员函数调用所需的构造并显示应用程序的主框架窗口。

- 成员函数: CwinApp::Run

函数 Run 隐藏在基类中, 但是它发送应用程序的消息到窗口, 以保持应用程序的正常行。

- CMainFrame 类

类 CMainFrame 的对象代表应用程序的主框架窗口。

- 文档与视图类

MFC 通过“文档-视图”结构为应用程序提供一种将数据与视图相分离的存储方式。文档类的作用是将应用程序的数据保存在文档类对象中, 并从磁盘文件中读取或向磁盘文件中写入数据。视图类的作用是显示数据和编辑数据。

- 关闭应用程序

用户通过关闭主框架窗口来关闭应用程序。

11.1.3 学习要求

- 了解项目和项目工作空间的作用。
- 了解 Windows 编程的方法。
- 了解 MFC 类库及 MFC 应用程序框架。
- 掌握使用 Visual C++ 开发 Windows 程序的基本步骤。

11.2 典型例题（或典型算法）分析

（略）

11.3 上机实验指导

11.3.1 实验目的

掌握使用 Visual C++ 开发 Windows 程序的基本方法。

11.3.2 实验内容

根据下述应用程序的功能要求，设计一个基于对话框的 Windows 应用程序：

(1) 应用程序的主窗口及其布局。该应用程序的主窗口布局如图 11-1 所示，标题为 Example。

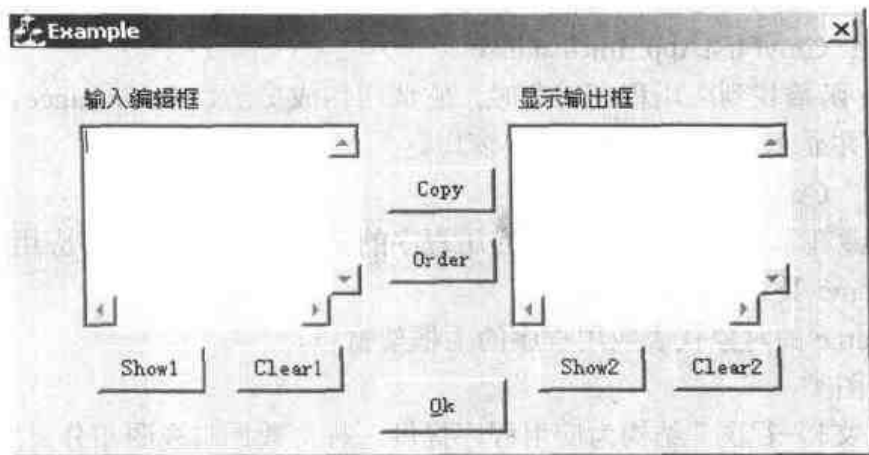


图 11-1 上机实验应用程序的主窗口布局