



# 学习成果检测——本书【练一练】答案

## 第 1 章 初识 C++——C++程序设计入门

### 一、选择题

1. D
2. D
3. C
4. C
5. C

### 二、填空题

1. C++
2. 面向过程程序设计、面向对象程序设计
3. 对象
4. 封装性、继承性、多态性
5. 面向过程、面向对象

### 三、简答题

1. C++语言起源于 C 语言，是在 C 语言的基础上增加了面向对象程序设计的要素而发展起来的。1979 年，Bjarne Stroustrup 到了 Bell 实验室，开始从事将 C 改良为带类的 C (C with classes) 的工作。1983 年将该语言正式命名为 C++。90 年代，程序员开始慢慢从 C 中淡出，转入 C++。此后，C++稳步发展，1998 年 ISO/ANSI C++标准正式制定。此后，又经过不断的改进，发展成为今天的 C++。

2. 结构化程序设计主要思想是功能分解并逐步求精。结构化程序设计方法是由 E.Dijkstra 等人于 1972 年提出来的，它建立在 Bohm、Jacopini 证明的结构定理的基础上。结构定理指出：任何程序逻辑都可以用顺序、选择和循环等三种基本结构来表示。

与结构化程序设计比较，面向对象程序设计更易于实现对现实世界的描述，因而得到了迅速发展，对整个软件开发过程产生了深刻影响。面向对象程序设计的本质是把数据和处理数据的过程看成一个整体——对象。



## 第2章 开始 C++编程之旅——Hello, C++

### 一、选择题

1. B
2. A

### 二、简答题

1. 注释用来提高程序的可读性。应在编程的过程中同时进行，注释内容应包含：源程序的总体注释(文件名、作用、创建时间、版本、作者及引用的手册、运行环境等)、函数注释(目的、算法、使用的参数和返回值的含义、对环境的一些假设等)及其他的少量注释。注释为了提高程序的可读性，它有两种形式，单行注释和多行注释。C++编译时忽略注释，也就是不参加编译。

单行注释，以“//”开头，直到该行结束。只能注释一行

//要注释的内容

多行注释，以“/\*”开头和以“\*/”作为结束，它们之间的内容都是注释内容。能注释多行。

/\*注释内容的开始

...

注释内容结束\*/

2. C++程序可以由一个或多个程序文件组成。一个程序文件往往由预处理命令、注释和函数组成。

预处理命令：在程序开始出现含有以“#”开头的命令，它们是预处理命令。

函数：C++程序是由一个主函数和若干个函数组成。

注释：注释是程序员为读者作的说明，用来提高程序的可读性的一种手段。

### 三、编程题

1. ❶ 在 Visual C++ 6.0 中，新建名为“Sum”的【Win32 Console Application】➤【A simple application】项目文件。

❷ 在工作区【FileView】视图中双击【Source Files】➤【Sum.cpp】，在代码编辑窗口中输入以下代码。

```
#include <iostream.h>
int main(int argc, char* argv[])
{
    int i,j,sum;//声明 3 个变量
```

```

cout<<"输入两个数据：";
cin>>i>>j;//输入两个数据
sum=i+j;//sum 保存两个数据的和
cout<<i<<"+"<<j<<"="<<sum<<endl;//输出结果
return 0;
}

```

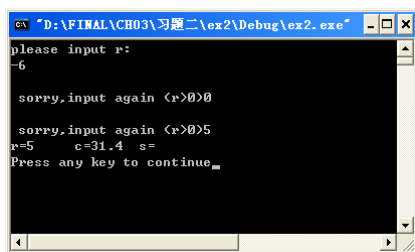
## 第3章 不变的和变的箱子——常量与变量

### 一、选择题

1. D
2. C
3. A
4. B
5. D

### 二、写出以下程序运行的结果

运行结果如下：



### 三、简答题

1. 常见的常量有数值常量、字符常量、字符串常量和符号常量 4 种。
2. 字符常量与字符串常量的区别是：
  - (1) 书写格式不同：字符常量用' '(单引号)，而字符串常量用" "(双引号)。
  - (2) 表现形式不同：字符常量是单个字符，字符串常量是一个或多个字符序列。
  - (3) 存储方式不同：字符常量占用一个字节，字符串常量占用一个以上的字节（比字符串的长度多一个）。
3. 在 C++ 中标识符用来定义变量名、函数名、类型名、类名、对象名、数组名、文件名等，

只能由字母、数字和下划线组成。且第一个字符必须是字母或下划线。

## 第4章 数据的种类——数据类型和声明

### 一、选择题

1. ACD
2. A
3. D
4. A
5. ABCD

### 二、填空题

1. 1236
2. 175
3. 96
4. 类型名，分类符
5. 初始化操作

### 三、简答题

1. 在计算机中，数据都是以二进制形式存储的，计算机仅仅能够识别二进制数据。
2. 在 C++ 中，任何运算都是在同一种数据类型的数据间进行的。不同的数据类型必须进行转化；数据类型的转化可以分为自动转化和强制转化两种形式，尤其自身的规律进行的转化为自动转化，也可以通过一定的形式进行强制的转化，在 C++ 中可以通过过 `static_cast`, `const_cast`, `dynamic_cast`, 和 `reinterpret_cast` 四个操作符进行强制的转化。

## 第5章 C++编程世界中的公式——运算符和表达式

### 一、选择题

1. C

2. C
3. C
4. A
5. D

## 二、填空题

1. 高；高
2. 整数；浮点数
3. true
4. 0
5. fabs(x-y)<1e-6

## 三、简答题

1. 自加自减运算单独成语句时，前置和后置运算没有区别；在复合语句中，前置运算先进行自加或自减运算再赋值，后置运算先赋值再进行自加或自减运算。

2.

优先级	运算符及功能说明	结合性
1	圆括号() 数组[] 成员选择 .->	从左至右
2	自增++ 自减-- 正+ 负- 取地址& 取内容* 按位求反~ 逻辑求反! 动态存储分配 new delete 强制类型转换() 类型长度 sizeof	从右至左
3	乘 * 除 / 取余数 %	从左至右
4	加 + 减 -	
5	左移位 << 右移位 >>	
6	小于 < 小于等于 <= 大于 > 大于等于 >=	
7	等于 == 不等于 !=	
8	按位与 &	
9	按位异或 ^	
10	按位或	
11	逻辑与 &&	
12	逻辑或	
13	条件表达式 ? :	从右至左
14	赋值运算符 = += -= *= /= %= &= ^=  = >>= <<= &&=   =	
15	逗号表达式 ,	

3. C++对于二元运算符"&&"和"||"可进行短路运算。由于"&&"与"||"表达式按从左到右的顺序进行计算,如果根据左边的计算结果能得到整个逻辑表达式的结果,右边的计算就不需要进行了,该规则叫短路运算。

## 第 6 章 C++程序的流程——程序控制结构和语句

### 一、选择题

1. C
2. B
3. D
4. B
5. B

### 二、填空题

1. (1, 1)(1, 2)(2, 1)(2, 2)
2. for 循环
3. continue
4. 顺序, 选择, 循环
5. 字符型、整型

### 三、简答题

1. break 语句常和 switch 语句配合使用。break 语句和 continue 语句也与循环语句配合使用, 并对循环语句的执行起着重要的作用。且 break 语句只能用在 switch 语句和循环语句中, continue 语句只能用在循环语句中。

根据程序的目的, 有时需要程序在满足另一个特定条件时立即终止循环, 程序继续执行循环体后面的语句, break 语句可实现此功能。根据程序的目的, 有时需要程序在满足另一个特定条件时跳出本次循环, continue 语句可实现该功能。continue 语句的功能与 break 语句不同, 它是结束当前循环语句的当前循环, 而执行下一次循环。在循环体中, continue 语句执行之后, 其后的语句均不再执行。

2. C++允许在 for 循环的各个位置使用几乎任何一个表达式, 但也有一条不成文的规则, 规定 for 语句的三个位置“表达式 1”用来进行设置循环初值, “表达式 2”用来设置循环条件, “表达式 3”通常用来进行循环增量。

## 第7章 程序设计的灵魂——算法与流程图

### 一、填空题

1. 确定的、有限的、次序、缺一不可的、是对问题求解过程的描述
2. 有穷性、唯一性、有输入和输出、正确性
3. 流程图、N-S图、伪代码、PAD图

### 二、简答题

1. 结构化程序设计是尽可能少用 GO TO 语句的程序设计方法。最好仅在检测出错误时才使用 GO TO 语句，而且应该总是使用前向 GO TO 语句。结构化程序的结构简单清晰，模块化强，描述方式贴近人们习惯的推理式思维方式。因此可读性强，在软件重用性、软件维护等方面都有所进步，在大型软件开发尤其是大型科学与工程运算软件的开发中发挥了重要作用。因此要提倡结构化程序设计方法。

2. 程序流程图一直是软件设计的主要工具，它的主要优点是对控制流程的描绘很直观，便于初学者掌握。

程序流程图的主要缺点如下：

程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。

程序流程图不易表示数据结构

N-S图描述算法的优点如下。

形象直观，可读性强。

限制了随意的控制转移。

强化了设计人员结构化设计方法的思维；

确保算法的设计质量。

缺点是：修改算法比较困难。

伪代码描述算法的优点是：

采用结构化语言提供的结构化控制过程；

可以方便转换为采用的计算机语言；

易于理解。

其缺点是：不如图形描述工具形象直观。

PAD图描述算法的优点是：

能展现算法的层次结构；

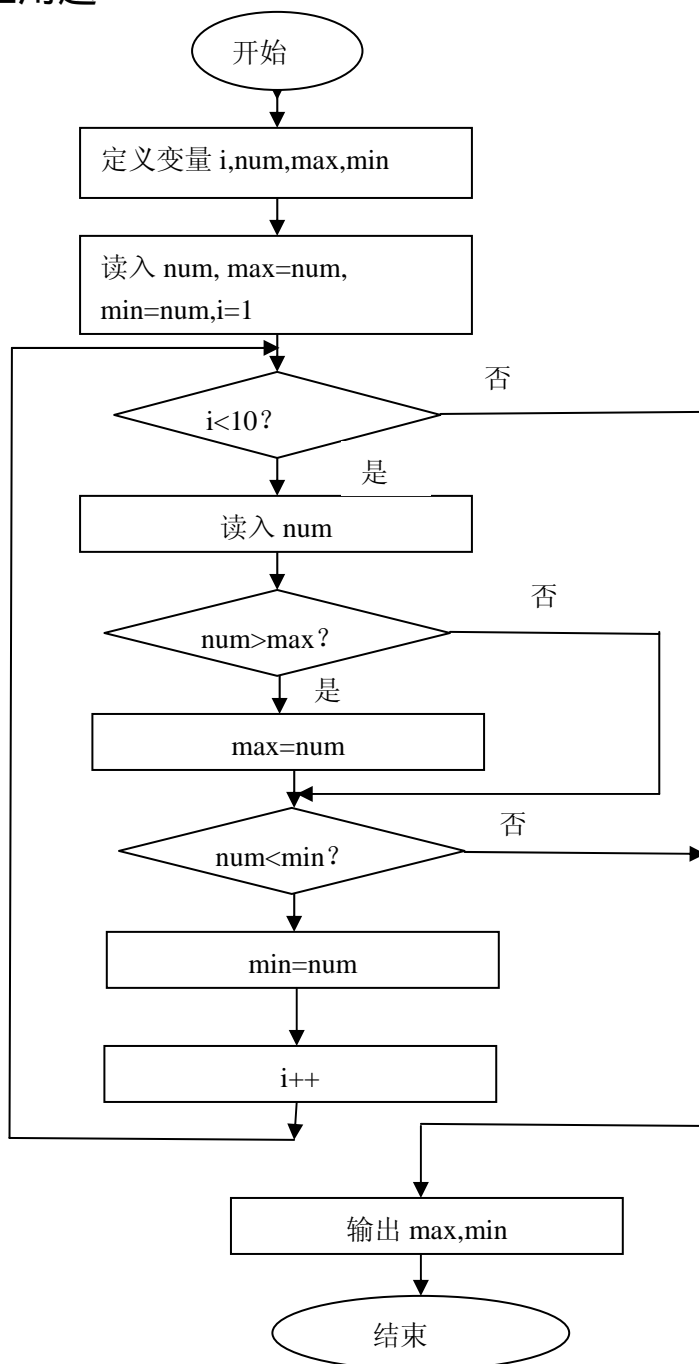
表示形式直观易懂；

即可用于表示程序逻辑，又可用于描述数据结构；

支持自顶向下，逐步求精的过程。  
缺点是不够形象直观。

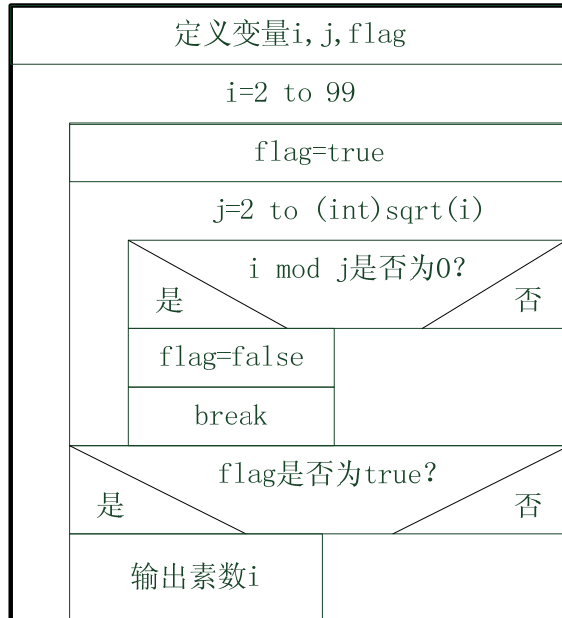
### 三、算法应用题

1.

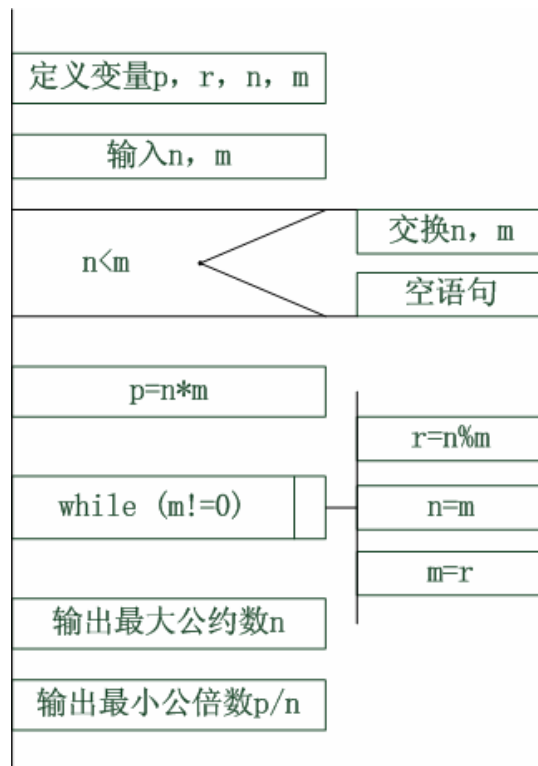




2.



3.



## 第 8 章 一系列的数值——数组

### 一、选择题

1. C
2. D
3. A
4. B
5. C
6. B

### 二、填空题

1. 0
2. 15 , 14
3. 5 , 0
4. 0

## 第 9 章 常用代码的集装箱——函数

### 一、选择题

1. A
2. D
3. A
4. D
5. D
6. A
7. B
8. A
9. A
10. A
11. A
12. B
13. D

## 二、填空题

1. #include
2. 传引用或传地址
3. inline
4. int fun(int ,int )
5. 函数的重载
6. 预编译

## 三、编程题

1. ❶在 Visual C++ 6.0 中,新建名为“ Exer9\_1”的【Win32 Console Application】>【A simple application】项目文件。

❷在工作区【FileView】视图中双击【Source Files】>【Exer9\_1.cpp】,在代码编辑窗口输入以下代码。

```
#include "stdafx.h"
#include <iostream.h>
int ftoc(int);
int main(int argc, char* argv[])
{
    int c,f;
    cout<<"输入华氏温度 : ";
    cin>>f;
    c=ftoc(f);
    cout<<"华氏 : "<<f<<" = 摄氏 : "<<c<<endl;
    return 0;
}
int ftoc(int f)
{
    return (f-32)*5/9;
}
```

2. ❶在 Visual C++ 6.0 中,新建名为“ Exer9\_2”的【Win32 Console Application】>【A simple application】项目文件。

❷在工作区【FileView】视图中双击【Source Files】>【Exer9\_2.cpp】,在代码编辑窗口输入以下代码。

```
#include "StdAfx.h"
#include "iostream"
#include <cmath>
```

```
using namespace std;
int main(int argc, char* argv[])
{
    void baha(int n);
    int n;
    cout<<"输入偶数：";
    cin>>n;
    baha(n);
    return 0;
}
void baha(int n)
{
    int prime(int n);
    int i,j;
    for(i=3;i<=n/2;i=i+2)
    {
        if(prime(i)==1)
        {
            j=n-i;
            if(prime(j)==1)
                cout<<n<<"="<<i<<"+"<<j<<endl;
        }
    }
}
int prime(int n)
{
    int i,k;
    k=sqrt(n);
    for(i=2;i<=k;i++)
    {
        if(n%i==0) break;
    }
    if(i>k)
        return 1;
    else
        return 0;
}
```

3. ❶ 在 Visual C++ 6.0 中，新建名为“Exer9\_3”的【Win32 Console Application】➤【A simple application】项目文件。

❷ 在工作区【FileView】视图中双击【Source Files】➤【Exer9\_3.cpp】，在代码编辑窗口输

入以下代码。

```
#include "stdafx.h"
#include <iostream.h>

double p(int x,int n)
{
    if(n==0)
        return 1;
    else
        if(n==1)
            return x;
        else
            return ((2*n-1)*x*p(x,n-1)-(n-1)*p(x,n-2))/n;
}

int main(int argc, char* argv[])
{
    int x,n;
    cout<<"输入 x : ";
    cin>>x;
    cout<<"输入 n : ";
    cin>>n;
    cout<<"\n  x="<<x<<"  n="<<n<<p(x,n)<<endl;;
    return 0;
}
```

## 第 10 章 内存的快捷方式——指针

### 一、选择题

1. D
2. B
3. A
4. D
5. C
6. D

## 二、填空题

1. 数组指针；指针数组
2. 指向整型常量的指针；指向整型常量的指针。

## 三、简答题

1 ❶ 在 Visual C++ 6.0 中，新建【C++ Source File】源程序。

❷ 在代码编辑窗口中输入以下代码。

```
#include <iostream>
#include <string>
using namespace std;
void fun(char *s)
{
    char arr[10]={0};
    int i,n=strlen(s);
    for(i=0;i<n;i++)
    {
        arr[i]=*(s+n-i-1);
    }
    cout<<arr<<endl;
}
void main()
{
    char *p="abcdefg";
    fun(p);
}
```

2. ❶ 在 Visual C++ 6.0 中，新建【C++ Source File】源程序。

❷ 在代码编辑窗口中输入以下代码。

```
#include <iostream>
#include <string>
using namespace std;
void fun(int *a,int n,int *odd,int *even)
{
    int i;
    for(i=0;i<n;i++)
    {
```

```
        if(a[i]%2==0)
            *odd+=a[i];
        else
            *even+=a[i];
    }
}
void main()
{
    int arr[]={1,8,2,3,11,6};
    int odd,even;
    odd=0;
    even=0;
    fun(arr,6,&odd,&even);
    cout<<"odd="<<odd<<endl;
    cout<<"even="<<even<<endl;
}
```

## 第 11 章 面向对象编程基础——类和对象

### 一、选择题

1. B
2. C
3. C
4. A
5. D

### 二、填空题

1. 对象
2. class、struct
3. 函数指针（或函数名）
4. 不同
5. ~

## 三、简答题

1. struct 和 class 都可以定义类，但是缺省访问权限说明时，struct 的成员是公有的，而 class 的成员是私有的。在 C++ 中，struct 可被 class 代替。

2. 类是类型，是对象的抽象，对象是类的具体实例。一个类可以有多个对象，每个对象都有自己的存储单元，而类不占存储单元。

## 第 12 章 C++ 中的文件夹——命名空间

### 一、选择题

1. D
2. A
3. B
4. C
5. D

### 二、填空题

1. using namespace std;
2. 命名冲突
3. 公有的
4. 有名命名空间；无名命名空间
5. using namespace IOC = International\_Olympic\_Community;

### 三、简答题

1. 有三种方式访问命名空间中的成员：命名空间限定（使用域作用符号 ::）、using 声明和 using 指令。

2. 命名空间是类的一种组织管理方式，可以根据需要把相关的类放在同一个命名空间中，实现类的分类整理，给类库添加了结构和层次组织关系。如果把类比作计算机里的文件，命名空间就好比文件夹，它包含了若干的文件（类），这样可以将定义的很多类整理地摆放起来，不仅可以避免命名冲突，还可以简化对类成员的访问。





## 第 13 章 父子之间——继承

### 一、选择题

1. A
2. B
3. B
4. D
5. C

### 二、填空题

1. 多重继承
2. 封装、继承
3. 基类
4. 继承
5. 作用域分隔符限定时

### 三、简答题

1. 单重继承的一般形式如下：

```
class 派生类名:访问权限 基类名
{
    ...
}
```

- 多重继承的一般形式如下：

```
class 派生类名:访问权限 1 基类名 1, 访问权限 2 基类名 2, ..., 访问权限 n 基类名 n
{
    ...
}
```

访问权限是访问控制说明符，它可以是 public、private 或 protected。

2. 多继承类成员的引用比单继承复杂。例如下面的例子，假定 Z 的基类 X 和 Y 均有成员函数 H。

```
class X
{
    public:
    //...
```



```
void H (int part);  
};  
class Y  
{   public:  
    //...  
void H (int part);  
};
```

派生类 Z 将继承这些成员函数，使得下面的调用发生歧义：

Z zObj;

ZObj. H (0);

编译器并不知道是要调用 X::H，还是 Y::H，我们可以通过显式调用来解决这个问题：ZObj.  
Y::H (0);

## 第 14 章 多态与重载

### 一、选择题

1. BD
2. BCD
3. BD
4. BCD
5. D

### 二、填空题

1. 重载，虚函数
2. 基类虚函数
3. 构造函数
4. operator    5. 抽象类；基类

### 三、简答题

1. 同一操作作用于不同的类的实例，将产生不同的执行结果。也就是说，不同的类的对象收到相同的消息时，得到不同的结果。多态性可以分为编译时的多态性和运行时的多态性两大类。编译时的多态又称为静态联编，其实现机制为重载；运行时的多态又称动态联编，其实现机制为



虚函数。

2.运算符重载的本质就是函数重载，他们仅仅在定义和调用的形式有区别。两者都是实现多态的有效途径。运算符重载的限制性更大一些，其名字是有严格规定的，而且不是多有的运算符都可以进行重载。

## 第 15 章 用户与计算机的交互——输入和输出

### 一、选择题

- 1. D
- 2. D
- 3. C

### 二、填空题

- 1. setw ; setfill ; setprecision
- 2. abc
- 3. abc
- 4. 12.345000 ; 12.35

### 三、简答题

1

名称	含义
showpos	在整数和零前显示+号
showbase	十六进制整数前加 0x，八进制整数前加 0
showpoint	浮点输出即使小数点后都是零也加小数点
left	左对齐，右边填充字符
right	右对齐，左边填充字符



dec	十进制显示整数
oct	八进制显示整数
hex	十六进制显示整数
fixed	定点数格式输出
scientific	科学计数法输出
fill( char ) setfill(char)	设置填充字符，默认为右对齐，即左填充
precision( int ) setprecision(int)	设置有效位数
width( int ) setw(int)	设置显示宽度，这里需要注意的是，它是一次性操作的，第二次再使用将

## 2 (1) 第一种形式

int get() 功能：读取一个字符，返回类型是整型。

## (2)第二种形式

istream &get(char &ch) 功能：读取一个字符，包括空白符，并将它存储到 ch 中。它返回被应用的 istream 对象。

## (3) 第三种形式

istream &get(char \*str,int length,char delimiter='\n') 功能：str 代表一个字符数组，用来存储读取到的字符。length 代表可以读取字符数量的最大值。delimiter 默认是 '\n'，用于指定一个特定的字符，当遇到该字符时，就会停止读入，delimiter 本身不会被读入。

## 3 格式

printf ( 格式控制列表，输出列表 )；

格式控制列表的格式常用的有：

%d：以带符号的十进制形式输出整数

%o：以八进制无符号形式输出整数

%x：以十六进制无符号形式输出整数

%u：以无符号十进制形式输出整数

%c：以字符形式输出，只输出一个字符

%s：输出字符串

%f：以小数形式输出单，双精度数，隐含输出六位小数

%e：以指数形式输出实数

几种常见的格式符的修饰符有：

m：代表一个正整数，表示数据最小宽度

n：代表一个正整数，对于实数表示输出 n 位小数；对字符串表示截取的字符个数

-：表示输出的数字或字符在域内向左靠，默认右对齐方式

4 产生[ a , b ]之间随机数公式：

$n = a + \text{rand}() \% (b - a + 1)$

## 第 16 章 文件操作

### 一、简答题

1.

```
#include "stdafx.h"
#include "iostream.h"
void main()
{
    int i,j;
    for(i=1;i<=7;i++)
    {
        for(j=20;j<=20+i;j++)
            cout<<" ";
        for(j=1;j<=15-2*i;j++)
            cout<<"*";
        cout<<endl;
    }
}
```

2.

```
#include<fstream.h>
void main(void)
{
    char f2[256];
    cout<<"请输入目标文件名？";
    cin>>f2;
    ofstream out(f2);
    if(!out){ cout<<"\n 不能打开目标文件:"<<f2; return; }
    int i=0;
    while(i<21)
    {
        out<<i;
        i=i+1;
    }
}
```

```
    }
    out.close();
    cout<<"\n 操作完毕！\n";
}

3.
#include<fstream.h>
void main(void)
{
    ofstream out("data2.dat",ios::out|ios::binary);
    if(!out){cout<<"data2.dat\n";return;}
    for(int i=1;i<100;i+=2)
        out.write((char*)&i,sizeof(int));
    out.close();
    cout<<"\n 程序执行完毕！\n";
}

4.
#include<strstream.h>
void main()
{
    char a[50];
    char b[50];
    istrstream sin(a);
        ostrstream sout(b,sizeof(b));
    cin.getline(a,sizeof(a));
    char ch=' ';
    int x;
        while(ch!='@')
    {
        if(ch>=48 && ch<=57)
        {
            sin.putback(ch);
            sin>>x;
            sout<<x<<' ';
        }
        sin.get(ch);    }
    sout<<'@'<<ends;
    cout<<b;
    cout<<endl;
}
```

5.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
float a[10],max,min,t;
int i,order;
ofstream outfile("data1.txt",ios::out);
if(!outfile)
{
    cerr<<"Open error!"<<endl;
    return 0;
}
cout<<"Enter 10 float numbers:"<<endl;
for(i=0;i<10;i++)
{
    cin>>a[i];
    outfile<<a[i]<<" ";
}
outfile.close();
ifstream infile("data1.txt",ios::in|ios::nocreate);
if(!infile)
{
    cerr<<"Open error"<<endl;
    return 0;
}
for(i=0;i<10;i++)
{
    infile>>a[i];
    cout<<a[i]<<" ";
}
cout<<endl;
max=a[0];
order=0;
for(i=1;i<10;i++)
{
    if(a[i]>max)
    {
        max=a[i];
        order=i;
    }
}
```



```
        }
    }
    cout<<"max="<<max<<endl<<"order="<<order<<endl;
    min=a[0];
    order=0;
    for(i=1;i<10;i++)
    {
        if(a[i]<min)
        {
            min=a[i];
            order=i;
        }
    }
    cout<<"min="<<min<<endl<<"order="<<order<<endl;
    for(i=0;i<9;i++)
        for(int j=0;j<9-i;j++)
        {
            if(a[j]>a[j+1])
            {
                t=a[j];a[j]=a[j+1];a[j+1]=t;
            }
        }

    for(i=0;i<10;i++)
        cout<<a[i]<<" ";
    cout<<ends;
}
```

## 第 17 章 容器

### 一、选择题

1. D
2. A
3. B
4. C
5. C





## 二、填空题

1. 顺序性容器；关联式容器；容器适配器
2. 向量（Vector）；双端队列（Deque）；列表（List）
3. true
4. 非线性；二叉树
5. 线性链表；信息块；前驱指针；后驱指针

## 三、简答题

1. vector 的查询性能最好，并且在末端增加数据也很好，除非它重新申请内存段；适合高效地随机存储。

list 是一个链表，任何一个元素都可以是不连续的，但它都有两个指向上一元素和下一元素的指针。所以它对插入、删除元素性能是最好的，而查询性能非常差；适合大量地插入和删除操作而不关心随机存取的需求。

2. 映射 map 就是标准模板库中的一个关联容器，它提供一对一的数据处理能力。map 的元素是由 key 和 value 两个分量组成的对偶（key,value）。元素的键 key 是唯一的，给定一个 key，就能唯一地确定与其相关联的另一个分量 value。在使用上 map 的功能是不可取代的，它保存了“键-值”关系的数据，而这种键值关系采用了类数组的方式。数组是用数字类型的下标来索引元素的位置，而 map 是用字符型关键字来索引元素的位置。在使用上 map 也提供了一种类数组操作的方式，即它可以通过下标来检索数据，这是其他容器做不到的。

## 第 18 章 C++中的便利之道——模板

### 一、选择题

1. A
2. D
3. AB
4. C
5. D
6. A
7. C
8. A
9. B

## 二、填空题

1. 函数模板；类模板
2. 模板类
3. 函数模板
4. template；<>
5. 逗号（,）；typename 或 class
6. 类模板
7. T x,T y, T z;、 T a;

## 三、简答题

1. 输出结果如下：  
d=88 I=88  
d=9999 I=9999
2. 输出结果如下：  
12.34  
The value is:56.78  
8910  
2.3 is OK  
I and you

## 第 19 章 C++超值放送——标准库

### 一、选择题

1. B
2. A

### 二、填空题

1. 后进先出
2. 算术；比较
3. #include <string>    using namespace std;

### 三、编程题

编写函数对象，求出 1~10 的阶乘。

❶ 在 Visual C++ 6.0 中，新建名为“FunObj”的【Win32 Console Application】➤【A simple application】项目文件。

❷ 在工作区【FileView】视图中双击【Source Files】➤【FunObj.cpp】，在代码编辑窗口中输入以下代码

```
// FunObj.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
template <class t>
void show(t & v)
{
    if(v.empty())
    {
        cout<<"没有元素"<<endl;
    }
    else
    {
        t::iterator i1=v.begin();
        for(;i1!=v.end();i1++)
            cout<<*i1<<" ";
        cout<<endl;
    }
}

class fun
{
public:
    operator()(int n)
    {
        int s=1;
        for(int i=1;i<=n;i++)
            s=s*i;
        return s;
    }
}
```

```
};  
int main(int argc, char* argv[])  
{  
    int a[10];  
    for(int i=0;i<10;i++)  
        a[i]=i+1;  
    vector<int>v1(a,a+10);  
    vector<int>v2(v1.size());  
    show(v1);  
    transform(v1.begin(),v1.end(),v2.begin(),fun());  
    show(v2);  
    return 0;  
}
```

## 第 20 章 错误终结者——异常处理

### 一、选择题

D

### 二、填空题

1. 语法错误；运行时发生错误
2. throw type exception;; throw;
3. catch ( type [exception]); catch (...)

### 三、简答题

1. 异常处理是对程序运行过程当中突如其来的错误的处理，使用异常处理机制，可以使我们的程序更加安全、可靠。

格式：

```
try  
{  
    throw type1 [param1];  
    throw typen [paramn];  
}
```

```
}  
catch (type1 [param1])  
{  
    语句块 1;  
}  
catch (typen [paramn])  
{  
    语句块 n;  
}  
catch (...)  
{  
    匹配 throw 抛出的任意类型的语句块;  
}
```

2. 不是的, 一般情况是外层的 throw 语句抛出的异常使用外层的 catch 来捕获, 内层的 catch 块捕获的异常是同级 try 块中的 throw 语句抛出来的, 但是有些情况下, 我们需要实现内层抛出的异常由外层的代码来处理, 而不是当前层的 catch 块中解决, 这时使用异常的重新抛出机制。

3. 简单来说, 多重异常的捕获就相当于函数的嵌套调用, 可以实现多重结构。