

C++重点知识点总结及习题

第 1 章 C++的特点

C++的特点： 1. 支持抽象数据类型

2. 多态性，一个接口，多重算法，支持早期联编和滞后联编

3. 继承性，保证了代码复用，支持分类的概念

一个 c++程序由一个或多个函数构成，并且在这些函数中只有一个主函数 main，它是程序执行的入口。

C++程序严格区别字母的大小写。

第 2 章 基本数据类型、运算符与表达式

【内容提要】

数据类型
变量与常量
运算符与表达式
简单输入与输出

【重点与难点】

2.1 数据类型

在 C++ 程序里，每个名字都有一个与之相关联的类型，这个类型决定了可以对这个名字所指代的实体应用什么操作，并决定这些操作将如何做出解释。

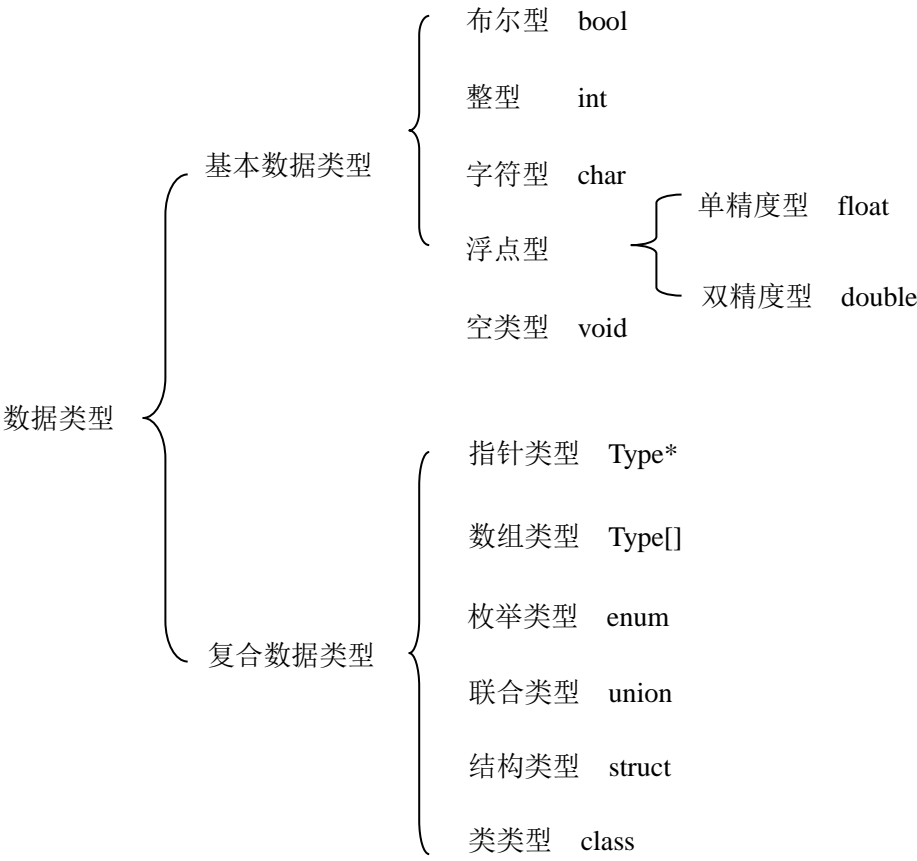


图 2.1 C++的数据类型

图中 Type 表示非控数据类型。

2.1.1 基本数据类型

- 逻辑型：又称布尔型，用关键字 `bool` 表示。有两个值 `true` 或 `false`。`true` 可以转换成整数 1，`false` 可以转换成整数 0。与此对应，整数也可以隐式的转换成 `bool` 值：非 0 的整数转换成 `true`，0 转换成 `false`。
- 字符型：用关键字 `char` 表示。取值包含具体实现所用的字符集里的一个字符。字符型数据通常占用 1 个字节的存储空间。
- 整型：用关键字 `int` 表示。取值是计算机所能表示的所有整数。
- 浮点型：包括单精度型(`float`)和双精度型(`double`)。单精度型通常占用 4 个字节的存储

空间，双精度型通常占用 8 个字节的空間。

✚ 空类型：用关键字 `void` 表示。用于说明某个函数没有返回值。没有 `void` 类型的对象。

在基本的数据类型前可以添加修饰符，以改变基本类型的意义。

`signed`：有符号。

`unsigned`：无符号。

`unsigned` 和 `signed` 只用于修饰 `char` 和 `int`，且 `signed` 修饰词可以省略。当用 `unsigned` 修饰词时，后面的类型说明符可以省略。对于有符号和无符号的整型数据，它们所占的存储空间的大小相同，但它们的表数范围却不相同（对字符型意义相同）。

`short`：短型。只用于修饰 `int`，且用 `short` 修饰时，`int` 可以省略。

`long`：长型。只用于修饰 `int` 和 `double`。当用 `long` 修饰 `int` 时，`int` 可以省略。

数据类型的描述确定了其内存所占空间的大小，也确定了其表示范围。用 `sizeof` (数据类型) 可以确定某数据类型的字节长度。请读者查看在你所用的机器上，下面类型的最大值和最小值是什么：`char`、`short`、`int`、`long`、`float`、`double`、`long double`、`unsigned`。

2.1.2 复合数据类型

✚ 数组：由具有相同数据类型的元素组成的集合。

✚ 结构体：由不同的数据类型构成的一种混合的数据结构，构成结构体的成员的数据类型一般不同，并且在内存中分别占据不同的存储单元。

✚ 共用体：是类似于结构体的一种构造类型，与结构体不同的是构成共同体的数据成员共用同一段内存单元。

✚ 枚举：将变量的值一一列举出来，每个枚举成员（枚举符）都与一个整数相对应。按默认方式，枚举符所赋的值从 0 开始递增。枚举的值也可以用整型的表达式赋值进行初始化。如果某个枚举中所有枚举符的值均非负，该枚举的表示范围是 $[0: 2^k - 1]$ ，其中 2^k 是能使所有枚举符位于此范围内的最小的 2 的幂；如果存在负的枚举符值，该枚举的取值范围就是 $[-2^k: 2^k - 1]$ 。

✚ 指针类型：指针类型变量用于存储另一变量的地址，而不能用来存放基本类型的数据。它在内存中占据一个存储单元。

✚ 类类型：类是体现面向对象程序设计的最基本特征，也是体现 C++ 与 C 最大的不同之处。类是一个数据类型，它定义的是一种对象类型，由数据和方法组成，描述了属于该类型的所有对象的性质。

2.2 变量与常量

2.1.1 变量：指在运行期间其值可以改变的量。

变量有三个特征：变量类型、变量名、变量值。

✚ 命名：遵循标识符命名规则。

标识符是对实体定义的一种定义符，用来标识用户定义的常量名、变量名、函数名、文件名、数组名、和数据类型名和程序名等。只能由字母、数字和下划线组成，且以字母或下划线开头。命名时注意以下几点：

1. C++ 的关键字不能用作用户自定义的实体名；
2. 以下划线开头的名字是保留给实现或者运行环境，用于特殊目的，在应用程序里不要采用这样的名字；
3. 大写与小写字母是区分的；
4. 见名知义。
5. 维持统一的命名风格。

✚ 定义：

格式：<类型名><变量名>[[= <表达式>], ...]

✚ typedef：为某种类型声明一个新名字，而不是定义一种新类型。

格式：typedef <已有类型名> <新类型名>

2.1.2 常量：程序执行期间其值不变的量。主要有下列几类常量。

✚ 布尔常量

✚ 字符常量

用单引号括起的一个字符。在内存中，字符数据以 ASCII 码存储，如字符 ‘a’ 的 ASCII 码为 97。以转义字符 ‘\’ 开始的字符常量后跟不同的字符表示不同的特殊字符。

✚ 字符串常量

由一对双引号括起来的零个或多个字符序列。

字符串可以写在多行上，不过在这种情况下必须用反斜线 ‘\’ 表示下一行字符是这一行字符的延续。

字符串常量实际上是一个字符数组，组成数组的字符除显示给出的外，还包括字符结尾处标识字符串结束的符号 ‘\0’，所以字符串 “abc” 实际上包含 4 个字符：‘a’、‘b’、‘c’ 和 ‘\0’。

注意字符常量与字符串常量在表示、存储、运算方面的区别。

✚ 整型常量

可以用十进制、八进制或十六进制表示。

十进制常量：一般占一个机器字长，是一个带正负号的常数（默认情况下为正数）。

八进制常量：由数字 0 开头，其后由若干 0-7 的数字组成，如 0378，0123 等。

十六进制常量：以 0x 或 0X 开头，其后由若干 0-9 的数字及 A-F（或小写 a-f）的字母组成，如 0x123，0x3ab。

✚ 浮点型常量

只能以十进制形式表示。共有两种表示形式：小数表示法和指数表示法。

如：11.3、.2、2.3e10、-2.3E-1 等。

✚ 枚举常量

枚举声明形式：enum <枚举名>{<枚举符表>}；

枚举符可以有两种形式：

<枚举符名>/<枚举符名>=<整形常量>

✚ 符号常量

定义形式：const <类型名> <符号常量名>=<表达式>[, ...]；

[] 表示可选项，以下都采用这种标记。

定义的符号常量必须初始化。一个符号常量可看作是一个只读变量，由 const 定义的常量的值不可以改变。const 的最常见的用途是作为数组的界和作为分情况标号。

2.3 运算符与表达式

在程序中，表达式是计算求值的基本单位，它是由运算符和运算数组成的式子。运算符是表示进行某种运算的符号。运算数包含常量、变量和函数等。C++语言的运算符按其其在表达式中与运算对象的关系（连接运算对象的个数）可分为：单目运算、双目运算、三目运算。

C++运算符一览表				
名称	运算符	举例	优先级	结合性
作用域区分符	::	::x	1	左结合性
分量运算符	. ->	p.next, p->next	2	
函数调用运算符	()	fac()		
下标运算符	[]	p[10]		右结合性
后增量、后减量	++ --	p++	2	
前增量、前减量	++ --	--p	3	
求字节运算符	sizeof	sizeof(int)	3	
正号、负号	+ -	-1, +3		
指针运算符	* 和 &	*p, &x		
分配、回收空间运算符	New delete	p=new int;		

		delete p;		
强制类型转换运算符	(type)	(int)x		
算术运算符	* / % + -	3%5(取模)	4 5	左结合性
左移、右移	<< >>	8>>3, 8<<3	6	
关系运算符	<<= > >= == !=	3<5 3!=5	7 8	
位运算符	&(按位“与”) ^(按位“异或”) (按位“或”)	1&2 1^2 1 2	9 10 11	
逻辑运算符	&& !	x&&y x y !x	12 13 3	
条件运算符	? :	a>b?x:y	14	右结合性
赋值运算符	= 及其扩展(+= -= *= /= %= 等)	a=2 a+=2	15	
逗号运算符	,	a=1, b=2, c=3	16	左结合性

表 2.1 C++运算符一览表

2.4 简单输入与输出

在 C++中，I/O 使用了流的概念-字符（或字节）流。每一个 I/O 设备传送和接收一系列的字节，称之为流。输入操作可以看成是字节从一个设备流入内存，而输出操作可以看成是字节从内存流出到一个设备。要使用 C++标准的 I/O 流库的功能，需要包括两个头文件：iostream.h 用于提供基本的输入输出功能，iomanip.h 用于提供格式化的功能。

2.4.1 I/O 的书写格式（基本数据类型的输入输出）

头文件 iostream.h 包含有 cin 和 cout 对象，对应于标准输入流和标准输出流。流读取运算符“>>”表示流的输入，可以从 cin 输入流中读取字符；流插入运算符“<<”表示流的输出，可以向 cout 输出流插入字符。

如：cout<<“hello world.”<<endl; //输出 hello world，然后换行

int i,j;

cin>>i>>j; //输入 i, j 的值

2.4.2 使用控制符（基本数据类型输入输出的格式控制）

C++中提供了大量的用于执行格式化输入输出的格式控制符，具体名称及作用请参看表 2.2。

控制符	描述	所在头文件
dec	置基数为 10	iostream.h
hex	置基数为 16	
oct	置基数为 8	
setfill(c)	设填充字符为 c	
setprecision(n)	设显示小数精度为 n 位	
setw(n)	设域宽为 n 个字符	
setiosflags ios::fixed)	固定的浮点显示	iomanip.h
setiosflags ios::scientific)	指数表示	
setiosflags ios::left)	左对齐	
setiosflags ios::right)	右对齐	

setiosflags(ios::skipws)	忽略前导空白
setiosflags(ios::uppercase)	16 进制数大写输出
setiosflags(ios::lowercase)	16 进制数小写输出
setiosflags(ios::showpoin)	显示小数点
setiosflags(ios::showpos)	显示符号（正负号）
setiosflags(ios::showbase)	指定在数值前输出进制（0 表示八进制，0x 或 0X 表示十六进制）

表 2.2 I/O 流的常用控制符

【典型例题】

例题 1：对以下各种数据类型比较所占用存储空间的大小：

- (1) char、int、short int、long int、double、long double.
- (2) signed int、unsigned int.

解答：

(1) 本题主要考查的知识点是各种类型所占用的存储空间的大小以及相应的表数范围。在 32 位计算机中，char 占 1 个字节，short int 占 2 个字节，int 占 4 个字节，long int 占 4 个字节，double 占 8 个字节。sizeof 运算符用于求解某种数据类型的大小。short 和 long 在修饰整型时可以省略 int。答案为：

$1 \equiv \text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long}) \leq \text{sizeof}(\text{double})$

(2) 对于一种确定的数据类型有符号数与无符号数所占的存储空间相同，表数范围不同。修饰符 signed 和 unsigned 只能用于修饰字符型和整型。答案为：

例题 2：下列哪一项能用作用户自定义的标识符：

- (a) const (b) 2var (c) my name (d) var2

解答：

本题主要考查标识符命名规则。C++ 关键字不能用于用户自定义标识符，(a) 中 const 是关键字；第一个字符必须是字母或下划线，(b) 中 2var 是以数字开头；不能含有空格，(c) 中 my name 含有空格。答案为：d。

例题 3：指出下列程序中的错误：_____。

```
int main()
{
    const int x;
    x=100;
    return 0;
}
```

解答：

本题主要考查对符号常量的理解。const 定义的符号常量必须初始化，由 const 定义的常量的值不可以改变。所以本题有两处错误：第一、没有对符号常量 x 进行初始化；第二、给符号常量赋值是错误的。

例题 4：给下列表达式加上全部的括号（假设所用变量均已定义）：

- (1) a+b-c--%b
- (2) a>b?b:c>d?a:c<a?c:d
- (3) a+=a+b||e
- (4) a&b+c++
- (5) -a&&b-c

(6) $k=b=c=a$

解答:

本题主要考查表达式中运算符的优先级与结合性。请参阅表 2.1。为了避免出错,建议读者在书写表达式时完整书写括号。

- (1) 答案为: $(a+b)-((c--)\%b)$
- (2) 答案为: $a>b?b:(c>d?a:(c<a?c:d))$
- (3) 答案为: $a+=((a+b)\|e)$
- (4) 答案为: $a\&(b+(c++))$
- (5) 答案为: $(-a)\&\&(b-c)$
- (6) 答案为: $k=(b=(c=a))$

例题 5: 请根据下列题意写出相应的表达式。

- (1) 有 a 、 b 、 c 、 \max 四个变量 a 、 b 、 c 中的最大值,并将结果放入 \max 中。
- (2) 年龄在 1 到 100 之间(包含 1 和 100,年龄用变量 age 表示)。
- (3) 公式 $\frac{1}{2}(a+b)h$ 。
- (4) 判断一年是否为闰年,年用 year 表示。满足下列两个条件之一即为闰年: ①能被 4 整除但不能被 100 整除 ②能被 400 整除。

解答:

- (1) 主要考查对条件表达式的理解和书写。答案为: $\max=a>b?(a>c?a:c):(b>c?b:c)$ 。
- (2) 主要考查对逻辑表达式的理解和书写。答案为: $1\leq\text{age}\&\&\text{age}\leq 100$ 。
- (3) 主要考查如何在计算机中表示一个数学公式。答案为: $(a+b)*h/2$ 。
- (4) 主要考查对逻辑表达式的理解和书写。答案为: $(\text{year}\%4==0\&\&\text{year}\%100!=0)\|(\text{year}\%400==0)$ 。

例题 6: 下列选项中两个表达式的运算结果相同的是 ()。

- (a) $3/2$ 和 $3.0/2.0$
- (b) $3/2$ 和 $3.0/2$
- (c) $3/2.0$ 和 $3.0/2.0$
- (d) $3/2.0$ 和 $3/2$

解答:

本题考查数据类型及表达式中数据类型的隐式转换。 $3/2$ 中两个操作数都为整型,运算结果仍为整型即 1; $3.0/2$ 和 $3/2.0$ 中一个操作数为整型另一个为浮点型,运算时整型隐式转换为浮点型,运算结果也为浮点型即 1.5; $3.0/2.0$ 两个操作数均为浮点型,结果也为浮点型即 1.5。答案为: (c)。

例题 7:

下列程序的运行结果为: _____。

```
#include<iostream.h>
void main()
{
    int a=2,b=4,i=0,x;
    x=a>b&&++i;
    cout<<"x: "<<x<<endl;
    cout<<"i: "<<i<<endl;
}
```

解答:

本题主要考查“短路”表达式的运算。对于表达式中的“与”运算而言,只要有一个操作数为假,结果为假。所以当第一个操作数为假时,不在求解其它操作数。对于表达式中的“或”运算而言,只要有一个操作数为真,则结果为真。所以当第一个操作数为真时,不在求解其它操作数。本题中 $a>b$ 为假,所以表达式 $a>b\&\&++i$ 为假,而 $++i$ 没有执行。故 i 为 0。答案为: $x:0$

$i:0$

例题 8: 求解下列各表达式的值 (其中 x 的值为 100)。

- (1) (a=1, b=2, c=3)
- (2) 1|3<<5
- (3) 'a'+3&&!0%1
- (4) x%2? "odd" : " even"

解答:

- (1) 逗号表达式的值是其最后一个表达式的值。答案为: 3。
- (2) <<运算符的优先级高于|运算符, 所以先算 3<<5 结果为 96 (二进制 1100000), 然后与 1 做按位与运算则结果为 97 (二进制 1100001)。答案为: 97。
- (3) 参与本题的运算符, 按优先级由高到低依次是: !运算符、算术运算符、逻辑运算符。
'a'+3 时字符型首先隐式转换成整型然后相加结果为 100, !0%1 即 1%1 结果为 0, 100&&0 结果为 0。答案为: 0。
- (4) 算术表达式的优先级高于条件表达式, 所以先算 x%2 结果为 0, 0?" odd" : " even" 结果为 " even"。本题完成判断一个数是奇数还是偶数, 若该数为奇数, 则表达式的值为 " odd", 为偶数, 则表达式的值为 " even"。答案为: " odd"。

例题 9: 下列程序运行结果为: _____。

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    int a=23;
    double b=23.123456789;
    cout<<a<<' \t' <<b<<endl;
    cout<<setprecision(0)<<b<<endl;
    cout<<setiosflags(ios::fixed)<<setprecision(7)<<b<<endl;
    cout<<setiosflags(ios::scientific)<<b<<endl;
    cout<<setprecision(6);
    cout<<setiosflags(ios::showbase);
    cout<<hex<<a<<' \t' <<a<<endl;
    cout<<dec;
    cout<<setw(10)<<setfill('*')<<setiosflags(ios::left)<<a<<endl;
    cout<<setfill(' ');
}
```

解答:

本题主要考查对格式化输入输出的掌握。

- ① 本题主函数中第三行输出 a, b, ' \t' 为转义字符, 其含义是跳过一个制表位。不设置输出宽度时, 默认输出 6 位有效数字, 超出部分四舍五入。所以该行输出为: 23 23.1235。
- ② setprecision(n) 设置显示精度, 最少显示一位有效数字。如果不重新设置, 则其保持效力, 所以使用完后要还原为 6 位默认值。第四行中设置 setprecision(0) 与 setprecision(1) 作用相同, 结果显示一位有效数字即为: 2e+001。
- ③ setiosflags(ios::fixed) 为固定的浮点显示, 其后跟 setprecision(n) 表示小数点后显示精度为 n。所以第五行输出结果为: 23.1234568。
- ④ setiosflags(ios::scientific) 为指数显示, 当其整数部分宽度大于设置的显示精度 (默认为 6 位) 时, 以指数形式显示结果。否则根据设置的 (或默认的) 显示精度显示 n 位有效数字。所以第六行输出结果为: 23.12346。
- ⑤ setiosflags(ios::showbase) 为指定在数值前输出进制。hex 置基数为 16, 且该操作保持效力, 所以使用完后应该恢复为默认值 10 进制。第九行输出结果为: 0x17 0x17。
- ⑥ setw(n) 设域宽为 n 个字符, setfill(c) 设填充字符为 c, setiosflags(ios::left) 为左对齐。第十一行输出结果为: 23*****。

答案为:

23 23.1235
2e+001
23.1234568
23.12346
0x17 0x17
23*****

【习题】

一、 选择题

1. 下列数据类型不是 C++ 语言基本数据类型的是 ()。
(a) 字符型 (b) 整型 (c) 浮点型 (d) 数组

一、下列字符列中，可作为 C++ 语言程序自定义标识符是 ()。 选择题

2.
(a) x (b) -var (c) new (d) 3i
3. 下列数中哪一个是 8 进制数 ()。
(a) 0x1g (b) 010 (c) 080 (d) 01b
4. 已知 a=1, b=2, c=3, 则表达式 ++a || -b && ++c 的值为 ()。
(a) 0 (b) 1 (c) 2 (d) 3
5. 下列表达式选项中, () 是正确的。
(a) ++(a++) (b) a++b (c) a+++b (d) a++++b
6. 已知枚举类型定义语句为: ()。
enum color {RED, BLUE, PINK=6, YELLOW, GREEN, PURPLE=15};
则下列叙述中错误的是 ()。
(a) 枚举常量 RED 的值为 1 (b) 枚举常量 BLUE 的值为 1
(c) 枚举常量 YELLOW 的值为 7 (d) 枚举常量 PURPLE 的值为 15

7. 下列程序的运行结果正确的是 ()。

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    const double pi=3.1415926;
    cout<<setprecision(3)<<pi<<endl
        <<setiosflags(ios::fixed)<<pi<<endl
        <<setprecision(8)<<setfill('*')<<setw(12)<<pi<<endl;
    return;
}
```

(a) 3.142
3.142
**3.14159260
(b) 3.14
3.142
**3.14159260
(c) 3.14
3.14
3.14159260**
(d) 3.14
3.142
***3.1415926

8. 若 int x=3, y=5; 则表达式 x&y++%3 的值为 ()。
(a) 0 (b) 1 (c) 2 (d) 3

9. 下列常量正确的是 ()。
- (a) "hello (b) 1FL (c) 3.14UL (d) 1.8E-3
World"
10. 若 char x=97;, 则变量 x 包含几个字符 ()。
- (a) 1 个 (b) 2 个 (c) 4 个 (d) 8 个

二、填空题

1. c++语言标示符是以字母或_____开头的, 由字母、数字、下划线组成。
2. 在 C++语言中, char 型数据在内存中的存储形式是_____。
3. 在内存中, 存储字符 'x' 占用 1 个字节, 存储字符串 "x" 要占用_____个字节。
4. 符号常量可以用宏定义 define 和_____表示。
5. 转义字符序列中的首字符是_____。
6. 空字符串的长度是_____。
7. 表达式 cout<<' \n' ;还可以表示为_____。
8. 若要为 unsigned int 定义一个新的名字 UINT 应采用的语句是_____。
9. 以下程序不借助第 3 个变量实现 a, b 值得交换, 请填空。

```
#include<iostream.h>
Void main()
{
    int a,b;
    cout<<" 输入 a,b: " ;
    cin>>a>>b;
    a=___a+b_____;
    b=___a-b_____;
    a=___(a-b)/2___;
    cout<<a<<' \t' <<b<<endl;
}
```

10. 大多数 C++程序都要包含头文件_____。该文件中包含了所有输入/输出流操作所需的基本信息。当使用带参数的操作时, 程序中必须包含头文件_____。

三、编程题

1. 编写一个程序, 输入一个三位数, 分别输出该数的百位、十位和个位。
2. 编写一个程序打印出各种基本数据类型、几个你自己定义的复合数据类型的大小。使用 sizeof 运算符。

【参考答案】

一、选择题

1. d
2. a
3. b
4. b
5. c
6. a
7. b
8. c
9. d
10. a

二、填空题

1. 下划线
2. ASCII
3. 2
4. const

```

5. \
6. 0
7. cout<<endl;
8. typedef unsigned int UINT;
9. a+b      a-b      a-b
10. iomanip.h      iostream.h

```

三、编程题

1.

```

#include<iostream.h>
void main()
{
    int num,var1,var2,var3;
    cout<<"请输入一个三位数: "<<endl;
    cin>>num;
    if(num>999||num<100)    //用于检查输入数据的合法性
        cout<<"您的输入有误!"<<endl;
    else
    {
        var1=num/100;
        var2=(num-var1*100)/10;
        var3=num%10;
        cout<<"百位数为: "<<var1<<endl
            <<"十位数为: "<<var2<<endl
            <<"个位数为: "<<var3<<endl;
    }
}

```

2.

```

#include<iostream.h>
#include<iomanip.h>
void main()
{
    int array[10];
    enum month{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};
    char *p;
    cout<<"The size of char is:"<<sizeof(char)<<endl
        <<"The size of int is:"<<sizeof(int)<<endl
        <<"The size of short int is:"<<sizeof(short int)<<endl
        <<"The size of long int is:"<<sizeof(long int)<<endl
        <<"The size of float is:"<<sizeof(float)<<endl
        <<"The size of double is:"<<sizeof(double)<<endl
        <<"The size of long double is:"<<sizeof(long double)<<endl
        <<"The size of signed int is:"<<sizeof(int)<<endl
        <<"The size of unsigned int is:"<<sizeof(unsigned)<<endl
        <<"The size of array is:"<<sizeof(array)<<endl
        <<"The size of month is:"<<sizeof(month)<<endl
        <<"The size of p is:"<<sizeof(p)<<endl;
}

```

第 3 章 C++程序的流程控制

【内容提要】

顺序控制语句
选择控制语句
循环控制语句
跳转语句

【重点与难点】

3.1 顺序控制语句

指按照语句在程序中出现的先后次序一条一条的顺次执行。包含声明语句,表达式语句,输入输出语句等。

3.1.1 表达式语句

任何一个表达式(上一章中给出了详细介绍)后面加上一个分号就构成了表达式语句(没有分号的不是语句)。常见的表达式语句有以下几种:空语句、赋值语句、函数调用语句等。

✚ 空语句:指只有一个分号而没有表达式的语句,它不作任何操作和运算。

格式为: ;

空语句被用在程序的语法上要求一条语句而逻辑上却不需要的时候。

✚ 函数调用语句:是由函数调用表达式加一个语句结束标志(分号“;”)构成的语句。函数的定义和使用请参见第五章中的详细介绍。

3.1.2 输入/输出语句

C++程序没有输入/输出语句,它的输入/输出功能由函数(scanf、printf)或流控制来实现。printf 和 scanf 是标准的输入输出函数,主要在 C 程序设计中使⤵用,其声明包含在头文件 stdio.h 中。在 C++中, I/O 流完全取代了它们,使用 I/O 流输入输出时,必须包含头文件 iostream.h。

从键盘输入语句格式为: cin>>表达式 1[>>表达式 2>>...];

向屏幕输出语句格式为: cout<<表达式 1[<<表达式 2<<...];

注意:连续输入多项数据时应键入空白字符(包括空格键、回车键和 Tab 键)将相邻的两项数据分开。而连续输出时,cout 不会在相邻数据项间加分隔符,为了增强显示效果,用户可通过控制符自定义显示宽度,换行等(部分控制符在 iomanip.h 头文件中声明)。

3.1.3 复合语句

C++语言允许把一组语句用一对花括号括起来,构成一个复合语句。复合语句被视为一个独立的单元它可以出现在程序中任何单个语句可以出现的地方复合语句不需要用分号作为结束。注意空语句;和空复合语句 { } 是等价的。

3.2 选择控制语句

针对于不同情况采用不同的处理方式的问题,用选择结构来实现。选择语句包含:if 语句和 switch 语句。

3.2.1 if 语句

✚ if 语句:

格式: if(<条件表达式>) <语句>

作用:当<条件表达式>为真时,执行<语句>,否则直接执行 if 语句后边的语句。

这里的<语句>可以是一条语句,也可以是几条语句,但对于多条语句的情况必须用 { } 将几条语句括起来构成复合语句,在 if 语句中当做一个整体处理。

✚ if...else 语句:

格式: if(<条件表达式>)

<语句 1>

 else

 <语句 2>

作用：当<条件表达式>为真时，执行<语句 1>；当<条件表达式>为假时，执行<语句 2>。
这里<语句 1>和<语句 2>可以是一条语句也可以是复合语句。

说明：

①if 语句有时可以用条件表达式替代。

②if 语句支持多种形式的嵌套使用，但一般嵌套层次不超过三层。另外约定 else 总是与它上边最近的一个 if 配对。为了防止语法或逻辑错误的发生，建议在较复杂的情形中使用花括号区分。

3.2.2 switch 语句

格式：switch(<表达式>){
 case <常量表达式 1>: <语句序列 1>
 case <常量表达式 2>: <语句序列 2>
 ...
 case <常量表达式 n>: <语句序列 n>
 [default: <语句序列 n+1>]
}

作用：计算<表达式>判断它与哪个<常量表达式>匹配，执行第一个相匹配的<常量表达式>后的语句，直到遇到 break 转去执行 switch 语句以外的语句；如果均不匹配则执行 default 后定义的语句（在 default 省略的情形中，直接执行 switch 语句以外的语句）。

说明：

①switch 语句中的<表达式>的值只能是整型、字符型或枚举型表达式。

②switch 语句中，case 和其后的<常量表达式>间必须有空格否则会产生逻辑错误。case 后的<常量表达式>的值是互不相同的，且它的类型与 switch 后的<表达式>的类型一致。

③通常情况下，每个 case 分支语句结束后都要加一个 break 语句来结束 switch 语句。但在要表示一个范围，或描述一类对象时（如 A,B,C 都属于合格，D 属于不合格）有可能几条 case 分支语句后才有一个 break 语句。

④switch 语句与嵌套的 if 语句都可以用于处理多分支选择的问题。

3.3 循环控制语句

对于同一个语句或同一组语句序列需要重复多次，则采用循环结构。C++提供了三种循环控制语句：while 语句，do...while 语句，for 语句。

3.3.1 while 语句

格式：while(<条件表达式>)

 <语句>

作用：判断<条件表达式>是否为真，若为真则执行<语句>，然后重复判断，直到<条件表达式>为假时跳出 while 循环执行其后的语句。

说明：

①这里的<语句>可以是单条语句也可以是复合语句。

②当<条件表达式>在循环开始就不满足，则不执行 while 循环也就是说 while 循环有可能一次都不执行。

3.3.2 do...while 语句

格式：do{

 <语句>

}while(<条件表达式>);

作用：当流程到达 do 后，立即执行循环体语句，然后再对条件表达式进行判断。若<条件表达式>的值为真，则重复执行循环体语句，直到<条件表达式>为假时退出循环，执行 do...while 循环后的语句。

说明：这种循环中<语句>至少执行一次，当条件在循环开始时就不满足时这种做法是危险的，所以应尽量使用 while 语句而避免使用 do...while 语句。

3.3.3 for 语句

格式: for(<表达式 1>; <表达式 2>; <表达式 3>)

<语句>

作用: <表达式 1>可以称为初始化表达式,一般用于对循环变量进行初始化或赋初值;<表达式 2>可以称为条件表达式,当它的判断条件为真时,就执行循环体<语句>,否则终止循环,退出 for 循环;<表达式 3>可以称为修正表达式,一般用于在每次循环体执行之后,对循环变量进行修改操作。

说明:

①这里的<语句>可以是一条语句也可以是复合语句。

②for 循环中三个表达式可能省略一个、两个或三个都省略,但它们之间分号在三种情况下都不可以省略。

3.4 跳转语句

3.4.1 break 语句

格式: break;

作用: 结束当前正在执行的循环(for、while、do...while)或多路分支(switch)程序结构,转而执行这些结构后面的语句。

说明: 在循环结构中使用时,如果有多层循环,只跳出其所在的最近的循环层。

3.4.2 continue 语句

格式: continue;

作用: 结束当前正在执行的这一次循环(for、while、do...while),接着执行下一次循环。即跳过循环体中尚未执行的语句,接着进行下一次是否执行循环语句的判定。

【典型例题】

例题 1: 下列程序的运行结果为_____。

```
#include<iostream.h>
void main( )
{
    char c='@';
    if (c>='A' && c<='Z') cout<<"是大写字母 ";
    else if (c>='a' && c<='z') cout<<"是小写字母";
    else cout<<"是其它字符";
}
```

解答:

本题主要考查 if 语句的嵌套使用方法。首先判断字符变量 c 是否满足 $c>='A' \ \&\& \ c<='Z'$, 如果满足则输出"是大写字母 "; 否则判断 c 是否满足 $c>='a' \ \&\& \ c<='z'$, 如果满足则输出"是小写字母"否则输出"是其它字符"。else 总是与离它最近的前一个 if 配对。

答案为: 是其它字符。

例题 2: 已定义: char grade; , 若成绩为 A、B、C 时输出合格, 成绩为 D 时输出不合格, 其他情况提示重新输入。要完成以上功能, 则下列 switch 语句正确的是 ()。

(a) switch(grade){
 case 'A':
 case 'B':
 case 'C': cout<<"合格";break;
 case 'D': cout<<"不合格";break;
 default: cout<<"请重新输入: ";
}

(b) switch(grade){
 case 'A':
 case 'B':
 case 'C': cout<<"合格";

```

        case 'D': cout<<"不合格";
        default: cout<<"请重新输入: ";
    }
(c) switch(grade){
    case 'A', 'B', 'C': cout<<"合格";break;
    case 'D': cout<<"不合格";break;
    default: cout<<"请重新输入: ";
}
(d) switch(grade){
    case A:
    case B:
    case C: cout<<"合格";break;
    case D: cout<<"不合格";break;
    default: cout<<"请重新输入: ";
}

```

解答:

本题主要考查 switch 语句的使用。在 switch 语句执行过程中, 找到第一个相匹配的表达式后, 转去执行该 case 后的语句, 直到遇到 break 语句后跳出 switch 语句执行其后的语句。对于选项 b, 若 grade 的值为 A 则执行结果为“合格不合格请重新输入”, 不满足本题的要求; switch 语句多个 case 分支不能简写为多个表达式之间用逗号隔开的一个 case 分支, 选项 c 错误; case 后的表达式只能是整型、字符型或枚举型常量表达式, 选项 d 中 case 后的 A、B、C、D 是变量。答案为: a。

例题 3: 找出并改正下列程序段中的错误:

```

(1) if(x>0);-----①
    y=x+1;-----②
    else;-----③
    y=x-1;-----④
(2) While(i) -----①
    { cout<<i<<endl; -----②
      i--; }-----③
(3) int i=1,sum; -----①
    while(i<=100) -----②
    { sum+=i; -----③
      i++; }-----④
(4) int i=1,sum=0;-----①
    do{ -----②
      sum+=i; -----③
      i++; }while(i<=100) -----④

```

解答:

本题中包含了初学者在编程中容易犯的一些错误, 提醒读者注意。

(1) 本题考查对 if 语句的语法结构的掌握。本题的错误在于在 if 和 else 后不应加分号。答案为: 将①、③句末的分号去掉。

(2) C++是大小写敏感的语言。答案为: 将①中 While 改为 while。

(3) 本题目的是完成 1 到 100 求和, 结果放在 sum 中, 但是 sum 在参与运算前应该首先对其赋值。答案为: 将①改为 int i=1,sum=0;

(4) 本题主要考查对 do...while 的语法结构的掌握以及与 while 结构进行对比区别两者在作用和语法上的不同。答案为: 将第④行 while 后加分号即 while(i<=100);

例题 4: 循环语句 for(int i=0;i<=5&&!i;i++) cout<<i<<endl; 执行循环次数为 ()。

(a) 1 次 (b) 3 次 (c) 5 次 (d) 6 次

解答:

本题考查对 for 循环的理解以及表达式运算。执行 for 循环 i 的初值为 0, 第一次循环时表达式 $0 \leq 5 \& \& !0$ 结果为 1 所以执行循环体输出 0; 然后 i 自加为 1, 计算表达式 $1 \leq 5 \& \& !1$ 结果为 0, 所以退出循环。答案为: a。

例题 5: 程序段:

```
int i, sum=0;
for(i=1; i<=100; sum+=i, i++);
```

与上边程序段不等价的是 ()。

- (a)

```
int sum=0, i=1;
do{
    sum+=i++;
}while(i<=100);
```
- (b)

```
int i=1, sum=0;
while(i<=100)
{
    sum+=i++;
}
```
- (c)

```
int i=1, sum=0;
while(1)
{
    if(i>100)
        break;
    sum+=i++;
}
```
- (d)

```
int i, sum=0;
for(i=1; i<=100; i++, sum+=i);
```

解答:

本题主要考查对各种循环结构的以及它们之间转换关系的理解。本题中的程序段是求解 1 到 100 的和, 循环结束后 i 的值为 101, sum 的值为 5050。选项 a、b 分别用 do...while 和 while 循环完成求解 1 到 100 的和; 选项 c 是永真循环, 通过 break 语句退出循环, 其作用也是求解 1 到 100 的和; 选项 d 中 i 先自加然后求和, 其作用为求解 2 到 101 的和, 循环结束后 i 的值为 101, sum 的值为 5150。答案为: d。

例题 6: 运行下列程序的结果为 (1) _____。(2) _____。

(1)

```
#include<iostream.h>
void main()
{
    int x,y,cmultiple;
    cout<<"输入两个整数:";
    cin>>x>>y;
    cmultiple=x;
    while(1)
    {
        if(cmultiple%y==0)break;
        cmultiple+=x;
    }
    cout<<"最小公倍数:"<<cmultiple<<endl;
}
```

输入: 24

(2)

```
#include<iostream.h>
void main()
{
    int x,y,var1;
    cout<<"输入两个整数: ";
    cin>>x>>y;
    if(x<y)var1=x,x=y,y=var1;
    var1=x%y;
    while(var1)
    {
        x=y;y=var1;var1=x%y;
    }
    cout<<"最大公约数:"<<y<<endl;
}
输入: 24
      7
```

解答:

本题考查理解程序的能力。(1)中求解两个数 x, y 的最小公倍数思路为若 x 能够被 y 整除则 x 就是这两个数的最小公倍数, 否则判断 x 的整数倍是否能被 y 整除, 直到 x 的某个倍数可以被 y 整除, 则该数即为这两个数的最小公倍数。(2)中求解两个数 x, y 的最大公约数思路为用两个数中较大的数作为被除数, 较小的数作为除数, 如果能够整除则较小的数就是这两个数的最大公约数, 否则, 将较小的数作为新的被除数, 将两个数的模作为除数, 重复以上工作直到模为 0, 则这个除数就是 x 和 y 的最大公约数。

答案为: (1) 最小公倍数:168 (2) 最大公约数:1

例题 7: 以下程序的功能是输出 1 到 100 之间每位数的乘积大于每位数的和的数, 如对数字 12 有 $1*2 < 1+2$, 所以不输出这个数; 对数字 23 有 $2*3 > 2+3$ 所以输出这个数。请填空。

```
#include<iostream.h>
void main()
{
    int num, product=1, sum=0, n;
    for (num=1; num<=100; num++)
    {
        product=1; sum=0;
        ① n=num ;
        while ( ② n )
        {
            product*=n%10; sum+=n%10;
            ③ n/=10 ;
        }
        if (product>sum) cout<<num<<endl;
    }
}
```

解答:

本题中变量 num 用于表示 1 到 100 的整数, 变量 $product$ 用于存放每位数的乘积, 变量 sum 用于存放每位数的和, 变量 n 用于存放求解每位数的中间结果。外层 for 循环用于控制判断 1 到 100 的数, 内层 $while$ 循环用于计算每位数的积与每位数的和, if 语句用于判断该数的每位数的乘积是否大于每位数的和, 若满足此条件, 则输出这个数。在①处要对 n 赋值, 最初它是 num 的副本而后用于存放计算每位数的中间结果; ②处为退出 $while$ 循环的条件, 当当前这个数的每位数都被取出, 则内层循环结束; ③改变循环变量的语句, 使得循环在某种情况下可以结束, 这里取 n 除 10 的商, 直到商为 0 时结束内层循环。

答案为：① ② ③

例题 8：以下程序的功能是判断一个数是否为素数。请填空。

```
#include<iostream.h>
void main()
{
    int num;
    cout<<"输入一个正整数： ";
    _____①_____;
    int isprime=1;
    for(int i=2;i<=num-1;i++)
        if(_____②_____)
        {
            isprime=0;
            _____③_____;
        }
    if(isprime)
        cout<<num<<" 是一个素数。"<<endl;
    else
        cout<<num<<" 不是一个素数。"<<endl;
}
```

解答：

本题中变量 num 存放要判断的数，变量 isprime 用于记录该数是否为素数，当 isprime 为 1 时即该数为素数，否则为合数。判断思路为如果 num 能被 2 到 num-1 的任意一个数整除则该数不是素数。①处需要输入待判断的数，②处为判断条件，当检测到 2 到 num-1 中第一个能整除 num 的数时则可判断出该数不是素数，此时退出循环，故③为退出语句。

答案为： ①cin>>num ②num%i==0 ③break

例题 9：编写一个程序，输入一个正整数，判断它是否能被 3，5，7 同时整除。

解答：

参考程序如下

```
#include<iostream.h>
void main()
{
    int num;
    cout<<"请输入一个正整数： ";
    cin>>num;
    if(num<0)
        cout<<"输入有误!";
    else
        if(num%3==0&&num%5==0&&num%7==0)
            cout<<num<<" 能被 3、5、7 同时整除。"<<endl;
        else
            cout<<num<<" 不能被 3、5、7 同时整除。"<<endl;
}
```

例题 10：编写一个程序，让用户输入年和月，然后判断该月有多少天。

解答：

算法思想：判断某年某月有多少天，每个月的天数有四种可能：1、3、5、7、8、10、12 月为 31 天，4、6、9、11 月为 30 天，闰年的 2 月有 29 天，不是闰年则 2 月为 28 天。因为每个月的天数有多种可能，我们选择用 switch 语句解决本题。程序流程是首先输入要判断的年月，然后判断该年月应有多少天，最后输出结果。

参考程序如下：

```
#include<iostream.h>
void main()
{
    int year,month,days,leap;
    cout<<"请输入年月:";
    cin>>year>>month;
    switch(month)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:days=31;break;
        case 4:
        case 6:
        case 9:
        case 11:days=30;break;
        case 2:if(year%400==0)
            leap=1;
            else if(year%4==0&&year%100!=0)
                leap=1;
            else leap=0;
            if(leap) days=29;
            else days=28;
    }
    cout<<year<<"年"<<month<<"月的天数为:"<<days<<endl;
}
```

例题 11：编写一个程序。计算若一头母牛，它每年年初生一头小母牛，每头小母牛从出生起第四个年头开始每年也生一头小母牛，按此规律，第 10 年时有多少头母牛？

解答：

算法思想：

(年)	1	2	3	4	5	6	7	8	9	...
(本年的母牛数)	2	3	4	2+4=6	3+6=9	4+9=13	6+13=19	9+19=28	13+28=41	...

假设 $f(n)$ 表示第 n 年的母牛数，已知 $f(0)=0, f(1)=2, f(2)=3, f(3)=4$ ，推得在第 n 年时应有 $f(n-3)$ 头母牛生育出 $f(n-3)$ 头母牛，所以第 n 年共有 $f(n-1)+f(n-3)$ 头母牛。据此得出数学表达式： $f(n)=f(n-1)+f(n-3)$ 。在下边的参考程序中分别用变量 sum、sum1、sum2、sum3 表示 $f(n), f(n-1), f(n-2), f(n-3)$ 其中 n 从 4 变化到 10。

参考程序如下：

```
#include<iostream.h>
void main()
{
    int sum1=2, sum2=3, sum3=4, sum=0, n=10;
    for(int i=4; i<=n; i++)
    {
        sum=sum1+sum3;
        sum1=sum2;
        sum2=sum3;
        sum3=sum;
    }
    cout<<"第十年有"<<sum<<"头牛!"<<endl;
}
```

例题 12: 计算 $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!} + \dots$, 当通项 $\frac{1}{n!} < 10^{-7}$ 时停止计算。

解答：

算法思想：

本题主要考查学生对循环结构的运用以及对数学问题编程技巧的掌握。本题为有规律的若干

项相加，所以采用循环结构处理。循环的条件是当 $\frac{1}{n!} \geq 10^{-7}$ 时继续运算直到这个条件不满足时就达到了题目要求的精度，则停止运算。循环体中完成三件事，首先是求出本次（第 i

次）循环中的 $\frac{1}{i!}$ 的值，然后计算当前的 e，最后完成循环变量增 1 的操作。在求解本题时主要注意计算中所采用的变量的数据类型以及如何完成运算精度的控制。

参考程序如下：

```
#include <iostream.h>
int main()
{
    double e=1.0;
    double x=1.0;
    int i=1;
    while(x>=1.0e-7)
    {
        x=x/i;
        e=e+x;
        i=i+1;
    }
    cout << "e = " << e<<endl;
    return 0;
}
```

【习题】

一、 选择题

1. 在循环语句中使用 break 语句的作用是（ ）。
(a) 结束本次循环 (b) 结束该层循环 (c) 结束所有循环 (d) 结束程序执行
2. 对 if 后的括号中的表达式，要求 i 不为 0 的时候表达式为真，该表达式表示正确的为（ ）。

- (a) i (b) !i (c) i<>0 (d) i=0
3. 下列循环语句的执行次数是 ()。
- ```
while(!1) cout<<" ok!" ;
```
- (a) 0 次                      (b) 1 次                      (c) 2 次                      (d) 无数次
4. 运行下列程序结果为 ( )。
- ```
#include<iostream.h>
void main()
{
    int i;
    for (i=0;i<=10;i++){
        if (i%2) cout<<i;
        else continue;
    }
}
```
- (a) 246810 (b) 12345 (c) 678910 (d) 13579

二、 填空题

1. 结构化程序设计的三种基本结构是_____、_____、_____。
2. continue 语句实现的作用是_____。
3. 若输入 " china 2008!", 运行下列程序的输出结果为_____。

```
#include <iostream.h>
#include <stdio.h>
void main( )
{
    char c;
    int letters=0,digits=0,others=0;
    cout<<"Please input a line charaters"<<endl;
    while ((c=getchar( ))!='\n')
    {
        if (c>='a' && c<='z' || c>='A' && c<='Z' )
            letters++;
        else
            if (c>='0' && c<='9')
                digits++;
            else
                others++;
    }
    cout<<"letters:"<<letters<<endl
        <<"digits"<<digits<<endl
        <<"others"<<others<<endl;
}
```

4. 本程序完成 1!+2! +.....+18! 的计算, 请将程序补充完整。

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    double sum=0,fac=1;
    for(int i=1;i<=18;i++)
    {
```

① _____

```

    }
    cout<<"1!+2!+.....+18!="
        <<setiosflags(ios::fixed)<<setprecision(0)
        <<sum<<setprecision(6)<<endl;
}

```

三、编程题

1. 输入某学生成绩，若成绩在 90-100 输出 " 优秀 "，若成绩在 80-89 输出 " 良好 "，若成绩在 70-79 输出 " 中 "，若成绩在 60-69 输出 " 及格 "，若成绩在 0-59 输出 " 不及格 "。
2. 输入三人数，按从小到大的大顺序输出。
3. 在 100~200 中找出同时满足用 3 除余 2，用 5 除余 3 和用 7 除余 2 的所有整数。
4. 求 100~999 中的。所谓水仙花数是指一个三位数，它的每位数字的立方之和等于该数。例如，因为 $153=1^3+5^3+3^3$ ，所以 153 为水仙花数。
5. 求 1000 之内的所有完数。所谓完数是指一个数恰好等于它的所有因子之和。例如， $6=1+2+3$ ，所以 6 为完数。
6. 编一程序显示如下图案：

```

      *
    * * *
  * * * * *
* * * * * * *
* * * * * * * *
* * * * * * *
  * * * * *
    * * *
      *

```

7. 编一程序显示如下图案：

```

      A
    A B C
  A B C D E
A B C D E F G

```

8. 猴子吃桃问题。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，发现只剩一个桃子了，求猴子第一天究竟摘了多少个桃子？
9. 编程序模拟剪刀，石头和纸游戏。游戏规则为：剪刀剪纸，石头砸剪刀，纸包石头。玩游戏者从键盘上输入 S（表示剪刀）或 R（表示石头）或 P（表示纸），要求两个游戏者交替输入，计算机给出输赢的信息。
10. 编写程序输出菲波那切数列的前 20 项。即前两项为 1，以后每一项为前两项之和。
11. 打印九九乘法表。

【参考答案】

一、选择题

1. b
2. a
3. a
4. d

二、填空题

1. 顺序结构、选择结构（分支结构）、循环结构

2. 跳出本次循环。

3.

please input a line charaters

letters:5

digits:4

others:2

4. ①fac*=i; ②sum+=fac;

三、编程题

1.

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

```
void main()
```

```
{
```

```
    double grade;
```

```
    char* degree;
```

```
    cout<<"请输入学生成绩: ";
```

```
    cin>>grade;
```

```
    if(grade>100||grade<0)
```

```
    {
```

```
        cout<<"您的输入有误!"<<endl;
```

```
        return;
```

```
    }
```

```
    else
```

```
        if(grade>=70)
```

```
            if(grade<80)
```

```
                degree="中";
```

```
            else if(grade<90)
```

```
                degree="良好";
```

```
            else
```

```
                degree="优秀";
```

```
        else if(grade>=60)
```

```
            degree="及格";
```

```
        else
```

```
            degree="不及格";
```

```
    cout<<"分数:"<<grade<<endl
```

```
        <<degree<<endl;
```

```
}
```

2.

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

```
void main()
```

```
{
```

```
    int num1,num2,num3,num;
```

```
    cout<<"请输入三个整数: ";
```

```
    cin>>num1>>num2>>num3;
```

```
    if(num1>num2)
```

```
    {
```

```
        num=num1;
```

```
        num1=num2;
```

```
        num2=num;
```

```
    }
```

```

        if(num1>num3)
        {
            num=num1;
            num1=num3;
            num3=num;
        }
        if(num2>num3)
        {
            num=num2;
            num2=num3;
            num3=num;
        }
        cout<<"三个数按从小到大输出为: "<<endl
            <<num1<<endl
            <<num2<<endl
            <<num3<<endl;
    }
}

```

3.

```

#include<iostream.h>
void main()
{
    cout<<"在 100~200 中同时满足用 3 除余 2,用 5 除余 3 和用 7 除余 2 的整数为: "<<endl;
    for(int i=100;i<=200;i++)
    {
        if(i%3==2&& i%5==3&& i%7==2)
            cout<<i<<endl;
    }
}

```

4.

```

#include<iostream.h>
#include<math.h>
void main()
{
    int x,y,z,sum;
    cout<<"100~999 中的水仙花数为: "<<endl;
    for(int i=100;i<=999;i++)
    {
        x=i/100;
        y=i%100/10;
        z=i%10;
        sum=pow(x,3)+pow(y,3)+pow(z,3);
        if(i==sum)
            cout<<i<<endl;
    }
}

```

5.

```

#include<iostream.h>
void main()
{
    int sum;
    cout<<"1000 之内的所有完数为: "<<endl;
    for(int i=1;i<=1000;i++)
    {
        sum=0;
    }
}

```



```

        for(int j=1;j<=i/2;j++)
            if(i%j==0)
                sum+=j;
        if(i==sum)
            cout<<i<<endl;
    }
}

```

6.

```

#include<iostream.h>
void main()
{
    int i,j,n;
    cout<<"请输入上三角行数: ";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=(n-i)*2;j++)
            cout<<' ';
        for(j=1;j<=2*i-1;j++)
            cout<<"* ";
        cout<<endl;
    }
    for(i=n-1;i>=1;i--)
    {
        for(j=1;j<=(n-i)*2;j++)
            cout<<' ';
        for(j=1;j<=2*i-1;j++)
            cout<<"* ";
        cout<<endl;
    }
}

```

7.

```

#include<iostream.h>
#include<iomanip.h>
void main()
{
    int i,j;
    char x;
    for(i=1;i<=4;i++)
    {
        x='A';
        for(j=1;j<=2*(4-i);j++)
            cout<<' ';
        for(j=1;j<=2*i-1;j++)
        {
            cout<<setw(2)<<x;
            x+=1;
        }
        cout<<endl;
    }
}

```

8.

```

#include<iostream.h>

```

```

void main()
{
    int peach=1;
    for(int day=1;day<10;day++) //day 表示 n 天前
        peach=2*(peach+1);
    cout<<"猴子第一天摘了"<<peach<<"个桃子！ ";
}

```

9.

```

#include<iostream.h>
void main()
{
    char play1,play2;
    char* result;
    cout<<"请两位玩家顺序输入 S （表示剪刀）或 R （表示石头）或 P （表示纸）： ";
    cin>>play1>>play2;
    if((play1=='R' || play1=='S' || play1=='P')&&(play2=='R' || play2=='S' || play2=='P'))
    {
        switch(play1)
        {
            case 'S':
                if(play2=='R') result="玩家 2 赢！ ";
                else if(play2=='P') result="玩家 1 赢！ ";
                else result="平局！ ";
                break;
            case 'R':
                if(play2=='P') result="玩家 2 赢！ ";
                else if(play2=='S') result="玩家 1 赢！ ";
                else result="平局！ ";
                break;
            case 'P':
                if(play2=='S') result="玩家 2 赢！ ";
                else if(play2=='R') result="玩家 1 赢！ ";
                else result="平局！ ";
                break;
        }
        cout<<result<<endl;
    }
    else
        cout<<"输入有误!"<<endl;
}

```

10.

```

#include<iostream.h>
void main()
{
    int item,item1=1,item2=1;
    cout<<"菲波那切数列的前 20 项为： "<<endl;
    cout<<item1<<endl;
    cout<<item2<<endl;
    for(int i=3;i<=20;i++)
    {
        item=item1+item2;
        cout<<item<<endl;
        item1=item2;
    }
}

```

```

        item2=item;
    }
}
11.
#include<iostream.h>
#include<iomanip.h>
void main()
{
    cout<<setw(3)<<'*';
    for(int title=1;title<=9;title++)
        cout<<setw(3)<<title;
    cout<<endl;
    for(int row=1;row<=9;row++)
    {
        cout<<setw(3)<<row;
        for(int line=1;line<=row;line++)
            cout<<setw(3)<<row*line;
        cout<<endl;
    }
}

```

第4章 数组

【内容提要】

一维数组

二维数组

多维数组

【重点与难点】

4.1 数组的基本概念

数组是由一组具有相同数据类型的元素组成的集合。数组的类型就是这组元素的数据类型。构成数组的这组元素在内存中占用一组连续的存储单元。可以用一个统一的数组名标识这一组数据，而用下标来指明数组中各元素的序号。根据数组维数的不同，可分为一维数组、二维数组和 multidimensional 数组，常用的是一维和二维数组。

4.1 一维数组

4.1.1 一维数组的定义与初始化

一维数组的定义

定义格式： 类型 数组名[常量表达式];

说明：

①类型为第二章中介绍的数据类型，这里的类型是数组的类型也就是数组中每个元素的类型。

②数组名是一个标识符，代表数组在内存中的起始地址。数组中各元素在内存中连续存储。

③常量表达式又称下标表达式，就是数组中的元素个数。通常采用符号常量，这样使得程序的伸缩性更强。

一维数组的初始化

在定义数组时对其中的全部或部分元素指定初始值，这称为数组的初始化。

初始化的格式为： 类型 数组名[常量表达式]={值 1，值 2，…，值 n};

说明：

①只有存储类别为静态的或外部的数组才可以进行初始化。

②当初始化的元素比数组中的元素个数少时，则按顺序对前一部分元素赋初值，其余元素自动初始化为 0 或空字符 ‘\0’（对字符数组）；当初始化的元素超过数组元素个数时，编

译器会报错。

③当初始化元素个数与数组实际元素个数相等时，可以省略数组长度但[]不能省略。

④对字符数组进行初始化可以对每个数组元素一一赋初值，也可将一个字符串直接赋值给一个数组。但要注意数组长度除了包含字符串中字符个数还包含一个'\0'字符。

4.1.2 访问数组元素

格式： 数组名[下标]

说明：对于一个有 n 个元素的数组，其下标从 0 开始到 n-1。在引用时要防止越界引用，这样会引起逻辑错误。可以通过 `sizeof(数组名)/sizeof(数组类型)` 来求得实际的数组长度。

4.2 二维数组

4.2.1 二维数组的定义与初始化

二维数组的定义

定义格式： 类型 数组名[常量表达式 1][常量表达式 2];

说明：

①数组名是一个标示符，代表数组在内存中的起始地址，二维数组按“先行后列”的顺序被存储在一维的存储空间中的。

②常量表达式 1 表示数组的行数，常量表达式 2 表示数组的列数。

二维数组的初始化

对二维数组的初始化主要有两种形式：

第一、 数值按行用花括号分组对二维数组初始化。

第二、 所有数值按顺序在一个花括号中给出。

说明：

①对以上两种形式，如果没有给出所有数组元素，则剩余元素自动初始化为 0。

②若在一个花括号中对所有元素赋初值或者按行用花括号分组而组内元素部分或全部赋值，则可以缺省第一维的长度，但是[]不能省略，并且在任何情况下，二维数组第二维的长度均不可省略。

4.2.2 访问数组元素

格式： 数组名[下标 1][下标 2]

说明：

若该数组为 n 行 m 列的二维数组，则下标 1 从 0 到 n-1，下标 2 从 0 到 m-1。

【典型例题】

例题 1：下面定义数组的语句正确的是（ ）。

(a) `int i=6;char a[i]= "hello" ;`

(b) `const int i=5;char a[i]= "hello" ;`

(c) `char a[6]= "hello" ;`

(d) `char a5= "hello" ;`

解答：

本题主要考查字符数组在定义及初始化时需要注意的问题。数组定义中数组长度不能指定为除 const 变量以外的变量，选项 a 中 i 为变量；选项 b 中字符串“hello”含有 6 个字符，而数组 a 的长度是 5 所以编译器报错；选项 d 中 a5 是一个字符变量而不是字符数组，而“hello”含有 6 个字符，无法放在一个字符变量中。所以答案为：c

例题 2：已知数组 a 定义为：`int a[][3]={ {1,2,3}, {4} };`，则 `a[1][2]` 的值为（ ）。

(a) 2

(b) 3

(c) 4

(d) 0

解答：

在对数组进行初始化时只给出部分元素的初始值，则剩余元素自动初始化为 0。答案为：d。

例题 3：已知数组定义为 `int a[2][4];`，下列对数组元素引用正确的为（ ）。

- (a) a[1, 2] (b) a[1][2] (c) a(1, 2) (d) a[1][4]

解答:

对二维数组的引用格式为: 数组名[下标 1][下标 2]; n 行 m 列的二维数组其第一维的下标从 0 到 n-1, 第二维的下标从 0 到 m-1。选项 a 和 c 引用格式错误, d 中下标越界。

答案为: b。

例题 4: 下列程序段错误的是 ()。

- (a) char str1[8];
 cin>>str1;
(b) char str1[8];
 strcpy(str1, " first");
(c) char str1[8];
 for(int i=0;i<7;i++)
 cin>>str1[i];
 str1[7]='\0' ;
(d) char str1[8],str2[8]=" first" ;
 str1=str2;

解答:

本题主要考查如何为字符数组赋值。为字符数组赋值可以直接从键盘输入一个字符串、用 strcpy 函数将一个字符串复制到该字符数组或者用循环语句逐个为字符数组元素赋值。不能直接将一个数组赋值给另一个数组。答案为: d。

例题 5: 下列说法正确的是 ()。

- (a) 数组可以存放不同类型的元素。 (b) 定义 int a[2];, 则数组 a 有两个数组元素。
(c) 定义 int a[3];, 则该数组中元素分别为: a[0], a[1], a[2], a[3]。
(d) 在编译时, 不必确定数组的大小。

解答:

数组中所有的元素具有相同类型, 选项 a 错误; 定义 int a[3]; 则数组 a 中含有三个元素分别为 a[0], a[1], a[2], 选项 c 错误; 编译时必须确定数组的大小, 即在定义数组时必须给出数组长度。

答案为: b。

例题 6: 运行下列程序结果为_____。

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    int array1[3][3]={ {1, 2, 13}, {4, 5, 16}, {7, 8, 9} }, i=0, j=2, sum1=0, sum2=0;
    for(;i<3;i++)
    {
        sum1+=array1[i][i];
        sum2+=array1[i][j--];
    }
    cout<<sum1<<setw(5)<<sum2<<endl;
}
```

解答:

本题中程序的作用是对一个给定的 3×3 矩阵求出其主对角线元素值之和与逆对角线元素值之和然后输出计算结果。其中主对角线上的元素即为行和列下标相同的元素, 逆对角线元素下标满足行下标从小到大依次递增同时列下标从大到小依次递减的元素。答案为: 15 25。

例题 7: 下列程序的作用是检查字符串 s 中是否包含字符串 t, 若包含, 则返回并输出 t 在

s 中的开始位置（下标值），否则返回-1。请将程序补充完整。

```
#include<iostream.h>
int main()
{
    int i, j, k;
    char s[20]="Today is sunday!", t[10]="sun";
    for(i=0; s[i]!='\0'; i++)
    {
        for(j=i, k=0; _____ ① _____ && s[j] == t[k]; j++, k++);
        if(_____ ② _____)
        {
            cout<<"t 在 s 中的开始位置下标为:"<<i<<endl;
            return i;
        }
    }
    return -1;
}
```

解答：

本程序用于解决检查一个字符串是否包含于另一个字符串。程序的思想是从 s 串的第 0 个字符起与 t 串进行比较，若到 t 串结束时所比字符均相等则说明 t 在 s 中，返回 t 在 s 中的开始位置，程序结束；否则从 s 的第 1 个字符起重复以上操作，依此类推直到匹配或 s 串结束为止。程序中变量 i 用于记录本轮中正在与 t 串相比较的 s 串的子串的首字符位置（下标），k 用于记录 t 串中当前比较字符的位置（下标），j 用于记录 s 串中当前比较的字符的位置（下标）。因此，答案为：① t[k]!='\0' ② t[k]=='\0'

例题 8：编程实现任意输入 10 个数，然后按从小到大的顺序输出这 10 个数。

解答：

算法思想：关于排序问题的实现算法非常多。有插入排序、希尔排序、快速排序、选择排序、归并排序、基数排序等。在时间空间复杂度方面这些算法各有优缺点。有关这方面的内容可以参考数据结构方面的介绍。在这里给出了一个采用简单选择排序的参考程序。其思路为循环 n-1 次（n 为元素个数），每次循环都从剩下的数中选择出最小的一个。如下给出排序过程：

下标 i	0	1	2	3
list[i]	100	34	20	200
第 1 轮	34	100	20	200
	20	100	34	200
第 2 轮	20	34	100	200
第 3 轮	20	34	100	200

参考程序：

```
#include <iostream.h>
#include<iomanip.h>
int main()
{
    const int COUNT=10;
    int list[COUNT];
    int i, j, tmp;
    cout<<"请输入 10 个数: "<<endl;

    for(i=0; i<COUNT; i++) //输入数组元素
        cin>>list[i];
}
```

```

for(i=0;i<COUNT-1;i++)          //排序
{
    for(j=i+1;j<COUNT;j++)
        if(list[j]<list[i])
        {
            tmp=list[i];
            list[i]=list[j];
            list[j]=tmp;
        }
}

for(i=0;i<COUNT;i++)            //输出排序后的结果
    cout<<setw(5)<<list[i];
cout<<endl;

return 0;
}

```

【习题】

一、选择题

1. 在 C++ 中对数组下标说法正确的是 ()。
 - (a) 初始化数组的值的个数可以多于定义的数组元素的个数，多出部分将被忽略。
 - (b) 初始化数组的值的个数可以少于定义的数组元素的个数。
 - (c) 初始化数组的值的个数必须等于定义的数组元素的个数。
 - (d) 初始化数组的值可以通过跳过逗号的方式来省略。如 `int a[3]={1,,2};`

2. 数组定义为: `int a[2][2]={1,2,3,4};` 则 `a[1][0]%3` 为 ()。

(a) 0 (b) 1 (c) 2 (d) 4

3. 数组定义为: `int a[][2]={5,6,1,2,3,8};` 则能用于计算数组下标的是 ()。

(a) `sizeof(a)/sizeof(int)` (b) `sizeof(a[])/sizeof(3)`
 (c) `sizeof(a[][2])/sizeof(int)` (d) `sizeof(a)/sizeof(a[2][1])`

4. 运行下列程序结果为 ()。

```

#include<iostream.h>
void main()
{
    int a[4]={1,2,3,4};
    for(int i=3;i>=0;i--)
        cout<<a[i];
}

```

(a) 1234 (b) 1324 (c) 4231 (d) 4321

5. 运行下列程序结果为 ()。

```

#include<iostream.h>
void main()
{
    int i, j, t, a[2][2]={8,7,6,5};
    for(i=0;i<1;i++)
        for(j=i+1;j<2;j++)
        {
            t=a[i][j];
            a[i][j]=a[j][i];
            a[j][i]=t;
        }
}

```

```

    }
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            cout<<a[i][j];
        cout<<endl;
    }
}

```

(a) 87 (b) 78 (c) 86 (d) 68
 65 65 75 57

二、填空题

1. $m \times n$ 数组包含_____行、_____列和_____个元素。
2. 定义数组 `int a[10];`，若要给该数组的第三个元素赋值 100，其语句为_____。
3. 已知数组 `a` 中的元素个数为 4，下列语句的作用是将下标为 `i` 的元素移动到下标为 `i-1` 的单元，其中 $1 \leq i < 4$ 。`a` 中原有数据为 1, 2, 3, 4，移动后 `a` 中元素结果为 2,3,4,4。请将下列程序补充完整。

```

#include<iostream.h>
void main()
{
    int a[4]={1,2,3,4};
    for(int i=0;_____①_____;i++)
        a[i]=_____②_____;
}

```

4. 程序填空

运行下列程序后当 `str` 是对称的时，输出“是回文”，否则输出“不是回文”，请将程序补充完整。

```

#include<iostream.h>
void main()
{
    char str[20];
    cin.get(str,20);//输入字符串
    int i=0,j=0;
    while (str[j])_____①_____;
    for(j--; i<j && str[i]==str[j]; i++,j--);
    if(_____②_____)cout<<"是回文";
    else
        cout<<"不是回文";
}

```

5. 运行下列程序的结果为_____。

```

#include<iostream.h>
#include<iomanip.h>
void main()
{
    int array1[3][3]={1,2,3,4,5,6,7,8,9},array2[3][3],i,j;
    for(i=0;i<3;i++)
        for(j=0;j<3;j++)
            array2[j][i]=array1[i][j];
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            cout<<setw(3)<<array2[i][j];
    }
}

```



```

        cout<<endl;
    }
}

```

6. 运行下列程序的结果为_____。

```

#include<iostream.h>
void main()
{
    int num[6], i, j;
    cout<<"请输入 1 到 50 的六个正整数: ";
    for(i=0; i<6; i++)
        cin>>num[i];
    for(i=0; i<6; i++)
    {
        for(j=1; j<=num[i]; j++)
            cout<<"*";
        cout<<endl;
    }
}

```

输入为: 2 4 1 6 3 1

三、编程题

1. 编写一个程序实现矩阵的乘法运算。
2. 输入一个 4×4 矩阵各元素的值, 求解该矩阵中的马鞍点 (即该点的值在它所在的行中最大, 在它所在的列中最小)。
3. 写统计输入的正文中有多少单词的程序, 这里的单词指的是用空白符分隔开的字符串。
4. 编写程序实现一个简单的加密器, 实现英文字符串的加密。加密规则如下: 将字符替换成它后面的第三个字符。如 "abc" 换成 "def"。
5. 编写一程序, 将字符数组 s2 中的全部字符拷贝到字符数组 s1 中。不用 strcpy 函数。拷贝时, '\0' 也要拷贝过去。'\0' 后面的字符不拷贝。
6. 有 17 个人围成一个圈 (编号 0-16), 从第 0 号的人开始从 1 报数, 凡报到 3 的倍数的人离开圈子, 然后再继续数下去。直到最后只剩一个人为止。问此人原来的位置是多少号?
7. 给定一个升序数组, 该数组的元素值为 1, 3, 5, 7, 9, 11, 13, 任意输入一个数判断该数在数组中是否存在。若存在, 给出它在数组中的位置, 否则显示该数不存在。
8. 打印输出杨辉三角形 (共输出 10 行)。

【参考答案】

一、选择题

- 1.b
- 2.a
- 3.a
- 4.d
- 5.c
- 6.

二、填空题

1. m n $m \times n$
2. a[2]=100;
3. ①i<3 ②a[i+1]
4. ①j++ ②i==j
5.

1	4	7
2	5	8

3 6 9

6.

```
**
****
*
*****
***
*
```

三、编程题

1.

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    const int A1=2,A2=3,A3=2;
    int i,j,k;
    double array1[A1][A2]={{1.0,2.0,3.0},{4.0,5.0,6.0}},
           array2[A2][A3]={{1.0,1.0},{1.0,1.0},{1.0,1.0}},
           array3[A1][A3]={0,0,0,0};
    for(i=0;i<A1;i++)
        for(j=0;j<A3;j++)
            for(k=0;k<A2;k++)
                array3[i][j]+=array1[i][k]*array2[k][j];
    for(i=0;i<A1;i++)
    {
        for(j=0;j<A3;j++)
            cout<<setw(4)<<array3[i][j];
        cout<<endl;
    }
}
```

2.

```
#include<iostream.h>
void main()
{
    int array[4][4],i,j,k,max,col,flag=0;
    cout<<"请输入 4 行 4 列矩阵: ";
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            cin>>array[i][j];
    for(i=0;i<4;i++)
    {
        max=array[i][0];col=0;
        for(j=1;j<4;j++)
        {
            if(array[i][j]>max)
            {
                max=array[i][j];
                col=j;
            }
        }
        for(k=0;k<4;k++)
        {
            if(array[k][col]<=max&& k!=i)
                break;
        }
    }
}
```

```

    }
    if(k==4)
    {
        cout<<"马鞍点在"<<i<<"行"<<col<<"列:"<<array[i][col];
        flag=1;
    }
}
if(!flag)
    cout<<"这个矩阵中没有马鞍点！";
}

```

3.

```

#include<iostream.h>
void main()
{
    const int MAX=100;
    char str[MAX];
    int i,num=0;
    cin.get(str,100);//输入字符串，以回车结束
    for(i=0;i<MAX&&str[i]!='\0';i++)
    {
        if(str[i]==' ')
            num++;
    }
    if(num!=0)num++;
    cout<<"单词数为: "<<num<<endl;
}

```

4.

```

#include <iostream.h>
void main()
{
    const LENGTH = 100;
    char str[LENGTH];
    cout<<"请输入字符串:";
    cin>>str;
    for(int i=0; i<LENGTH;i++)
    {
        if(str[i] == '\0') break;
        str[i] = str[i] + 3;
    }
    cout<<"加密后的字符串为: "<<str<<endl;
}

```

5.

```

#include<iostream.h>
void main()
{
    int i,j,len1=0,len2=0;
    char s1[20]="hello ",s2[10]="world";
    for(i=0;i<20;i++)
        if(s1[i]!='\0')
            len1++;
    else
        break;
    for(i=0;i<10;i++)

```

```

        if(s2[i]!='\0')
            len2++;
        else
            break;
    if((len1+len2)>19)
    {
        cout<<"存储空间不足!";
        return;
    }
    i=len1;
    j=0;
    while(i<20&& s2[j]!='\0')
        s1[i++]=s2[j++];
    s1[i]='\0';
    cout<<s1<<endl;
}

```

6.

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

```
void main()
```

```

{
    int array[17],n=17,i,j=1;
    for(i=0;i<17;i++)
        array[i]=1;
    i=1;
    while(n!=1)
    {
        if(array[(j-1)%17]==0)
        {
            j++;continue;
        }

        if(i%3==0)
        {
            array[(j-1)%17]=0;
            n--;
        }
        j++;i++;
    }
    for(i=0;i<17;i++)
    {
        if(array[i]==1) cout<<"最后一个人原来的位置（下标）是:"<<i<<endl;
    }
}

```

7.

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

```
void main()
```

```

{
    int array[7]={1,3,5,7,9,11,13},x,low=0,high=6,mid,flag=0;
    cout<<"请输入待查找的数: "<<endl;
    cin>>x;
    while(low<=high)
    {
        mid=(high+low)/2;
        if(x==array[mid])

```

```

{
    cout<<"找到了，该数的下标为："<<mid<<endl;
    flag=1;
    break;
}
else
    if(x<array[mid])
        high=mid-1;
    else
        low=mid+1;
}
if(!flag)
    cout<<"该数不存在！"<<endl;
}

```

8.

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

```
#include<iomanip.h>
```

```
void main()
```

```

{
    const int ROW=10;
    const int COL=10;
    int yh[ROW][COL],row,col;
    for(row=0;row<ROW;row++)
    {
        yh[row][0]=1;yh[row][row]=1;
    }
    for(row=2;row<ROW;row++)
    {
        for(col=1;col<row;col++)
        {
            yh[row][col]=yh[row-1][col-1]+yh[row-1][col];
        }
    }
    for(row=0;row<ROW;row++)
    {
        for(col=0;col<=row;col++)
            cout<<setw(5)<<yh[row][col];
        cout<<endl;
    }
}

```

第 4 章 C++函数与程序结构

【内容提要】

函数的定义与函数原型

函数的调用方法与函数的参数传递规则

带有默认参数的函数

内联函数

函数重载

数组参数

变量的四种存储类型、作用域和生存期

多文件结构

【重点与难点】

5.1 函数的定义与函数原型

5.1.1 函数的定义

在程序中要调用的每个函数都必须给出定义且只能定义一次。

定义格式：

```
    函数类型  函数名（形式参数列表）
    {
        函数体
    }
```

说明：

①函数的类型就是它的返回值（函数处理结果）的类型，其返回值由 return 语句给出。函数可以有返回值也可以没有返回值，当没有返回值时，函数类型声明为 void 型。每个函数都有类型，如果在定义中没有给出类型则默认为 int 型。

②函数可以有 0 个或多个参数，无论有没有参数，函数名后的括号均不能省略；当有多个形式参数时，各参数之间用逗号隔开。每个形式参数由参数类型和参数名组成。

③任何情况下不能在一个函数中定义另一个函数。

5.1.2 函数原型

函数原型告诉编译器函数名、函数类型、函数参数个数及类型，编译器可以对函数调用进行检查。

格式：

```
    函数类型  函数名（形式参数列表）；
```

说明：

①如果在首次使用函数之前已经对函数进行了定义，则不需要函数原型，函数定义就作为函数原型。否则在函数调用前必须给出函数原型说明。

②函数原型与函数定义关于函数类型、函数名、参数类型以及参数个数都是一致的。

③形式参数列表中的每个参数可以只给出参数类型，参数名称只是为了增强程序可读性，可以省略。

5.2 函数的调用方法与函数的参数传递规则

5.2.1 函数的调用方法

✚ 使用函数（即函数调用）主要有三种方式：将函数用作一个独立的表达式语句；用作某条语句的一部分；用作另一个函数的实参。

✚ 格式：

```
    函数名（实际参数列表）
```

✚ 函数调用的过程：首先传递参数，其次执行函数体，最后返回到调用该函数的位置。

5.2.2 参数传递规则

在函数调用时，实参要向形参传递信息使形参具有确切的含义（即使形参具有对应的存

存储空间和初值)。参数传递主要有两种方式：按值传递和引用传递。

按值传递

按值传递参数时，生成实际参数值的副本并传递给被调用函数的形式参数，形参值的改变不会影响到实参。

引用传递

引用传递是将形参作为实参的别名，所以通过形参可以直接访问实参数据，也就是说对形参值的改变就是对实参值的改变。引用传递中需在定义形式参数时在形参前加引用符“&”。

地址传递

地址传递是将实参的地址传递给形参，所以对形参所指地址中的内容进行修改也会使实参值发生改变。按地址传递中需将形式参数的类型定义为指针类型。

5.3 带有默认参数的函数

程序员可以指定参数的默认值。当调用程序没有给出实参时，按指定的默认值为形参赋值。函数调用时实参与形参按照从左到右顺序匹配，当实参全部匹配而形参还有剩余时，则剩下的形参采用默认值。在对默认值进行定义时应该从右向左定义，在一个没有默认值的参数的左边又出现有默认值的参数是错误的。默认参数应在函数名首次出现时定义。

5.4 内联函数

当程序执行到调用普通函数时程序就转去执行该函数，执行完该被调用函数后再返回到调用函数。对于内联函数在编译阶段编译器就把每个出现调用该内联函数的地方都用该函数体中的代码替代。因此内联函数的使用会减少函数调用的开销，但是会增加程序的长度。

定义内联函数时在函数定义前加关键字 `inline`。

内联函数适用于经常使用的小函数。对于内联函数的函数体有一些限制：

- ①内联函数中不能含有任何循环以及 `switch` 和 `goto` 语句；
- ②内联函数中不能说明数组；
- ③递归函数（自己调用自己的函数）不能定义为内联函数。

5.5 函数重载

函数重载是指同一个函数名可以对应多个函数实现。也就是说这些函数具有相同的函数名，完成含义相同的工作，但是它们具有不同的参数（即参数个数或参数类型不同），在函数调用时根据参数的类型、个数决定具体调用哪个函数。

函数重载时首先进行参数完全匹配，当无法完全匹配时，按隐式数据类型转换的方向进行匹配，仍无法匹配时，则报错。函数重载解析与函数定义或声明的顺序无关。

当多个函数参数个数及类型均相同，只有函数返回值类型不同时则报错。

5.6 数组参数

数组作为函数的参数时，它传递的是数组中第 0 个元素的地址（指针）。因此在被调用函数中对形参数组值的改变将被应用到实参数组。

数组长度不是参数类型的一部分，函数不知道传递给它的数组的实际长度，当编译器对实参类型进行参数类型检查时并不检查数组的长度，因此在定义形参时可以只写数组名[]，方括号中是否写长度作用相同。有时在被调用函数中需要知道数组长度，那么可以采用下面两种方法传递数组长度信息。

- ① 提供一个含有数组长度的额外参数，即定义一个用于存放数组长度的形参。
- ② 将被调用函数的形式参数声明为数组的引用，当形式参数是一个数组类型的引用时数组长度成为形式参数类型的一部分，编译器会检查数组实参的长度与在函数形参类型中指定的长度是否匹配。

5.7 变量的作用域与生存期

5.7.1 局部变量与全局变量

程序的内存区域

一个程序将操作系统分配给其运行的内存块分为 4 个区域。

- ① 代码区，存放程序的代码，即程序中各个函数中的代码块。
- ② 全局数据区，存放程序全局数据和静态数据。
- ③ 堆区，存放程序的动态数据。
- ④ 栈区，存放程序的局部数据，即各个函数中的数据。

局部变量

在一个函数内部说明的变量是内部变量，它只在该函数范围内有效。也就是说，只有在包含变量说明的函数内部，才能使用被说明的变量，在此函数之外就不能使用这些变量了。所以内部变量也称“局部变量”。

全局变量

在函数外部定义的变量称为外部变量。外部变量不属于任何一个函数，其作用域是：从外部变量的定义位置开始，到本文件结束为止。外部变量可被作用域内的所有函数直接引用，所以外部变量又称全局变量。

说明：

- ① 在同一源文件中，允许外部变量和内部变量同名。在内部变量的作用域内，外部变量将被屏蔽而不起作用。
- ② 外部变量的作用域是从定义点到本文件结束。如果定义点之前的函数需要引用这些外部变量时，需要在函数内对被引用的外部变量进行说明。外部变量说明的一般形式为：

extern 数据类型 外部变量[，外部变量 2……]；

外部变量的定义，必须在所有的函数之外，且只能定义一次。而外部变量的说明，出现在要使用该外部变量的函数内，而且可以出现多次。

5.7.2 静态变量

静态局部变量

定义格式：**static** 数据类型 内部变量表；

存储特点：

- ① 静态局部变量属于静态存储。在程序执行过程中，即使所在函数调用结束也不释放。换句话说，在程序执行期间，静态内部变量始终存在，但其它函数是不能引用它们的。
- ② 定义但不初始化，则自动赋以"0"（整型和实型）或'\0'（字符型）；且每次调用它们所在的函数时，不再重新赋初值，只是保留上次调用结束时的值！

静态全局变量

- ① 在全局变量前加一个 **static**，使该变量只在这个源文件中可用，称之为全局静态变量。全局静态变量就是静态全局变量。
- ② 静态全局变量对组成该程序的其他源文件是无效的。

5.7.3 生命期

静态生命期

这种生命期与程序的运行期相同，只要程序一开始运行，这种生命期的变量就存在，当程序结束时，其生命期就结束。

局部生命期

在函数内部声明的变量或者是块中声明的变量具有局部生命期。

动态生命期

这种生命期由程序中特定的函数调用（**malloc()**和**free()**）或操作符（**new**和**delete**）来创建和释放。

【典型例题】

例题 1：下列函数定义语句正确的是（ ）。

(a)

```
void fun1(int i,int j);
{
    cout<<i+j;
}
```


(b)

```
void fun1(int i,int j)
{
    void fun2()
    {
        cout<<" This is fun2. " <<endl;
    }
    cout<<i+j;
}
```

(c)

```
void fun1(int i,int j)
{
    cout<<i+j;
    return 1;
}
```

(d)

```
void fun1(int i,int j)
{
    cout<<i+j;
}
```

解答:

本题主要考查对函数定义方法的掌握。函数定义的形式为

函数类型 函数名 (形式参数列表)

```
{
    函数体
}
```

选项 a 中多了一个分号；函数可以嵌套调用但是不能嵌套定义，选项 b 在函数内部定义函数是错误的；若果函数类型定义为 void 型，则该函数没有返回值，选项 c 定义了 void 型函数却有返回值，这是错误的。答案为：d

例题 2：下列关于 C++函数的叙述中，正确的是 ()。

- (a) 每个函数至少要具有一个参数
- (b) 每个函数都必须返回一个值
- (c) 函数在被调用之前必须先声明或定义
- (d) 函数不能自己调用自己

解答:

本题主要考查对函数的要素及其调用方法的理解。函数可以有参数，也可以没有参数（无参函数），选项 a 错误；函数可以有返回值，也可以没有返回值，当没有返回值时将这个函数定义为 void 型，选项 b 错误；如果一个函数自己调用自己则称为递归调用，这是允许的。选项 d 错误。函数在调用之前必须已经声明或定义过。答案为：c。

例题 3：下面的函数声明语句正确的是 ()。

- (a) int fun(int var1=1,char* var2=" Beijing",double var3);
- (b) int fun(int,char* =" Beijing",double =3.14159);
- (c) int fun(int var1=1,char* var2=" Beijing",double var3=3.14159);
int fun(int,char*,double var3= 12.34);
- (d) int fun(int var1=1,char*,double var3=3.14159);

解答:

本题主要考查带默认参数的函数原型声明方法。函数调用时实参与形参按照从左到右顺序匹

配，在对默认值进行定义时应该从右向左定义。选项 a 和 d 都没有遵从对默认值定义时应该从右向左定义的原则，即对于第一个有默认值的参数而言，它后面还有参数没有定义默认值这是错误的。选项 c 对函数中第 3 个参数定义了两次，错误。答案为：b
本题考查默认参数问题。

例题 4：运行下列程序结果为_____。

```
#include <iostream.h>
int f(int[ ],int);
void main()
{
    int a[]={-1,3,5,-7,9,-11};
    cout<<f(a,6)<<endl;
}
int f(int a[],int size)
{
    int i,t=1;
    for(i=0;i<size;i++)
        if(a[i]>0) t*=a[i];
    return t;
}
```

解答：

本题主要考查对数组参数的理解与应用。本程序的作用是计算数组中元素值为正数的元素的乘积。函数 f 含有两个形式参数，第一个参数存放一个整型数组的首地址，第二个参数存放该数组的长度，在该函数中仍然通过数组名[下标]的方式对数组元素进行引用。在主函数中定义了一个数组 a，然后调用函数 f，传递实参时第一个实参 a（数组名）为数组 a 的首地址，第二个参数 6 为数组 a 的长度。答案为：135

例题 5：运行下列程序结果为（ ）。

```
#include<string.h>
#include<iostream.h>
void main( )
{
    char str[][10]={"vb","pascal","c++"},s[10];
    strcpy(s,(strcmp(str[0],str[1])<0?str[0]:str[1]));
    if (strcmp(str[2],s)<0) strcpy(s,str[2]);
    cout<<s<<endl;
}
```

解答：

本程序的作用是将“vb”、“pascal”、“c++”三个字符串中最小的一个放入数组 s，并输出 s。采用二维数组 str 存放三个字符串，则 str[0]、str[1]、str[2] 中分别存放三个字符串的首地址，strcmp() 函数用于比较两个字符串的大小，strcpy() 函数用于将第二个参数所指字符串复制到第一个参数所指数组。C++ 中有一些操作字符串的标准的库函数，头文件 string.h 包含所有字符串处理函数的说明。常用的一些函数如下：

```
strcpy(char destination[], const char source[]);
//把第二个字符串拷贝到第一个字符串
strncpy(char destination[], const char source[], int numchars);
strcat(char target[], const char source[]);
strncat(char target[], const char source[], int numchars);
int strcmp(const char firststring[], const char secondstring);
//比较两个字符串是否相等
strlen( const char string[] );//返回字符串长度
```

本题答案为: c++

例题 6: 运行下列程序的结果为_____。

```
#include<iostream.h>
void fun1(const double& i)
{
    cout<<i<<endl;
}
void fun2(double &j)
{
    cout<<j<<endl;
}
void fun3(double k)
{
    cout<<k<<endl;
}
void main()
{
    fun1('a');
    fun2('a');
    fun3('a');
}
```

- (a) 97 (b) 97 (c) a (d) 程序出错, 无法运行。
- 97 a a
- 97 97 a

解答:

文字量、常量和需要类型转换的参数都可以传递给 const&参数, 但不能传递给非 const 的引用参数。也就是说对非 const 引用参数不允许做类型转换。本题中对于函数 fun1 以及 fun3 在调用时参数都发生了隐形数据类型转换, 而对于函数 fun2 它的参数为非 const 的引用参数, 不允许作类型转换, 所以对 fun2 的调用出错。因此程序出错无法运行。答案为: d。

例题 7: 运行下列程序的结果为_____。

```
#include<iostream.h>
#include<iomanip.h>
int findmax(int Iarg[]);
float findmax(float Farg[]);
double findmax(double Darg[]);

main()
{
    int Iarg[6]={15, 88, 34, 12, 31, 10};
    float Farg[6]={145. 5, 32. 3, 363. 2, 19. 3, 70. 1, 35. 4};
    double Darg[6]={15. 54323, 2. 47763, 63. 29876, 19. 67863, 78. 34541, 35. 44009};
    cout<<"largest value in the Iarg is "<<(findmax(Iarg))<<"\n";
    cout<<"largest value in the Farg is "<<(findmax(Farg))<<"\n";
    cout<<"largest value in the Darg is "<<(findmax(Darg))<<"\n";
    return 0;
}
int findmax(int Iarg[])
{
    int max=0;
```

```

        for(int i=0;i<6;i++)
        {
            if(Iarg[i]>max)
            {
                max=Iarg[i];
            }
        }
    return max;
}
float findmax(float Farg[])
{
    float max=0;
    for(int i=0;i<6;i++)
    {
        if(Farg[i]>max)
        {
            max=Farg[i];
        }
    }
    return max;
}
double findmax(double Darg[])
{
    double max=0;
    for(int i=0;i<6;i++)
    {
        if(Darg[i]>max)
        {
            max=Darg[i];
        }
    }
    return max;
}

```

解答:

本题主要考查函数重载。本程序用于求解并输出一个整型数组的最大元素、一个浮点型数组的最大元素和一个双精度型数组的最大元素。在主函数中调用三次 findmax() 函数, 根据传递实参数组类型不同, 自动调用不同的函数完成求解操作, 这里 findmax() 函数实现了重载。

答案为:

largest value in the Iarg is 88

largest value in the Farg is 363.2

largest value in the Darg is 78.3454

例题 8: 运行下列程序的结果为_____。

```

#include<iostream.h>
int fun(int,int);
void main()
{
    cout<<"n"<<fun(0,0)<<endl;
}
int fun(int n,int s)
{

```

```

    int s1,n1;
    s1=s+n*n;
    if(s1<100)
    {
        n1=n+1;
        fun(n1,s1);
    }
    else
        return n-1;
}

```

解答:

本题主要考查对递归调用程序的理解。

f	s1	n1
f(7,91)		
f(6,55)	91	7
f(5,30)	55	6
f(4,14)	30	5
f(3,5)	14	4
f(2,1)	5	3
f(1,0)	1	2
f(0,0)	0	1

压栈情况如上图所示。当调用 f(7,91) 时返回 7-1=6 程序结束。

答案为: n=6

例题 9: 运行下列程序结果为_____。

```

#include<iostream.h>
void fun()
{
    for(int i=1;i<=3;i++)
    {
        static int var1=1;
        int var2=1;
        cout<<"var1="<<var1++
            <<"",var2="<<var2++<<endl;
    }
}
void main()
{
    fun();
}

```

解答:

本题考查对静态局部变量的理解。静态局部变量存放在内存的全局数据区。函数结束时,静态局部变量不会消失,每次函数调用时,也不会为它重新分配空间,它始终驻留在全局数据区,直到程序运行结束。静态局部变量只在第一次调用时被初始化。在本程序中 for 循环 3 次,但是对静态变量 var1 的初始化只在第一次循环时完成,后两次循环不执行初始化。而变量 var2 在三次循环中都会被初始化。答案为:

var1=1, var2=1

```
var1=2, var2=1
var1=3, var2=1
```

例题 10:

```
#include<iostream.h>
void fun1(int);
void main()
{
    void fun1(double);
    fun1(1);

}
void fun1(int i)
{
    cout<<"int:"<<i<<endl;
}
void fun1(double i)
{
    cout<<"double:"<<i<<endl;
}
```

解答:

本题主要考查重载与作用域的关系问题。本程序中虽然定义了重载函数 fun1。但是在主函数中仅声明了参数类型为 double 的函数 fun1，然后就对 fun1 进行调用（调用前没有看到以整型数据作为形参的 fun1 函数的定义或声明）。那么尽管调用时赋的实参 1 是整型，也无法调用以整型数据作为形参的 fun1 函数。只能通过隐式数据类型转换与以 double 类型作为形参的 fun1 函数匹配。也就是说，在这里实际上没有实现重载。

所以，结果为：double:1。

例题 11: 运行下列程序的结果为 var1=300, var2=400
var1=①, var2= ②。

```
#include<iostream.h>
void output(int var1,int var2)
{
    cout<<"var1="<<var1<<", var2="<<var2<<endl;
    var1=100;
    var2=200;
}
void main()
{
    int var1=300;
    int var2=400;
    output(var1, var2);
    cout<<"var1="<<var1<<", var2="<<var2<<endl;
}
```

解答:

本题主要考查变量的作用域问题。在主函数中定义的变量 var1 和 var2 其作用域为本程序范围，在函数 output 中形参 var1 和 var2 的作用域在 output 函数范围内，因为它和主函数中定义的变量同名，所以在该函数范围内会屏蔽主函数中定义的同名变量，其中的访问变量 var1 和 var2 的操作都是针对于形参变量，与主函数中定义的变量无关。当对 output 函数的调用结束后，形参变量消失，返回主函数，在主函数中访问的变量 var1 和 var2 仍为主函数中所定义的变量。所以，答案为：①300、②400

例题 12: 写一个函数, 它以一个名字作为命令行参数, 打印输出 “hello, name” (其中 name 为输入命令行参数)。

解答:

本程序主要考查对命令行参数的理解及使用。

说明:

```
void main(int argc, char* argv[]) {...
```

即是说, 除按通常的那种无参方式来使用 main 函数外, 如果需要的话, main 函数还可带有两个上述格式以及数据类型的形式参数。

main 带有形式参数时, 其对应实参值由用户在执行该程序时指定, 而后通过操作系统将它们传递给 main 函数。main 函数所含两个参数的含义如下:

argc-----第一参数, 记录命令行参数的个数 (其值为实际命令行参数的个数加 1);

argv-----第二参数, 为字符串数组, 存放执行程序名以及各实际命令行参数。各数组元素的含义为:

argv[0]: 本执行程序的文件名

argv[1]: 第 1 个实际命令行参数 (如果有);

...

argv[argc-1]: 第 argc-1 个实际命令行参数。

在 vc6.0 开发环境下设置命令行参数方法如下:

project→settings...→debug→” program arguments:” 中, 输入以空格分隔的各参数。

在命令提示符环境中, 可通过如下形式运行程序:

程序文件名 参数 1 ...

参数之间用空格隔开。

参考程序为:

```
#include<iostream.h>
void main(int argc, char* argv[])
{
    cout<<"hello, "<<argv[1]<<endl;
}
```

例题 13: 使用重载函数的方法定义两个函数, 用来分别求出两个 int 型数的点间距离和 double 型数的点间距离。两个 int 型点分别为 A(5, 8), B(12, 15); 两个 double 型点分别为 C(1.5, 5.2), D(3.7, 4.6)。

解答:

本题主要考查重载函数的应用。假设有两点 A(x1, y1), B(x2, y2) 则两点之间距离为

$\sqrt{(x2-x1)^2 + (y2-y1)^2}$ 。函数 sqrt() 的作用是求平方根, 使用该函数应包含头文件 math.h。这个程序中要实现求整型的两点之间距离和双精度型的两点之间距离, 作用相同, 但操作的对象类型不同, 结果类型不同, 所以用函数重载解决。

参考程序如下:

```
#include<iostream.h>
#include<math.h>

void main()
{
    double distance(int, int, int, int);
    double distance(double, double, double, double);
    int x1=5, y1=8, x2=12, y2=15;
    double xd1=1.5, yd1=5.2, xd2=3.7, yd2=4.6;
```

```

        double disi=distance(x1,y1,x2,y2);
        cout<<"两个 int 型数的点间距离： ";
        cout<<disi<<endl;
        double disd=distance(xd1,yd1,xd2,yd2);
        cout<<"两个 double 型数的点间距离： ";
        cout<<disd<<endl;
    }
double distance(int a1,int b1,int a2,int b2)
{
    cout<<"\n 调用计算两个 int 型数的点间距离函数\n";
    double s=sqrt((a1-a2)*(a1-a2)+(b1-b2)*(b1-b2));
    return s;
}

double distance(double a1,double b1,double a2,double b2)
{
    cout<<"\n 调用计算两个 double 型数的点间距离函数\n";
    return sqrt((a1-a2)*(a1-a2)+(b1-b2)*(b1-b2));
}

```

例题 14：使用递归函数求解并输出 fibonacci 数列的前十项。

解答：

本题主要考查对递归函数的应用。对于 fibonacci 数列在前边已经介绍，这里使用递归完成，请读者对比递归于非递归编程的方法。

参考程序如下：

```

#include<iostream.h>
#include<iomanip.h>
int fib(int n)
{
    if(n<3) return 1;
    return (fib(n-2)+fib(n-1));
}
void main()
{
    for(int i=1;i<=10;i++)
        cout<<setw(5)<<fib(i);
}

```

例题 15：打印某一年的年历。

解答：

关于本题的编程细节请参见程序中的注释。

参考程序如下：

```

#include<iostream.h>
#include<iomanip.h>

int FirstDayOfYear(int y);
int DaysOfMonth(int m);
void PrintMonth(int m);
void PrintHead(int m);
bool IsLeapYear(int y);

```


[illegible]

//判断某月天数的函数

```
int DaysOfMonth(int m)
{
    switch(m) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12: return 31;
        case 4:
        case 6:
        case 9:
        case 11: return 30;
        case 2: if(IsLeapYear(year))
                return 29;
                else
                return 28;
    }
    return 0;
}
```

//判断是否为闰年

```
bool IsLeapYear(int y)
{
    return ((y%4==0&&y%100!=0) || year%400==0);
}
```

//判断某年的第一天

//因为每年都是 52 个整星期多一天，因此 n 年多出的天数还是 n 天，

//再加上 1900 年到 year 年间因闰年多出来的天数，

//再加上 1900 年元旦的星期序号 1 与 7 取模便得出是第几天。由于 2000 年正好是闰年，所

//以可以计算到 2099 年。

```
int FirstDayOfYear(int y)
{
    int n=y-1900;
    n=n+(n-1)/4+1;
    n=n%7;
    return n;
}
```

【习题】

一、选择题

1. 下列函数定义语句正确的是 ()。

(a)

```
void fun(int var1)
{
    int var1=0;
    cout<<var1<<endl;
}
```

(b)

```
void fun(int var1, var2)
{
    cout<<var1+var2<<endl;
}
```

(c)

```
int fun(int var1)
{
    if(var1)
        return 1;
    else
        return 0;
}
```

(d)

```
int fun(int var1)
{
    if(var1)
        return 1;
    else
        cout<<0<<endl;
}
```

2. 下列叙述中正确的是 ()

- (a) C++语言程序中, main() 函数必须在其它函数之前, 函数内可以嵌套定义函数。
- (b) C++语言程序中, main() 函数的位置没有限制, 函数内不可以嵌套定义函数。
- (c) C++语言程序中, main() 函数必须在其它函数之前, 函数内不可以嵌套定义函数。
- (d) C++语言程序中, main() 函数必须在其它函数之后, 函数内可以嵌套调用函数。

3. 下列对 return 语句叙述错误的是 ()。

- (a) 在函数定义中可能有 return 语句, 也可能没有 return 语句。
- (b) 在函数定义中可以有多条 return 语句。
- (c) 在函数定义中每条 return 语句可能返回多个值。
- (d) 如果函数类型不是 void 型, 则函数定义中必须有 return 语句。

4. C++语言中函数返回值的类型是由 () 决定的。

- (a) return 语句中的表达式类型
- (b) 调用该函数的主调函数类型
- (c) 定义函数时所指定的函数类型
- (d) 以上说法都不正确

5. C++中, 关于参数默认值的描述正确的是 ()。

- (a) 只能在函数定义时设置参数默认值
- (b) 设置参数默认值时, 应当从右向左设置
- (c) 设置参数默认值时, 应当全部设置
- (d) 设置参数默认值后, 调用函数不能再对参数赋值

6. 使用重载函数编程的目的是 ()。

- (a) 使用相同的函数名调用功能相似但参数不同的函数
- (b) 共享程序代码
- (c) 提高程序的运行速度
- (d) 节省存储空间

7. 系统在调用重载函数时, 下列不能作为确定调用哪个重载函数的依据的选项是 ()。

- (a) 函数名
- (b) 参数个数
- (c) 函数类型
- (d) 参数类型

8. 运行下列程序结果为 ()。

```
#include<iostream.h>
inline int abs(int x)
{
    return x<0?-x:x;
}
void main()
{
    int var1=10,var2=-10;
    cout<<" |var1|+|var2|=" <<abs(var1)+abs(var2)<<endl;
}
```

- (a) |var1|+|var2|=0 (b) |var1|+|var2|=20
(c) |var1|+|var2|=10 (d) |var1|+|var2|=-10

9. 数组作为函数的形参，把数组名作为函数的实参时，传递给函数的是 ()。

- (a) 数组中各元素的值
(b) 数组中元素的个数
(c) 数组中第 0 个元素的值
(d) 该数组的首地址

10. 运行下列程序结果为 ()。

```
#include <iostream.h>
int f(int[][3],int,int);
void main()
{
    int a[][3]={0,1,2,3,4,5,6,7,8};
    cout<<f(a,3,3)<<endl;
}
int f(int a[][3],int row,int col)
{
```

```
    int i,j,t=1;
    for(i=0;i<row;i++)
        for(j=0;j<col;j++)
            {a[i][j]++;
             if(i==j) t*=a[i][j];
            }
    return t;
}
```

- (a) 0 (b) 48 (c) 105 (d) 45

11. 运行下列程序的输出结果为 ()。

```
#include<iostream.h>
int var;
main()
{
    int var=2;
    ::var=0;
    if(var>1)
    {
        int var=5;
```

```

    cout<<var;
}
cout<<var;
cout<<::var<<endl;
return 0;
}

```

(a) 20 (b) 000 (c) 520 (d) 500

二、填空题

1. 一个 C++ 语言程序总是从_____开始执行。
2. _____使编译器可以检查传入函数的参数个数、类型和顺序。
3. 在函数原型说明中必须包含的要素有函数类型（如果省略则默认为 int 型）、_____、_____。
4. _____限定符声明只读变量。
5. 若某个函数没有返回值，则该函数的类型应定义为_____类型。
6. 一个函数直接或间接地调用自身，这种现象称为函数的_____。
7. 在一个函数的定义或声明前加上关键字_____则就把该函数定义为内联函数，它主要是解决_____问题。
8. 函数的参数传递的方式分为两类，分别是_____方式和_____方式。
9. 在 c++ 中，可以有多个同名而处理不同参数类型或个数的函数，称为函数_____。
10. 标识符的四种作用域是_____、_____、_____、_____。
11. 要让函数中的局部变量在函数调用之间保持其数值，则要用存储类说明符_____声明。
12. 运行下列程序的结果为_____。

```

#include <iostream.h>
void swap(int &,int &);
void main()
{
    int a=66,b=4;
    cout<<"a="<<a<<" ,b= "
        <<b<<endl;
    swap(a,b);
    cout<<"a="<<a<<" ,b= "
        <<b<<endl;
}
void swap(int &x,int &y)
{
    int t=x;
    x=y;
    y=t;
}

```

13. 运行下列程序的结果为_____。

```

#include <iostream.h>
#include <iomanip.h>
void fun(int array[ ],int n);
void main( )
{
    int a[10]={1,1};
    int i;
    fun(a,10);
    for(i=0;i<10;i++)

```

```

        cout<<setw(4)<<a[i];
    cout<<endl;
}
void fun(int array[ ],int n)
{
    int i;
    for(i=2;i<n;i++)
        array[i]=array[i-1]+array[i-2];
}

```

14. 运行下列程序，若输入 1 2 3 4 5 则输出结果为_____。

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

```

void rev(int n)
{
    int x;
    cin>>x;
    if(n==1)
        cout<<x;
    else
    {
        rev(n-1);
        cout<<x;
    }
}

```

```

void main()
{
    rev(5);
}

```

15. 运行下列程序结果为_____。

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

```
void fun(char PrChar='$',int num=10);
```

```
main()
```

```

{
    char ch;
    int num;
    ch='#';
    num=20;
    fun(ch,num);
    fun();
    fun('&');
    return 0;
}
void fun(char ch,int num)
{
    for(int i=0;i<num;i++)
    {
        cout<<ch;
    }
    cout<<"\n";
}

```

16. 运行下列程序结果为_____。

```

#include <iostream.h>
void fun( );
void main( )
{
    int i;
    for(i=0;i<5;i++)
        fun();
}
void fun( )
{
    static int m=0;
    cout<<m++;
}

```

三、编程题

1. 编写函数将华氏温度转换为摄氏温度，公式为 $C = (F - 32) \times 5 \div 9$ ；并在主函数中调用。
2. 编写函数利用全局变量统计数组中奇数和偶数的个数。
3. 利用重载编写求整数绝对值和求实数绝对值两个函数。
4. 编写函数利用递归的方法计算 x 的 n 阶勒让德多项式的值。该公式如下：

$$P_n(x) = \begin{cases} 1 & (n = 0) \\ x & (n = 1) \\ ((2 * n - 1) * x * P_{n-1}(x) - (n - 1) * P_{n-2}(x)) / n & (n > 1) \end{cases}$$

5. 编写程序，验证哥德巴赫猜想：任何大于 2 的偶数都是两个素数之和（在 1000 以内验证）。

【参考答案】

一、选择题

1. c
2. b
3. c
4. c
5. b
6. a
7. c
8. b
9. d
10. d
11. c

二、填空题

1. main 函数
2. 函数原型
3. 函数名、参数表
4. const
5. void
6. 递归调用
7. inline、程序的运行效率
8. 值传递、引用传递（或地址传递）
9. 重载
10. 函数范围、文件范围、块范围、函数原型范围
11. static

```
double b=-3.14;
```



```

        cout<<abs(a)<<","
            <<abs(b)<<endl;
    }
    int abs(int x)
    {
        return (x>=0?x:-x);
    }
    double abs(double x)
    {
        return (x>=0?x:-x);
    }
4.
#include<iostream.h>
double p(double,int);
void main()
{
    cout<<38.14<<"的"<<5<<"阶勒让德多项式的值为: "<<p(38.14,5)<<endl;
}
double p(double 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;
}
5.
#include<iostream.h>
#include<math.h>
int isprime(int);
void main()
{
    cout<<4<<"="<<2<<"+"<<2<<endl;
    for(int i=6;i<=100;i+=2)
        for(int j=3;j<=i/2;j+=2)
        {
            if(isprime(j)&&isprime(i-j))
            {
                cout<<i<<"="<<j<<"+"<<(i-j)<<endl;
                break;
            }
        }
}
int isprime(int n)
{
    int x=sqrt(n);
    for(int i=2;i<=x;i++)
        if(n%i==0)
            return 0;
    return 1;
}

```

第 6 章 指针、引用和动态空间管理

【内容提要】

指针概述
指针与数组
指针与函数
指针与字符串
动态存储分配
引用的概念及应用

【重点与难点】

6.1 指针概述

指针变量用于存放一个对象在内存中的地址。通过指针变量可以间接访问对象。

6.1.1 指针变量的定义

格式：

存储类型 数据类型 *指针变量名；

说明：

①通常把指针指向的变量的数据类型称为指针的数据类型；指针变量的类型确定后只能指向这种既定的数据类型。而任何一个指针变量本身数据值的类型都是 `unsigned long int`。

②这里的“*”表示定义的是一个指针变量。

6.1.2 指针运算符

& 取地址运算符

* 间接引用运算符

在指针变量的定义和指针变量的引用中都有*p。但引用指针时的*p 与定义指针变量时含义是不同，在指针变量的定义中如 `int *p`；是指 p 是一个指向整型的指针，而在引用时*p是指 p 所指向的变量。

6.1.3 指针运算

✚ 指针的赋值运算

当向指针变量赋值时，赋的值必须是地址常量或变量，不能是普通整数。指针赋值运算常见的有以下几种形式：把一个变量的地址赋予一个指向相同数据类型的指针；把一个指针的值赋予相同数据类型的另外一个指针；把数组的地址赋予指向相同数据类型的指针。

✚ 指针的算术运算

指针与整数的加减运算：指指针从当前指向位置向前或向后移动几个数据单元。由于指针可以指向不同数据类型，即数据长度不同的数据，所以这种运算的结果值取决于指针指向的数据类型。该运算通常用于对数组元素进行操作的场合。

两个指针相减运算：指在两个指针指向的变量类型相同时相减的结果为两个指针所指的地址之间相隔的数据元素的个数。

✚ 指针的关系运算

在两个指向相同类型变量的指针之间可以进行各种关系运算，它实现对两个指针所指变量地址值的比较。

6.1.4 const 指针

✚ 指向常量的指针

格式：`const` 数据类型 *指针变量名；

说明：在程序中不能通过指针来改变它所指向的数据的值，但是指针本身的值可以改变。

✚ 指针常量

格式：数据类型 * `const` 指针变量名=初始地址值；

说明：指针本身的值不可改变，但它所指向的数据的值可以改变。



指向常量的指针常量

格式: `const 数据类型 *const 指针变量名=初始地址值;`

说明: 指针本身的值不可改变, 它所指向的数据的值也不能通过指针改变。

6.2 指针与数组



数组元素的访问

在第四章中已经介绍了如何访问数组元素的简单方法, 这里介绍访问数组元素的另两种方法。

①地址法: 一个数组名代表它的起始地址。地址法即通过地址访问某一数组元素。

如定义 `int a[5];` 则对数组的第 $i+1$ 个元素的访问可以用 `a[i]` 或 `*(a+i)`。

对于二维数组如定义 `int b[2][3];` 则 `b[0]`, `b[1]` 分别代表第一行和第二行的首地址。所以要访问数组元素 `b[i][j]` 可以用下列几种形式: `*(b+i+j)`、`*(b[i]+j)`、`*(b+i)[j]`、`*(b+3*i+j)`。

②指针法: 通过指针访问某一数组元素。

如定义一个指向数组元素的指针, `int a[5], *p=a;` 则对数组的第 $i+1$ 个元素的访问可以用 `*(p+i)` 它等价于 `a[i]`。

对于二维数组如定义 `int b[2][3], *q=b[0];` 则访问数组元素 `b[i][j]` 可以用以下几种形式: `*(q+i+j)`、`*(q[i]+j)`、`*(q+i)[j]`、`*(q+3*i+j)`、`q[i][j]`。



数组指针与指针数组

①数组指针

格式: 数据类型 (*指针名) [常量表达式];

说明: 数组指针是一个指向一维数组的指针变量。

②指针数组

格式: 数据类型 *指针数组名[常量表达式];

说明: 数组元素为指针的数组, 即数组中的每个元素为指向既定的数据类型的指针。

6.3 指针与函数

6.3.1 指针作为函数参数

若指针作为某函数的参数, 对该函数的调用即为传地址调用。

6.3.2 指针型函数

当一个函数的返回值是指针类型时, 这个函数就是指针型函数。它的作用是当需要返回大量数据时可以通过指针型函数完成, 当然这需要有效组织数据所占的内存空间。这种情况多用于返回数组、字符串等。

定义指针函数的函数头的格式为: 数据类型 *函数名 (参数表)

6.3.3 函数指针



函数指针就是指向函数的指针。

定义格式:

数据类型 (*函数指针名) (参数表);

说明: 数据类型是指函数指针所指向函数的返回值的类型, 参数表中指明该函数指针所指向函数的形参类型和个数。



函数指针变量在使用前应该先给它赋值, 格式为: 函数指针名=函数名;



当函数指针指向某函数以后, 可以用下列形式调用函数: (*指针变量) (实参表列)



函数指针数组

定义格式: 数据类型 (*函数指针名[常量表达式]) (参数表);

说明: 函数指针数组中的每个元素是一个指向既定类型函数的指针。每个元素所指向的函数具有相同的数据类型和相同的参数类型和参数个数。

6.4 指针与字符串



可以定义一个字符指针, 通过指针的指向来访问所需的字符。



常用的字符串处理函数:

当程序中使用这些字符串处理函数时, 需要在程序的开始加上头文件 `string.h`。

①strcat ()

函数原型: `char *strcat(char *s1,char *s2)`

函数说明: 字符串连接函数, 将字符串 s2 连接到字符串 s1 的后面, 并返回 s1 的地址值。

②strcmp ()

函数原型: `int strcmp(const char *s1,const char *s2,[int n])`

函数说明: 字符串比较函数, 比较两个字符串 s1 和 s2 的大小 (如果有参数 n, 比较前 n 个字符的大小)。当字符串 s1 大于、等于或小于字符串 s2 时, 函数返回值分别是正数、零和负数。

③strcpy ()

函数原型: `char *strcpy(char *s1,const char *s2)`

函数说明: 将 s2 所指向的字符串复制到 s1 所指向的字符数组中, 然后返回 s1 的地址值。

④strlen ()

函数原型: `int strlen(const char *s)`


函数说明: 返回字符串 s 的长度。

6.5 动态存储分配

动态内存分配的存储空间在堆中, 堆也称为自由存储单元。`new` 运算符与 `delete` 运算符一起使用, 就可以直接进行动态内存的申请和释放 (也称为创建和删除)。


6.5.1 new 运算符

`new` 运算符用于申请所需的内存单元, 返回指定类型的一个指针。

 用 `new` 动态分配某种类型的变量

格式: 指针=`new` 数据类型;

说明: 指针应预先声明, 指针指向的数据类型与 `new` 后的数据类型相同。若申请成功, 则返回分配单元的首地址给指针; 否则 (比如没有足够的内存空间), 则返回 0 (一个空指针)。

 用 `new` 动态分配数组

格式: 指针=`new` 数据类型[常量表达式];

说明: 常量表达式给出数组元素的个数, 指针指向分配的内存首地址, 指针的类型与 `new` 后的数据类型相同。

6.5.2 delete 运算符

`delete` 运算符用于释放 `new` 申请到的内存空间。

格式一: `delete` 指针;

作用: 释放非数组内存单元。

格式二: `delete`[常量] 指针;

作用: 释放数组内存单元。

其中, 指针是指向需要释放的内存单元的指针的名字。并且 `delete` 只是删除动态内存单元, 并不会将指针本身删除。

6.6 引用的概念及应用

引用是个变量的别名, 当建立引用时, 程序用另一个变量或对象 (目标) 的名字初始化它。引用通常用来做函数的参数或函数的返回值。

声明引用的格式为: 类型 &引用名=变量名;

说明:

①引用在声明时必须初始化, 否则会产生编译错误。

②引用一旦初始化, 它就维系在一定的目标上, 再也不分开。任何对该引用的赋值, 都是对引用所维系的目标赋值, 而不是将引用维系到另一个目标上。

③对 `void` 进行引用是不允许的。

④不能建立引用的数组。

⑤引用本身不是一种数据类型, 所以没有引用的引用, 也没有引用的指针。

⑥不可以有空引用。

【典型例题】

例题 1: 下列语句正确的是 ()。

- (a) `int *p=0;`
- (b) `int var1=100,*p=var1; //cannot convert from 'int' to 'int *'`
- (c) `int var1=100;char *p=&var1; // cannot convert from 'int *' to 'char *'`
- (d) `int var1=100,*q;void *p=&var1;q=p; // cannot convert from 'void *' to 'int *'`

解答:

本题主要考查对指针的理解。指针不能被赋予非地址值, 指针不能被初始化或赋值为其他类型对象的地址值。选项 a 中 p 被初始化为没有指向任何对象, 这是可以的; 选项 b 中定义了一个整型变量和一个指向整型的指针, 不能把一个整型变量赋值给一个指向整型的指针, 指针用来存放变量的地址, 所以选项 b 是错误的; 选项 c 中将一个整型变量的地址赋给了指向字符型量的指针, 这是错误的; 选项 d 中将一个整型变量的地址赋给 void 型指针是错误的。所以答案为: a。

例题 2: 有如下定义 `int array[3]={0,1,2},*p=array;`, 则要访问数组中的第二个元素, 下列语句错误的是 ()。

- (a) `*(array+1)`
- (b) `array[1]`
- (c) `*array + 1`
- (d) `*(p+1)`

解答:

数组名 array 和指针变量 p 都存放数组的首地址, 所以选项 a 和 d 都是求出第二个元素的地址, 然后取该地址中的值, 所以这两个选项实现了对数组中第二个元素的访问; 选项 b 为下标法访问数组元素; 选项 c 的作用是求出数组中第一个元素的值, 然后用这个值加 1。所以, 本题答案为: c。

例题 3: 程序改错题

```
#include<iostream.h>
#include<string.h>
void main()
{
    const char *p="hello world!"; //①
    int len=0;
    while(p++) len++; //②
    p-=len+1;
    cout<<"the lenth of "<<p<<" : "<<len<<endl; //③
}
```

答案: `while(*p++) len++;`

解答:

本题主要考查对地址和地址中的值得区分和应用。这个程序的作用是求字符串的长度。这里计算长度的循环语句中的循环条件是只要当前字符不为'\0'表示字符串还没有结束, 则长度加 1, 指针移到字符串中的下一个字符。所以程序中语句②错误, 它没有判断字符而是判断地址是否存在, 这是错误的。应改为: `while(*p++) len++;`

例题 4: 对于定义 `int *f()` 中, 标识符 f 代表的是 ()。

- (a) 一个指向函数的指针
- (b) 一个指针型函数, 该函数返回值为指针
- (c) 一个指向整型数据的指针
- (d) 一个指向数组的指针

解答:

本题主要考查对指针函数和函数指针的理解。这里定义的是指针型函数, 也就是说这个函数的返回值是指针。所以答案为: b。

例题 5: 运行下列程序结果为_____。

```
#include<iostream.h>
#include<stdlib.h>
const int N=3;
```

```

void process(float *p,int n,float (*fun)(float *p,int n));
float arr_add(float arr[],int n)
{
    int i;
    float sum=0;
    for(i=0;i<n;i++)
        sum=sum+arr[i];
    return(sum);
}
float odd_add(float *p,int n)
{
    int i;
    float sum=0;
    for(i=0;i<n;i=i+2,p=p+2)
        sum=sum+*p;
    return(sum);
}
float arr_ave(float *p,int n)
{
    int i;
    float sum=0,ave;
    for(i=0;i<n;i++)
        ave=sum/n;
    return(ave);
}
float arr_max(float arr[],int n)
{
    int i;
    float max;
    max=arr[0];
    for(i=0;i<n;i++)
        if(arr[i]>max)
            max=arr[i];
    return(max);
}
void process(float *p,int n,float (*fun)(float *p,int n))
{
    float result;
    result=(*fun)(p,n);
    cout<<result;
}
void main()
{
    void process(float *p,int n,float (*fun)(float *p,int n));
    float arr_add(float arr[],int n);
    float odd_add(float *p,int n);
    float arr_ave(float *p,int n);
    float arr_max(float arr[],int n);
    static float a[]={ 1.5,3.8,5.6};
    int n=N;
    cout<<"\nthe sum of "<<n<<"element is : "<<"\n";
    process(a,n,arr_add);
    cout<<"\nthe sum of old element is : "<<"\n";
    process(a,n,odd_add);
    cout<<"\nthe sum of average of"<<n<<" element is : "<<"\n";
    process(a,n,arr_ave);
    cout<<"\nthe sum of maximum of"<<n<<" element is : "<<"\n";
}

```

```

    process(a,n,arr_max);
}

```

解答:

本题主要考查对函数指针的理解。本程序的作用是通过调用一系列的函数完成一些运算。函数 arr_add(), odd_add(), arr_ave(), arr_max()的作用分别是求数组元素的和, 求奇数位元素的和 (即求下标为偶数的元素的和), 求数组中元素的平均值, 求数组中元素的最大值。函数 process(float *p,int n,float (*fun)(float *p,int n))的作用是通过函数指针调用不同的函数 (其中第三个参数为函数指针)。在主函数中调用 process()函数, 传递的第一个实参为数组首地址, 第二个实参为数组长度, 第三个实参为函数名, 用于传递函数的首地址。答案为:

the sum of 3element is:

10.9

the sum of old element is:

7.1

the sum of average of3 element is:

0

the sum of maximum of3 element is:

5.6

例题 6: 对于下列函数定义, 说法正确的是 ()。

```

void fun1()
{

```

```

    int var1=2,var2=3;
    int *p=&var1,*q=&var2;
    p=q;

```

```

}

```

```

void fun2()
{

```

```

    int var1=2,var2=3;
    int &p=var1,&q=var2;
    p=q;

```

```

}

```

(a) fun1 与 fun2 的作用完全相同。

(b) 运行 fun1 后 p、q 中均存放 var2 的地址; 运行 fun2 后 p、q 均为 var2 的别名。

(c) 运行 fun1 后 p 中存放 var1 的地址, q 中存放 var2 的地址; 运行 fun2 后 p 为 var1 的别名, q 为 var2 的别名。

(d) 运行 fun1 后 p、q 中均存放 var2 的地址; 运行 fun2 后 p 为 var1 的别名, q 为 var2 的别名。

解答:

本题主要考查引用和指针的区别。对于函数 fun1(), p 和 q 为指向整型的指针, 它们分别被初始化为 var1 和 var2 的地址, p=q 则使 p 中存放的地址为 q 中的地址即为 var2 的地址, 所以最终 p,q 里都存放 var2 的地址。对于函数 fun2(), p 和 q 为引用, 它们分别被初始化为 var1 和 var2 的别名, 引用一旦被初始化, 它就不能再指向其他对象。p=q 的作用是让 var1 的值等于 var2 的值, 即这个语句执行以后 var1 和 var2 的值都为 3, 而 p 和 q 仍然分别为 var1 和 var2 的别名。引用必须被初始化为指向一个对象, 一旦初始化了它就不能再指向其他对象; 指针可以指向一系列不同的对象也可以什么都不指向; 如果一个参数可能在函数中指向不同的对象或者这个参数可能不指向任何对象则必须使用指针参数。答案为: d

例题 7: 编写一个程序完成字符串比较。当两个字符串长度相等时, 对其中各个字符进行比较, 比到第一个不相等的字符为止, 字符大的该字符串就大; 若两个字符串长度不等, 则长度小的字符串小; 当两个字符串长度相等, 对应位字符也相等时, 两个字符串相等。

解答:

参考程序为;

```

#include<iostream.h>

```

```

int strcmp1(const char *str1,const char *str2);

```

```

void main()
{
    char s1[100],s2[100];
    int result;
    cout<<"请输入两个字符串： ";
    cin>>s1>>s2;
    result=strcmp1(s1,s2);
    if(result>0)
        cout<<s1<<">"<<s2<<endl;
    else
        if(result<0)
            cout<<s1<<"<"<<s2<<endl;
        else
            cout<<s1<<"="<<s2<<endl;
}
int strcmp1(const char *str1,const char *str2)
{
    while(*str1!='\0'&&*str2!='\0')
    {
        if(*str1==*str2)
        {
            str1++;str2++;
        }
        else
            return *str1-*str2;
    }
    return *str1-*str2;
}

```

例题 8：输入 10 个国家名称，用指针数组实现排序输出。

解答：

本题主要考查对指针数组的应用。这里用指针数组存放十个国家，并将指针数组作为参数。则数组中的每个元素都是一个指针用于存放字符串的首地址。通过字符串比较函数，比较字符串大小。排序可以采用任何一种排序方法，这里用简单选择排序。

参考程序为：

```

#include<iostream.h>
#include<string.h>
void ccmp(char *a[]);
void main()
{
    char *cname[10]={ "Afghanistan","Australia","Brazil",
                      "Oman ","Romania ","Singapore","Zambia",
                      "Spain ","Mexico","Canada"};

    ccmp(cname);
    for(int i=0;i<10;i++)
        cout<<cname[i]<<endl;
}
void ccmp(char *a[10])
{
    char *p;int i,j;
    for(i=0;i<9;i++)
        for(j=i+1;j<10;j++)
        {
            if(strcmp(a[i],a[j])>0)
            {
                p=a[i];

```



```

        a[i]=a[j];
        a[j]=p;
    }
}

```

【习题】

一、选择题

- 要使变量 *i* 成为 *int* 型变量 *x* 的别名，正确的定义语句是 ()。
(a) `int &i=x;` (b) `int i=&x;` (c) `int &i=&x;` (d) `int i=x;`
- 在下列指针表达式中，与下标访问 `a[i][j]` 不等效的是 ()。
(a) `*(a+i+j)` (b) `*(a+i)[j]` (c) `*(a+i+j)` (d) `*(a[i]+j)`
- 已定义字符串 `char str[5]`，则下列表达式中不能表示 `str[1]` 的地址的是 ()。
(a) `str+1` (b) `str++` (c) `&str[0]+1` (d) `&str[1]`
- 已知：`int a[]={1,2,3,4,5,6}, *p=a, x;` 下面语句中 *x* 的值为 5 的是 ()。
(a) `p+=3; x=*(p++);` (b) `p+=5; x=*p++;` (c) `p+=4; x=*++p;` (d) `p+=4; x=*p++`
- 若有说明：`int i, j=6, *p; p=&i;` 则与 `i=j` 等价的语句是 ()。
(a) `i=*p;` (b) `*p=&j;` (c) `i=&j;` (d) `i=**p;`
- 设 *p1* 和 *p2* 是指向同一个 *int* 型一维数组的指针变量，*k* 为 *int* 型变量，则不能正确执行的语句是 ()。
(a) `k=*p1+*p2;` (b) `p2=k;` (c) `p1=p2;` (d) `k=*p1*(p2);`
- 下面函数的功能是 ()。

```
int fun(char *x)
```

```

{
    char *y=x;
    while(*y++){ };
    return y-x-1;
}

```

- 求字符串的长度
- 求字符串存放位置
- 比较两个字符串的大小
- 将字符串 *x* 连接到字符串 *y* 后面

- 执行以下程序段后，*m* 的值为 ()。

```

int a[2][3]={ {1,2,3}, {4,5,6} };
int m, *p=&a[0][0];
m=(*p)*(p+2)*(p+4);

```

- 15
- 14
- 13
- 12

- 设有如下定义，下面关于 *ptr* 正确叙述是 ()。

```
int (*ptr)();
```

- ptr* 是指向一维数组的指针变量。
- ptr* 是指向 *int* 型数据的指针变量。
- ptr* 是指向函数的指针，该函数返回一个 *int* 型数据。
- ptr* 是一个函数名，该函数的返回值是指向 *int* 型数据的指针。

- 若有如下语句：

```

int **pp, *p, a=10, b=20;
pp=&p;
p=&a;
p=&b;
cout<<*p<<" "<<*pp<<endl;

```

则输出结果是 ()。

- 10,20
- 10,10
- 20,10
- 20,20

二、填空题

1. 运行下列程序结果为_____。

```
#include <iostream.h>
void main()
{ int a[3]={ 10,15,20};
  int *p1=a,*p2=&a[1];
  *p1=*(p2-1)+5;
  *(p1+1)=*p1-5;
  cout<<a[1]<<endl;
}
```

2. 运行下列程序结果为 max=15,min=-5, 请将程序补充完整。

```
#include<iostream.h>
const int N=10;
void max_min(int arr[],int *pt1,int *pt2,int n)
{
    int i;
    *pt1=*pt2=arr[0];
    for(i=1;i<n;i++)
    {
        if(arr[i]>*pt1) _____①_____
        if(arr[i]<*pt2) _____②_____
    }
    return ;
}
void main()
{
    void max_min(int arr[],int *pt1,int *pt2,int n);
    int array[N]={ 1,8,10,2,-5,0,7,15,4,-5};
    int *pt1,*pt2,a,b;
    pt1=&a;
    pt2=&b;
    max_min(_____③_____);
    cout<<"max="<<a<<"min="<<b<<"\n";
}
```

3. 运行下列程序结果为_____。

```
#include<iostream.h>
#include<stdlib.h>
char * month_name(int n)
{
    static char *name[]={
        "ILLEGAL MONTH","JANUARY",
        "FEBRUARY","MARCH","APRIL",
        "MAY","JUNE","JULY","AUGUST",
        "SEPTEMBER","OCTOBER","NOVEMBER","DECEMBER"};
    return((n<1||n>12)?name[0]:name[n]);
}
void main()
{
    char * month_name(int n);
    cout<<"MONTH NO"<<3<<" "<<month_name(3)<<"\n";
}
```

4. 运行下列程序结果为_____。

```

#include<iostream.h>
void callbyval(int a,int b,int c)
{
a=3;b=2;c=1;
}
void callbypointer(int* a,int* b,int* c)
{
*a=3;*b=2;*c=1;
}
void callbyreference(int& a,int& b,int& c)
{
a=1;b=2;c=3;
}
void main()
{
int a=1,b=2,c=3;
int& a1=a;
int& b1=a;
int& c1=a;
callbyval(a,b,c);
cout<<a<<b<<c<<endl;
callbypointer(&a,&b,&c);
cout<<a<<b<<c<<endl;
callbyreference(a1,b1,c1);
cout<<a<<b<<c<<endl;
}

```

三、编程题

1.在很多场合填写金额时，为了防止篡改，要求同时填写数字和大写汉字金额。编写一个程序，输入数字金额，输出其汉字金额（该程序仅处理金额为0.00-99999.99元）。如金额10234.56元，写成壹万零贰百叁拾肆元伍角陆分。

2.使用引用参数编制程序，实现两个字符串变量的交换。例如开始时：

```
char *ap="hello";
```

```
char *bp="how are you";
```

交换后使 ap 和 bp 指向的内容别是：

```
ap: "how are you"
```

```
bp: "hello"
```

3.编程建立一个有序单链表，对该链表完成如下操作：

①输出该链表

②求表长

③插入一个元素，保持有序性

④删除链表中第 i 个位置上的数

【参考答案】

一、选择题

1. a
2. a
3. b
4. d
5. b
6. b
7. a
8. a
9. c

10. d

二、填空题

1. 10
2. ①*pt1=arr[i];②*pt2=arr[i];③array,pt1,pt2,N
3. MONTH NO3 MARCH
4. 123
321
321

三、编程题

1.

参考程序为：

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
    double n;
    const char* a[10]={"零","壹","贰","叁","肆","伍","陆","柒","捌","玖"};
    const char* c[8]={"万","仟","佰","拾","元","角","分"};
    int b[7]={0};
    int i,j,m,k;

    cout<<"\n 请输入金额(0.00~99999.99): \n";
    cin>>n;
    if(n<0||n>99999.99)
    {
        cout<<"输入有误! \n";
        exit(0);
    }
    if(n==0)
        cout<<"汉字大写为: 零元\n";
    else
    {
        k=n*100;
        i=6;
        while(k!=0)
        {
            m=k%10;
            b[i]=m;
            k/=10;
            --i;
        }

        cout<<"汉字大写金额为: \n";

        for(j=0;j<=6;j++)
        {
            if(b[j]!=0)
                break;
        }
        for(k=6;k>=0;k--)
        {
            if(b[k]!=0)
                break;
        }
    }
}
```

```

    for(i=0;i<=6;i++)
    {
        m=b[i];
        if(b[i+1]!=0&&b[i]==0&&i<=3&&i>j&&i<=k)
            cout<<a[0];
        if(m!=0)
            cout<<a[m]<<c[i];
        else
            if(i==4&&n>=1)
                cout<<c[i];
    }
    cout<<endl;
}
}

```

2.

参考程序为：

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

```
void Swap(char*& str1, char*& str2);
```

```
void main()
```

```
{
    char* ap="hello";
    char* bp="how are you?";
    cout <<ap <<endl <<bp <<endl;

```

```
    Swap(ap, bp);

```

```
    cout <<"交换以后:\n";
    cout <<ap <<endl <<bp <<endl;

```

```
}
```

```
void Swap(char*& str1, char*& str2)
```

```
{
    char* temp=str1;
    str1=str2;
    str2=temp;
}
```

3.

参考程序为：

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

```
#include<malloc.h>
```

```
struct node
```

```
{
    int data;
    node *next;
};

```

```
int creat(node* h)    //逆序建立线性链表
```

```
{
    node *p,*s;
    int a;
    p=h;

```

```

cout<<" 请按从大到小的顺序输入结点值,输入 0 结束: \n";
cout<<"请输入结点的值";
cin>>a;
while(a!=0)
{
if(!(s=(node *)malloc(sizeof(node)))) //分配结点空间, 可以该用 new 操作
    return 1;
    else
    {
        s->data=a;
        s->next=p->next;    //每次将新建结点 s 插入到头结点后 (逆序)
        p->next=s;
        cout<<"请输入结点值:";
        cin>>a;
    }
}
return 0;
}

void showlist(node* h) //顺序显示链表内容
{
    node *q;
    q=h->next;
    cout<<"链表内容如下: "<<endl;
    while(q!=NULL)    //遍历
    {
        cout<<"->"<<q->data;
        q=q->next;
    }
    cout<<endl<<endl<<endl;
}

int len(node *h)    //求表长 (不包含头结点)
{
    node *p=h->next;
    int i=0;
    while(p)
    {
        i++;
        p=p->next;
    }
    return i;
}

int insert(node *h)    //插入一个元素, 保持有序性
{
    int a;
    node *s,*p=h;
    cout<<"请输入要插入的数 \n";

```

```

    cin>>a;
    if(!(s=(node*)malloc(sizeof(node))))
        return 1;
    else
    {
        s->data=a;
        while(p->next&& p->next->data<a)
        {
            p=p->next;
        }
        s->next=p->next;
        p->next=s;
    }
    return 0;
}

int del(node *h) //删除链表中第 i 个位置上的数
{
    int i,j;
    node *p=h,*q;
    cout<<" 请输入要删除数的位置: ";
    cin>>i;
    if(i<=0||i>len(h))
    {
        cout<<"位置有误!\n";
        return 1;
    }
    else
    {
        for(j=1;j<i;j++)
        {
            p=p->next;    //p 记录待删除结点的前一结点
        }
        q=p->next;    //q 记录待删除结点
        p->next=q->next;
        free(q);
    }
    return 1;
}

void main()
{
    node *head;

    if(!(head=(node*)malloc(sizeof(node))))
        return;
    else
    {
        head->next=NULL;
        creat(head);
    }
}

```

```
    showlist(head);
    cout<<"链表长度为 : "<<len(head)<<endl<<endl;
    insert(head);
    showlist(head);
    del(head);
    showlist(head);
}
}
```


第 8 章 类和对象的创建

【内容提要】

本章是介绍类和对象的定义、创建的相关内容，这一章围绕对象的生、死与实现来展开。

①**类和对象的关系以及区别**：类和对象的理解要深入，要比较起来看。类是抽象的，对象是具体的；类是模具，对象是由模具生产出来的构件；类是设计阶段的产物，对象是实现阶段的物品。

②**构造函数和析构函数**：类的构造函数和析构函数关系到对象的生死。构造函数有：默认构造函数、拷贝构造函数；析构函数要注意虚函数。

③**对象数组和对象指针**：理解清楚，构建对象数组是需要有默认构造函数的；定义对象指针变量是会调用对象的构造函数的。

④**static 变量**：类的 static 变量属于类，而不是属于某个具体的对象，是所有类对象共有。同理，static 成员函数，亦只可以访问类的 static 变量，而无法访问对象的成员变量。

⑤**友元函数**：破坏类的封装性的声明，记住声明为 friend 友元，仅仅只是注册而已，并非声明这个函数。

⑥**inline 函数**：inline 函数是采用一种源码展开的编译模式，可以提高效率，但会增大程序的体量。一般而言，被频繁调用的 3-5 句简单函数（不包含循环）可以定义为 inline。

【重点与难点】

8.1 类和对象

8.1.1 类的定义

类实质上是用户自定义的一种特殊的数据类型，它不仅包含相关的数据，还包含能对这些数据进行处理的功能，同时，这些数据具有隐蔽性和封装性。类中包含的数据和函数统称为成员，数据称为数据成员，函数称为成员函数，它们都有自己的访问权限。类定义一般分为两部分，即说明部分和实现部分。说明部分用于说明该类中的成员，实现部分用于对成员函数进行定义。

🔗 类定义格式：

```
class 类名
{
    private:
        私有数据成员和成员函数
    protected:
        保护数据成员和成员函数
    public:
        公有数据成员和成员函数
};
各成员函数的实现
```

🔗 类成员的访问控制

类成员有三种不访问权限：私有（private）、保护（protected）、公有（public）。

私有成员只能被本类的成员函数访问及其友元访问。当声明中未指定访问控制时，系统默认该成员为私有成员。

保护成员一般情况下与私有成员的含义相同，在类的继承和派生中与私有成员有不同的含义，保护成员可被本类或派生类的成员函数访问，但不能被外部函数访问。

公有成员可以被程序中的任何函数访问，它提供了外部程序与类的接口功能。

类的数据成员与成员函数

类定义中声明数据成员的数据类型和名称，不能在类内说明数据成员的同时为其赋初值，只有在类的对象定义以后才能给数据成员赋初值。

对于成员函数可以在类内定义，也可以在类内给出函数原型，然后在类外对成员函数进行定义。成员函数在类内说明原型，在类外给出定义时其定义格式如下：

```
返回类型 类名::函数名(参数表)
{ //函数体
}
```

在所定义的函数名前必须加上类名，以表示该函数属于哪个类。类名与函数名之间必须加上作用域运算符::。

注意：如果在类的内部定义成员函数，该成员函数即被声明为内联函数。也可以在类中声明，在类外将该成员函数定义为内联函数。只要在定义前加关键字 `inline`，以显示定义该成员函数为内联函数。这种函数一般是小的、频繁使用的函数。

内联函数的声明有显式声明和隐式声明两种形式。

隐式声明：直接将成员函数定义在类内部

显示声明：将内联函数定义在类外，其声明的形式与在类外定义成员函数的形式类似，但为了使成员函数起到内联函数的作用，在函数声明前要加关键字 `inline`，以显式地声明这是一个内联函数

8.1.2 对象的定义和使用

对象的定义

类定义只是定义了一种新的数据类型，只有定义了类的实例即类的对象以后系统才会为该对象分配存储空间。对象定义可以在类定义的同时直接完成，即在类定义的最后“}”后直接跟对象名列表；也可以在类定义后要使用该对象时定义。

格式：类名对象名（参数列表）；

说明：可以同时定义多个对象，之间用逗号隔开。

对象成员的引用

格式：

对象名.数据成员；

对象名.成员函数（实参表）；

说明：

①在引用成员时要注意访问权限的控制问题。

②对于指向对象的指针在引用其成员时不能使用“.”运算符。其格式为：

对象名->数据成员；

对象名->成员函数（实参表）；

8.2 构造函数和析构函数

8.2.1 构造函数

构造函数是一种特殊的成员函数，被声明为公有成员，其作用是为类的对象分配内存空间，进行初始化。

关于构造函数有以下几点说明：

①构造函数的名字必须与类的名字相同。

②构造函数没有返回值，不能定义返回类型，包括 `void` 型在内。

③对象定义时，编译系统会自动地调用构造函数完成对象内存空间的分配和初始化工作。

④构造函数是类的成员函数，具有一般成员函数的所有性质，可访问类的所有成员，可以是内联函数，可带有参数表，可带有默认的形参值，还可重载。

⑤如果没有定义构造函数，编译系统就自动生成一个缺省的构造函数，这个缺省的构造函数不带任何参数，仅给对象开辟存储空间，不完成对数据成员赋初值。此时数据成员的值是随机的。系统自动生成的构造函数的形式为：

```
类名::类名()
{
```

```
}
```

8.2.2 析构函数

析构函数也是一种特殊的成员函数，也被声明为公有成员，其作用是释放分配给对象的内存空间，并做一些善后工作。

①关于析构函数有以下几点说明：

②析构函数的名字必须是 `~类名`。

③析构函数没有参数、没有返回值、不能重载。

④当对象撤销时，系统会自动调用析构函数完成内存空间的释放和善后工作。

⑤如果没有定义析构函数，系统会自动生成一个缺省的空析构函数。完成善后工作，其形式为：

```
类名::~类名()
```

```
{
```

```
}
```

对于构造函数和析构函数常见用法是在构造函数中用 `new` 动态申请空间，在析构函数中用 `delete` 释放内存空间。

8.2.3 拷贝构造函数

拷贝构造函数是一个特殊的构造函数，其作用是用一个已经存在的对象初始化本类的新对象。每个类都有一个拷贝构造函数，它可以是根据用户的需要自定义，也可以由系统自动生成。拷贝构造函数名与类名相同，但参数是本类对象的引用。拷贝构造函数没有返回值。定义拷贝构造函数的格式为：

```
类名(类名&对象名)
```

```
{
```

```
    //函数体
```

```
}
```

其中，对象名是用来初始化另一个对象的对象的引用。

构造函数只在对象被创建时自动调用，而拷贝构造函数在下列三种情况下会被自动调用：

①用一个对象去初始化本类的另一个对象时。

②函数的形参是类的对象，在进行形参和实参的结合时。

③函数的返回值是类的对象，函数执行完返回时。

8.2.4 对象成员

定义对象成员

当用一个类的对象作为另一个类的成员时，该成员称为对象成员。声明对象成员的一般格式为：

```
class 类名{
```

```
    类名 1 对象成员名 1; //需要此类在前面已经定义或声明。
```

```
    ...
```

```
};
```

对象成员的初始化

在类中有对象成员时，创建本类的对象则本类的构造函数要调用其对象成员所在类的构造函数，并采用成员初始化列表对对象成员进行初始化。这种类的构造函数的定义格式为：

```
类名::类名(参数总表): 对象成员 1(形参表), ..., 对象成员 n(形参表)
```

```
{
```

```
    //构造函数体
```

```
}
```

说明：

对象成员的构造函数的调用顺序由对象成员在类中的声明顺序决定，与成员初始化列表中的顺序无关。析构函数的调用顺序正好与构造函数的调用顺序相反。

8.3 对象数组与对象指针

8.3.1 对象数组

对象数组是指数组中的每个元素都是一个类的对象。当然这些对象属于同一个类。

✚ 定义一维对象数组的一般格式为：

类名 数组名[常量表达式];

✚ 对象数组的引用

由于对象数组的元素是对象，只能访问其公有成员。引用格式为：

数组名[下标]. 公有成员

8.3.2 对象指针

对象指针就是对象在内存中的首地址。指向类类型的指针变量用于存放对象指针。其定义格式为：

<类名> * <指针变量名>;

说明：

可以在定义的同时对该指针变量进行初始化即用“&对象名”的形式取出对象首地址赋给该变量，也可以在使用该指针变量时再对它赋值。

8.4 静态成员

静态成员是指类中用关键字 `static` 说明的那些成员。静态成员仍然服从访问控制。

8.4.1 静态数据成员

静态数据成员是指类中用关键字 `static` 说明的那些数据成员。静态数据成员属于类而不属于某个对象。它实现同类对象之间的数据共享。

✚ 在类中声明静态数据成员时，必须加 `static` 说明。

✚ 对静态数据成员初始化只能在类外进行，一般在在类声明与 `main()` 之间的位置。

格式为：

数据类型 类名::静态数据成员名=值;

✚ 对静态数据成员的引用可以有两种形式：

类名::静态数据成员

对象名. 静态数据成员

8.4.2 静态成员函数

静态成员函数是指类中用关键字 `static` 说明的那些成员函数。可以用静态成员函数在未建立任何对象之前去处理静态数据成员。静态成员函数只能直接引用该类的静态数据成员和静态成员函数，不能直接引用非静态数据成员。

调用静态的两种形式：

类名::静态函数名(); 或 对象名. 静态函数名();

8.5 友元

C++引入了友元实现了在类的外部访问类的私有成员的功能。这样，即不放弃私有数据的安全性，又可在类的外部访问类的私有成员。但一定程度上说友元破坏了类的封装性，在使用友元时一定要慎重。友元关系是单向的，也是不能传递的。

8.5.1 友元函数

一个普通函数作为某个类的友元时即为友元函数。在该函数中可以访问其由 `friend` 声明语句所在的类的对象的私有成员和公有成员。在类中作如下声明，则说明该函数不是本类的成员函数，而是友元函数。

`friend` 函数类型 友元函数名(参数表);

友元函数的定义可以在类内也可以在类外，在类外定义时不需要加类名和普通函数定义没有区别。通常友元函数的定义在类外进行。

友元函数不是类的成员，因而不能直接引用对象成员的名字，也不能通过 `this` 指针引用对象的成员，必须通过作为入口参数传递进来的对象名或对象指针来引用该对象的成员。为此，友元函数一般都带有一个该类的入口参数。

8.5.2 友元成员函数

某个类的成员函数作为另一个类的友元即为友元成员函数。通过友元成员函数，可以访问由 `friend` 声明语句所在的类的对象的私有成员和公有成员。

当一个类 A 的成员函数作为另一个类 B 的友元函数时，在类 B 中的声明格式为：

`friend 函数类型 成员函数所在类类名::函数名 (参数表);`

8.5.3 友类

当一个类作为另一个类的友元时即为友类。若类 A 是类 B 的友类，则类 A 中的所有成员函数都是类 B 的友元成员函数，所以可以通过对象名访问 B 的私有成员和公有成员。

当类 A 为类 B 的友类时，在类 B 中的声明格式为：

`friend class <友元类名>;` 或 `friend <友元类名>;`

【典型例题】

例题 1. 下列程序段是否有错，若有错请改错。

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

```
class point-----①
{
    private:
        int x,y;
    public:
        void setpoint(int, int);-----②
};
int point:: setpoint(int xx, int yy) -----③
{
    x=xx;-----④
    y=yy;-----⑤
    return 1;-----⑥
}
void main()
{
    point p1;-----⑦
    p1.setpoint(2,4);-----⑧
    cout<<"坐标为: ("<<p1.x<<","<<p1.y<<")"<<endl;-----⑨
}
```

解答：

这里⑨错误，不能在类定义以外直接访问类的私有成员。要得该点的两个坐标，应该在类中定义获取私有成员 x, y 的公有成员函数，使得可以在类外通过类的公有成员函数对其私有成员进行间接访问。所以程序应改为：

```
class point
{
    private:
        int x,y;
    public:
        void setpoint(int, int); //声明成员函数 setpoint()的函数原型
        int getx(); //声明成员函数 getx()的函数原型
        int gety(); //声明成员函数 gety 的函数原型
};
void point:: setpoint(int xx, int yy)//定义成员函数 setpoint()
{
    x=xx;
    y=yy;
}
```

```
int point::getx() //定义成员函数getx()
{   return x;}
int point::gety() //定义成员函数gety
{   return y;}
```

例题 2. 在下列函数原型中，可以作为类 student 的构造函数的说明的是（ ）。

(a)void student(int age); (b)int student(); (c)student(int)const; (d)student(int);

解答：

本题主要考查对构造函数的特点的掌握。构造函数的名字必须与类的名字相同。构造函数没有返回值，不能定义返回类型，包括 void 型在内。构造函数可以是内联函数，可带有参数表，可带有默认的形参值，还可重载。选项 a、b 均有返回值类型，不能作为构造函数。选项 c 为常成员函数，构造函数不能为常成员函数。答案为：d

例题 3. 下列说法正确的是（ ）。

- (a) 可以定义修改对象数据成员的 const 成员函数。
- (b) 不允许任何成员函数调用 const 对象，除非该成员函数也声明为 const。
- (c) const 对象可以调用非 const 成员函数。
- (d) const 成员函数可以调用本类的非 const 成员函数。

解答：

c++编译器不允许任何成员函数调用 const 对象，除非该成员函数本身也声明为 const。声明 const 的成员函数不能修改对象，因为编译器不允许其修改对象。对 const 对象调用非 const 成员函数是个语法错误。定义调用同一类实例的非 const 成员函数的 const 成员函数是个语法错误。答案为：b

例题 4. 运行下列程序后，”constructing A!”和”destructing A!”分别输出几次（ ）。

```
#include<iostream.h>
class A
{
    int x;
public:
    A()
    {cout<<" constructing A!"<<endl;}
    ~A()
    {cout<<" "<<endl;}
};
void main()
{
    A a[2];
    A *p=new A;
    delete p;
}
```

(a)2 次，2 次 (b)3 次，3 次 (c)1 次，3 次 (d)3 次，1 次

解答：

本题主要考查在什么情况下系统会调用构造函数与析构函数。在主函数中定义了一个对象数组，其中有两个元素，该数组中的每个元素都是一个类的对象，所以这里会调用 2 次构造函数；new A 时创建一个 A 类的对象，所以也会调用构造函数，因此一共调用 3 次构造函数。delete p;会撤消 new 运算分配的空间，它会调用 1 次析构函数。主函数结束时要释放数组所占空间，会调用 2 次析构函数，因此析构函数也调用了 3 次。答案为：b

例题 5. 运行下列程序的结果为_____。

```
#include<iostream.h>
#include<string.h>
class course
{
```

```

    int id;
    char name[50];
public:
    course(int csid,char *csname)
    {
        cout<<"constructing course!"<<endl;
        id=csid;
        strcpy(name,csname);
    }
    ~course()
    {
        cout<<"destructing course!"<<endl;
    }
    int getid()
    {return id;}
    char* getname()
    {
        return name;
    }
};
class student
{
    char name[10];
    int age;
    course c1;
public:
    student(char *sname,int sage,int cid,char *cname):c1(cid,cname)
    {
        cout<<"constructing student!"<<endl;
        strcpy(name,sname);
        age=sage;
    }
    ~student()
    {
        cout<<"destructing student!"<<endl;
    }
    void prints()
    {
        cout<<"name:"<<name<<endl
            <<"age:"<<age<<endl
            <<"course id:"<<c1.getid()<<endl
            <<"course name:"<<c1.getname()<<endl;
    }
};

void main()
{
    student st1("tom",23,1,"c++ programming language");
    st1.prints();
}

```

解答:

本题主要考查在为含有对象成员的类创建对象时，构造函数的调用顺序，对象成员的初始化问题以及对象撤消时调用析构函数的顺序。对于本程序，在主函数中创建 `student` 类的对象则调用其构造函数 `student()`，该构造启动时，首先为数据成员分配空间，然后根据在类中声明的对象成员的顺序依次调用其构造函数，在这里调用 `course` 类的构造函数，最后才执行自己的构造函数的函数体。析构函数以调用构造函数相反的顺序被调用。

答案为:

```
constructing course!
constructing student!
name:tom
age:23
course id:1
course name:c++ programming language
destructing student!
destructing course!
```

例题 6. 运行下列程序输出结果为_____。

```
#include<iostream.h>
class A{
public:
    A(int X){cout<<"ok!";}
    A(){}
};
int main()
{
    A a[3],a1(3);
    return 0;
}
```

解答:

本题主要考查对重载构造函数的理解。这里创建对象数组时,对数组的每一个元素都将调用一次构造函数,如果没有显式给出数组元素的初值,则调用缺省构造函数。而创建对象 a1 时带有一个整型参数,所以调用以整型作为参数的构造函数,它输出 ok!。所以,本题答案为: ok!

例题 7. 运行下列程序结果为_____。

```
#include<iostream.h>
const double PI=3.14159;
class circle
{
    double r;

public:
    static int num;
    circle(double);
    circle(circle &);
    double getr();
};
circle::circle(double i)
{
    r=i;
}
circle::circle(circle &c)
{
    num++;
    cout<<"第"<<num<<"次调用拷贝构造函数! "<<endl;
    r=c.r*num;
}
double circle::getr()
{
    return r;
}
double getradius(circle c3)
```



```

{
    return c3.getr();
}
circle fun1()
{
    circle c4(5);return c4;
}
int circle::num=0;
void main()
{
    circle c1(1);
    cout<<"c1:"<<c1.getr()<<endl;
    circle c2(c1);
    cout<<"c2:"<<c2.getr()<<endl;
    cout<<"c3:"<<getradius(c1)<<endl;
    circle c4(1);
    c4=fun1();
    cout<<"c4:"<<c4.getr()<<endl;
}

```

解答:

本题主要考查在什么情况下会调用拷贝构造函数。构造函数只在对象被创建时自动调用，而拷贝构造函数在下列三种情况下会被自动调用：

- ①用一个对象去初始化本类的另一个对象时。
- ②函数的形参是类的对象，在进行形参和实参的结合时。
- ③函数的返回值是类的对象，函数执行完返回时。

本题答案为：

c1:1

第 1 次调用拷贝构造函数！

c2:1

第 2 次调用拷贝构造函数！

c3:2

第 3 次调用拷贝构造函数！

c4:15

例题 8. 读程序写结果或者程序填空。

```

#include<iostream.h>
class A
{
    const int i;
    int &j;
public:
    A(int& var):i(10),j(var)
    {}
    void show()
    {
        cout<<"i:"<<i<<endl
        <<"j:"<<j<<endl;
    }
};
void main()
{
    int x=1;
    A a1(x);
    a1.show();
}

```

解答:

本题主要考查对符号常量和引用的理解。常量是不能被赋值的,一旦初始化后,其值就永不改变,引用变量也是不可重新指派的,初始化后,其值就固定不变了。

结果为:

i:10

j:1

例题 9. 运行下列程序结果为_____。

```
#include<iostream.h>
class Obj{
    static int i;
public:
    Obj(){i++;}
    ~Obj(){i--;}
    static int getVal(){return i;}
};
int Obj::i=0;
void f(){Obj ob2;cout<<ob2.getVal();}
int main(){
    Obj ob1;
    f();
    Obj*ob3=new Obj;cout<<ob3->getVal();
    delete ob3;cout<<Obj::getVal();
    return 0;
}
```

解答:

本题主要考查对静态数据成员的理解。在主函数中创建对象 ob1 则调用该类的构造函数,使得静态数据成员加 1,为 1;接着调用函数 f(),在函数中创建对象 ob2,这时再次调用构造函数,使得静态成员的值变为 2,ob2.getVal()返回静态数据成员 i 的值,即输出 2。函数 f() 结束,则 ob2 的生存期结束,自动调用其析构函数使静态数据成员 i 的值变为 1。接着在主函数中用 new 运算符动态分配存储空间,又一次调用构造函数使 i 加 1,所以再次输出时 i 的值为 2。最后用 delete 释放 ob3 所指的对象空间,则会调用析构函数使 i 的值减 1,因此输出 i 的值为 1。本题答案为: 221

例题 10. 若类 A 是类 B 的友元,类 B 是类 C 的友元,则下列说法正确的是 ()。

- (a)类 B 是类 C 的友元 (b)类 A 是类 C 的友元
(c)类 A, B, C 互为友元 (d)以上说法都不对

解答:

本题考查对友元关系的理解。友元关系是单向的,也是不能传递的。答案为: a

例题 11. 当输入为 2 3 时,下列程序输出“两个数的和为: 5”。请将程序补充完整。

```
#include<iostream.h>
class num
{
    int x,y;
public:
    num(int=0,int=0);
    _____①_____
};
num::num(int x,int y)
{
    _____②_____
    _____③_____
}
```

```

}
int sum(num& n)
{
    return n.x+n.y;
}
void main()
{
    int i,j;
    cout<<"请输入两个数: "<<endl;
    cin>>i>>j;
    _____④_____//定义对象 num1
    cout<<"两数的和为: "<<sum(num1)<<endl;
}

```

解答:

本题主要考查友元的应用以及对不同作用域变量的引用方法。

答案为: ①friend int sum(num&);②num::x=x;③num::y=y;④num num1(i,j);

【习题】

一、选择题

1. 下列各项中不能用于声明类的成员访问控制权限的关键字是 ()。

(a)private (b)protected (c)public (d)static

2. 下列关于构造函数的说法错误的是 ()。

(a)构造函数的名字必须与类的名字相同。

(b)构造函数可以定义为 void 类型。

(c)构造函数可以重载、可以带有默认参数。

(d)构造函数可以由用户自定义也可以由系统自动生成。

3. 有如下类声明:

```
class student
```

```
{
```

```
    int age;
```

```
    char *name;
```

```
};
```

则 student 类的成员 age 是 ()。

(a)公有数据成员

(b)私有数据成员

(c)保护数据成员

(d)私有成员函数

4. 有如下类定义

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

```
class point
```

```
{
```

```
int x,y;
```

```
public:
```

```
point():x(0),y(0){}
```

```
point(int x1,int y1=0):x(x1),y(y1){}
```

```
};
```

若执行语句

```
point a(2),b[3],*c;
```

则 point 类的构造函数被调用的次数是 ()。

(a)2 次

(b)3 次

(c)4 次

(d)5 次

5. 在下列哪种情况下不会调用拷贝构造函数 ()。

(a)用一个对象去初始化本类的另一个对象时。

(b)函数的形参是类的对象,在进行形参和实参的结合时。

(c)函数的返回值是类的对象，函数执行完返回时。

(d)将类的一个对象赋值给另一个本类的对象时。

6. 下列关于友元的描述错误的是（ ）。

(a) 友元关系是单向的且不可传递

(b) 在友元函数中可以通过 **this** 指针直接引用对象的私有成员。

(c) 友元可以是一个普通函数也可以是一个类。

(d) 通过友元可以实现在类的外部对类的私有成员的访问。

7. 有如下程序

```
#include <iostream>
using namespace std;
class AA{
    int n;
public:
    AA(int k):n(k){ }
    int get( ){ return n;}
    int get( )const{ return n+1;}
};
int main( )
{
    AA a(5);
    const AA b(6);
    cout<<a.get( )<<b.get( );
    return 0;
}
```

运行该程序结果为（ ）。

(a)56 (b)57 (c)67 (d)66

8. 有如下程序：

```
#include <iostream.h>
class Test {
public:
    Test( ) { n+=2; }
    ~Test( ) { n-=3; }
    static int getNum( ) { return n; }
private:
    static int n;
};
int Test::n = 1;
int main( )
{
    Test* p = new Test;
    delete p;
    cout << "n=" << Test::getNum( ) << endl;
    return 0;
}
```

执行后的输出结果是（ ）。

(a) n=0 (b)n=1 (c)n=2 (d)n=3

9. 下列程序的运行结果为（ ）。

```
#include<iostream.h>
class A
{
public:
    static int num;
    A& fun()
    {
        num++;
        return *this;
    }
}
```

```

    }
};
int A::num=0;
void g(A& a)
{
    cout<<a.fun ().num<<endl;
}
void main()
{
    A a1;
    g(a1);
}

```

(a)0 (b)1 (c)2 (d)3

10. 运行下列程序结果为_____。

```

#include<iostream>
#include<iomanip>
using namespace std;
class MyClass{
public:
    MyClass(){cout<<'A';}
    MyClass(char c){cout<<c;}
    ~MyClass(){cout<<'B';}
};
int main(){
    MyClass p1,*p2;
    p2=new MyClass('X');
    delete p2;
    return 0;
}

```

执行这个程序幕上将显示输出_____。

(a)ABX (b)ABXB (c)AXB (d)AXBB

二、填空题

1. 类的成员包括_____成员和成员_____。
2. 释放对象所占的内存空间并完成善后处理工作的是_____函数。
3. 拷贝构造函数以_____作为参数。
4. 用指向对象的指针引用对象成员使用操作符_____。
5. 当一个对象生成以后，系统就为这个对象定义了一个_____，它指向这个对象的地址。
6. 在类中声明静态成员的关键字是_____。
7. 非成员函数应声明为类的_____函数才能访问这个类的 **private** 成员。
8. C++建立和初始化对象的过程由_____完成。
9. 对于常量数据成员和引用数据成员的初始化只能通过_____来完成。
10. 在类中说明的具有类类型的成员称为_____。
11. 下列为类的定义语句，是否有错，若有错请改正。

```

class circle -----①
{
    double r=3;-----②
public:-----③
    circle(double i) -----④
    {
        r=i; -----⑤
    }
    double area();//面积-----⑥
}

```

```

        double prm();//周长-----⑦
        void printarea(double); -----⑧
        void printprm(double); -----
    }-----⑩
//成员函数的实现

```

12. 下列程序输出结果为 0,1.请将程序补充完整。

```

#include<iostream.h>
class A{
    int num;
public:
    A():num(0){}
    void set(int num)
    { _____①_____} //给 A 的数据成员 num 赋值
    int get()
    { _____②_____} //获得数据成员 num 的值
};
int main()
{
    A a;
    cout<<a.get()<<" ";
    a.set(1);
    cout<<a.get()<<endl;
    return 0;
}

```

13. 下列程序输出： 2, 33.4, tom。请将程序补充完整。

```

#include<iostream.h>
#include<string.h>
class A
{
    int i;
    float j;
    char c[20];
public:
    A(int x, float y, char ch[]):i(x),j(y)
    {
        _____①_____ //初始化成员 c 赋值
    }
    void printA()
    {
        _____②_____ //输出三个私有数据成员
    }
};
void main()
{
    A a(2,33.4,"tom");
    a.printA();
}

```

14. 下列程序的输出结果为

Object id=0

Object id=1

请将程序补充完整。

```

#include <iostream.h>
class Point
{
public:

```

```

    Point(int xx=0, int yy=0) {X=xx; Y=yy; countP++; }
    Point() { countP--; }
    int GetX() {return X;}
    int GetY() {return Y;}
    static void GetC() {cout<<" Object id="<<countP<<endl;}
private:
    int X,Y;
    static int countP;
};
_____//静态数据成员的初始化

```

```
int main( )
```

```

{
    Point::GetC( );
    Point A(4,5);
    A.GetC( );
    return 0;
}

```

15. 插入排序算法的主要思想是：每次从未排序序列中取出一个数据，插入到已排序序列中的正确位置，InsertSort 类的成员函数 sort()实现了插入排序算法，请将画线处缺失的部分补充完整。

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

```
class InsertSort{
```

```
public:
```

```
    InsertSort(int*a0,int n0):a(a0),n(n0){} //a 是数组首地址， n 是数组元素个数
```

```
    void sort()
```

```
    { //此函数假设已排序序列初始化状态只包含 a[0]，未排序序列初始为 a[1]...a[n-1]
```

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

```
        {
```

```
            int j,t;
```

```
            for(_____;j>0;--j)
```

```
            {
```

```
                if(t>a[j-1])break;
```

```
                a[j]=a[j-1];
```

```
            }
```

```
            a[j]=t;
```

```
        }
```

```
    }
```

```
protected:
```

```
    int*a,n; //指针 a 用于存放数组首地址， n 用于存放数组元素个数
```

```
};
```

三、编程题

1. 自定义一个正方体类，它具有私有成员 x，表示正方体的每个面的正方形的边长。提供构造函数以及计算正方体的体积和表面积的公有成员函数，并编制主函数，对正方体类进行使用：说明正方体类对象，输入棱长，计算其体积和表面积并显示结果。

2. 设计一个时间类 Time，包括 3 个数据成员，时 (hour)、分 (minute)、秒 (second)，以及成员函数用于设置和读取时、分、秒，并按上午、下午各 12 小时或按 24 小时输出时间。

【参考答案】

一、选择题

1. d
2. b
3. b
4. c

- 5. d
- 6. b
- 7. b
- 8. a
- 9. b
- 10. d

二、填空题

- 1. 数据、函数
- 2. 析构
- 3. 本类对象的引用
- 4. ->
- 5. this 指针
- 6. static
- 7. 友元
- 8. 类的构造函数
- 9. 成员初始化列表
- 10. 对象成员
- 11. ②⑩ 改正如下:
 - ②double r;
 - ⑩末尾加分号。
- 12. ①A::num=num; ②return num;
- 13.
 - ①strcpy(c,ch);
 - ②cout<<i<<","<<j<<","<<c<<endl;
- 14. int Point::countP=0;
- 15. j=i,t=a[i]

三、编程题

1.参考程序如下:

```
#include<iostream.h>
#include<math.h>
class cube
{
    double x;
public:
    cube(double xx)
    {
        x=xx;
    }
    double volume();
    double sarea();
};

double cube::volume()
{
    return pow(x,3);
}
double cube::sarea()
{
    return pow(x,2)*6;
}
void main()
{
    double a;
```



```

        cout<<"请输入棱长： ";
        cin>>a;
        cube c(a);
        cout<<"该正方体的体积为： "<<c.volume ()<<endl
            <<"该正方体的表面积为： "<<c.sarea()<<endl;
    }

```

2.参考程序如下：

```

#include<iostream.h>

```

```

class Time

```

```

{
    int hour;
    int minute;
    int second;

```

```

public:

```

```

    int sethour(int);
    int setminute(int);
    int setsecond(int);
    int gethour();
    int getminute();
    int getsecond();
    void show12();
    void show24();

```

```

};

```

```

int Time::sethour(int h)

```

```

{
    if(h>=24||h<0)
        return 0;
    else
    {
        hour=h;
        return 1;
    }
}

```

```

}

```

```

int Time::setminute(int m)

```

```

{
    if(m>=60||m<0)
        return 0;
    else
    {
        minute=m;
        return 1;
    }
}

```

```

}

```

```

int Time::setsecond(int s)

```

```

{
    if(s>=60||s<0)
        return 0;
    else
    {
        second=s;
        return 1;
    }
}

```

```

}

```

```

int Time::gethour(){return hour;}

```

```

int Time::getminute(){return minute;}

```

```

int Time::getsecond(){return second;}

```

```

void Time::show12()
{
    if(hour==12)
        cout<<hour<<":"<<minute<<":"<<second<<"AM"<<endl;
    else
        cout<<hour%12<<":"<<minute<<":"<<second
            <<((hour<12&&hour>=0)?"AM":"PM")<<endl;
}
void Time::show24()
{
    cout<<hour<<":"<<minute<<":"<<second<<endl;
}

void main()
{
    int h,m,s;
    cin>>h>>m>>s;
    Time t;
    if(t.sethour(h)==0||t.setminute(m)==0||t.setsecond(s)==0)
    {
        cout<<"输入时间错误！";
        return;
    }
    t.show12();
    t.show24();
}

```

第9章 继承性与派生类

【内容提要】

我们知道，软件工程的一个重要旋律就是“复用”，说直白点，就是如何重复使用已经开发的功能，软件的开发一定是站在前人的肩膀上前行的。不然，每个软件都平地而起，我们哪来的那么多软件可用。继承正是解决面向对象程序“复用”的关键所在，以下是其内容：

① **继承性与派生类的基本概念**：继承和派生类似于“父与子”的关系，子（派生类）会继承父（基类）的特性。

② **派生类的声明和访问权限**：掌握语法和规则

③ **派生类构造函数和析构函数的定义及使用**：明白了现有父，再有子的道理，就应该知道子类的构造之前，要先构造基类，析构的是否顺序相反。

④ **多重继承的声明、构造函数和析构函数的定义及使用**：多重继承具有传递性，即子类只需考虑其父类的构造和析构，其“祖父类”有其“父类”负责，无需多管。

⑤ **虚基类的作用、定义和使用**：基类来自于多个，而且多个基类来自于同一个“祖宗”的时候，为了避免各自创建“祖宗”，产生多个“祖宗”，因此需要加 `virtual` 来予以限定，只产生一个“祖宗”。

【重点与难点】

9.1 继承性与派生类的基本概念

继承是软件复用的一种形式，它是从现有类的基础上建立新类，新类继承了现有类的属性和方法，并且还拥有其特有的属性和方法。继承的过程称为派生，新建的类为派生类（或子类），原有的类称为基类（或父类）。继承可分为：单继承和多重继承。若派生类只有一个基类则称为单继承；若派生类有多个基类则称为多重继承。

9.2 派生类的声明与访问权限

9.2.1 派生类的声明

单继承中派生类的定义格式为：

```
class <派生类名>: <派生方式><基类名>
{
    派生类新定义的成员声明;
};
```

说明：

①派生方式关键字为 `private`、`public` 和 `protected`，分别表示私有继承、公有继承和保护继承。缺省的继承方式是私有继承。继承方式规定了派生类成员和类外对象访问基类成员的权限。

②派生类新定义的成员是指继承过程中新增加的数据成员和成员函数。通过在派生类中新增加成员实现功能的扩充。

9.2.2 派生类的访问权限

公有继承（public）

①继承后基类的公有成员、私有成员、保护成员在派生类中访问权限保持不变。

②在派生类中可以直接访问基类的公有成员和保护成员，但对于私有成员的访问只能通过基类的非私有成员函数间接访问。

- ③在基类和派生类定义以外只能通过派生类的对象访问基类的公有成员,无法通过派生类对象直接访问基类的私有成员和保护成员。

✚ 私有继承 (private)

- ①继承后基类的所有成员在派生类中均为私有成员。
②在派生类中可以直接访问基类的公有成员和保护成员,但对于私有成员的访问只能通过基类的非私有成员函数间接访问。
③在基类和派生类定义以外对基类的所有成员均无法直接访问也无法通过派生类的对象间接访问。

✚ 保护继承 (protected)

- ①继承后基类的公有成员和保护成员在派生类中均为保护成员,基类的私有成员在派生类中仍为私有成员。
②在派生类中可以直接访问基类的公有成员和保护成员,但对于私有成员的访问只能通过基类的非私有成员函数间接访问。
③在基类和派生类定义以外对基类的所有成员均无法直接访问也无法通过派生类的对象间接访问。

9.3 派生类构造函数和析构函数的定义及使用

在派生过程中,构造函数和析构函数不被继承。在创建一个派生类对象时,分别调用基类和派生类的构造函数,完成各自成员的初始化工作。当撤销一个派生类对象时,分别调用基类和派生类的析构函数完成善后处理工作。

✚ 在 C++ 中对构造函数与析构函数的调用顺序有如下规定:

- ①对于构造函数,先执行基类的,再执行对象成员的,最后执行派生类的。
②对于析构函数,先执行派生类的,再执行对象成员的,最后执行基类的。

✚ 派生类构造函数定义格式为:

<派生类名>::<派生类名>(参数总表):基类名(参数表),对象成员名 1(参数表 1),...,对象成员名 n(参数表 n)

```
{  
    //派生类新增成员的初始化语句  
}
```

说明:

- ①派生类的构造函数名与派生类名相同。
②参数总表列出初始化基类成员数据、新增对象成员数据和派生类新增数据成员所需要的全部参数。
③冒号后列出需要使用参数进行初始化的基类的名字和对象成员的名字及各自的参数表,之间用逗号分开。对于使用缺省构造函数的基类或对象成员,可以不给出类名或对象名以及参数表。
④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,全都采用缺省的构造函数。如果基类定义了带形参表的构造函数,派生类就必须定义构造函数,保证在基类进行初始化时能获得所需的数据。
⑤如果派生类的基类也是派生类,则每个派生类只需负责其直接基类的构造,不负责其间接基类的构造。

9.4 多重继承的声明、构造函数和析构函数的定义及使用

9.4.1 多重继承的声明

多重继承声明的格式为:

```
class <派生类名>: <派生方式 1><基类名 1>,...,<派生方式 n><基类名 n>  
{  
    派生类成员声明;  
};
```

说明: 这里的派生方式以及访问权限定义与单继承中规则相同。

9.4.2 多重继承的构造函数与析构函数

✚ 多重继承中对构造函数和析构函数的调用顺序的规定：

①对于构造函数，先执行基类的，再执行对象成员的，最后执行派生类的。多个基类构造函数的执行次序严格按照声明时从左到右的顺序来执行的，与定义派生类构造函数时指定的初始化表中的次序无关。多个对象成员所在类的构造函数的执行次序按照对象成员定义的顺序来执行。

②析构函数的调用顺序正好与构造函数的调用顺序相反。

✚ 定义多重继承构造函数的格式为：

<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),对象成员名 1(对象成员参数表 1),...,对象成员名 m(对象成员参数表 m)

```
{  
    //派生类新增成员的初始化语句  
}
```

说明：单继承中构造函数定义的说明在多重继承构造函数中均适用。

9.5 虚基类的作用、定义和使用

9.5.1 多重继承中的二义性问题

✚ 问题的产生：

①当派生类继承的多个基类中存在同名成员时，派生类中就会出现来自不同基类的同名成员，就出现了标识符不唯一或二义性的情况，这在程序中是不允许的。

②当一个类从多个基类派生而来，这多个基类又有共同的基类，则在派生类中访问这个共同基类中的成员时会产生二义性。

✚ 解决办法：

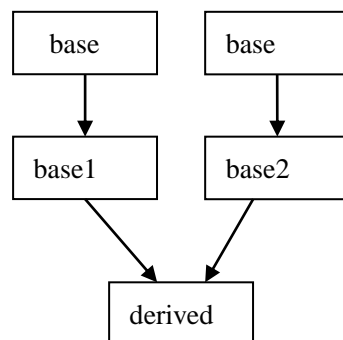
对于第一种情况：

- ①使用作用域运算符“::”
- ②使用同名覆盖的原则
- ③使用虚函数(在下一章中介绍)

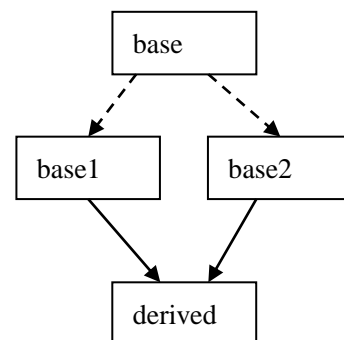
对于第二种情况：

使用虚基类

9.5.2 虚基类



非虚基类的情况



虚基类的情况

如上图所示，对于非虚基类的情况，derived 有两个基类 base1 和 base2，这两个基类又有共同基类 base，derived 类中就有基类 base 的两个不同的拷贝。在 derived 要访问 base 类中的成员时，就会产生二义性问题。虚基类就是为了解决这个问题而引入的。如上图所示，对于虚基类的情况，derived 中的公共基类 base 就只有一个拷贝而不会出现二义性问题。

9.5.3 虚基类的定义

虚基类的声明是在派生类的声明过程中进行的，格式为：

class<派生类名>:virtual <派生方式><基类名>

说明：

①虚基类关键字的作用范围和派生方式与一般派生类的一样，只对紧跟其后的基类起作用。

②声明了虚基类以后，虚基类的成员在进一步派生过程中和派生类一起维护同一个内存

拷贝。

9.5.4 虚基类的构造函数和初始化

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，但构造函数的执行顺序不同：

- ①虚基类的构造函数在非虚基类的构造函数之前执行。
- ②若同一层次中包含多个虚基类，这些虚基类的构造函数按它们说明的先后次序执行。
- ③ 若虚基类由非虚基类派生而来，则仍然先执行基类的构造函数，再执行派生类的构造函数。

【典型例题】

例题 1. 请将下列类定义补充完整。

```
#include<iostream.h>
class base{
public:
    void fun()
    {
        cout<<"base::fun"<<endl;
    }
};
class derived : public base {
public:
    void fun() {
        _____//显式调用基类的 fun 函数
        cout<<"derived::fun"<<endl;
    }
};
```

解答：

本题考查在继承过程中，如果基类与子类有同名成员时，如何完成各自的引用。如果基类与子类有同名成员时，子类的同名成员会屏蔽基类的同名成员，所以要在自类中引用该成员则需要使用作用域区分符确定要调用谁的成员。因此，本题答案为：base::fun();

例题 2. 有如下程序：

```
#include<iostream.h>
class base
{
public:
    void show(){cout<<"base:public member"<<endl;}
protected:
    void show1(){cout<<"base:protected member"<<endl;}
private:
    void show2(){cout<<"base:private member"<<endl;}
};
class derived:protected base
{
public:
    void fn()
    {
        show1();//①
        show2();//②
    }
};

void main()
```

```

{
    derived a;
    a.fn();
    a.show(); //③
    a.show1(); //④
    show(); //⑤
}

```

有语法错误的语句是 ()。

(a) ①②③④ (b) ②③④⑤ (c) ①③④⑤ (d) ①②④⑤

解答：

本题主要考查各种派生中派生类的访问权限问题。这里 `derived` 采用保护继承的方式继承了 `base` 类。对于保护继承其访问权限有如下规则：

- ① 继承后基类的公有成员和保护成员在派生类中均为保护成员，基类的私有成员在派生类中仍为私有成员。
- ② 在派生类中可以直接访问基类的公有成员和保护成员，但对于私有成员的访问只能通过基类的非私有成员函数间接访问。
- ③ 在基类和派生类定义以外对基类的所有成员均无法直接访问也无法通过派生类的对象间接访问。

对于语句①②是在派生类内部访问基类的保护成员函数和私有成员函数，无论哪种继承方式，基类的私有成员都不能被基类定义以外的任何地方直接访问，所以语句②是错误用法；而在派生类定义的内部访问基类的公有和保护成员是允许的，所以语句①正确。对于语句③④是在类定义以外通过子类的对象访问基类的公有成员函数和保护成员函数，因为保护继承后基类的公有和保护成员在子类中均为保护成员，所以在类外通过对象不能对其直接进行访问，所以③④语句是错误的用法。语句⑤执行后找不到相应的函数定义，因此是错误的。答案为：b

例题 3. 下列说法正确的是 ()。

- (a) 基类的构造函数和析构函数不能被派生类继承。
- (b) 在派生类中用户必须自定义派生类构造函数。
- (c) 析构函数与构造函数被调用的顺序是一致的。
- (d) 在多重继承中，多个基类的构造函数的调用顺序由定义派生类构造函数时指定的初始化表中的次序决定。

解答：

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，全都采用缺省的构造函数。如果基类定义了带有形参表的构造函数，派生类就必须定义构造函数，保证在基类进行初始化时能获得所需的数据。析构函数与构造函数被调用的顺序正好相反。在多重继承中，多个基类的构造函数的调用顺序由在定义派生类时基类的声明顺序决定。答案为：a

例题 4. 有如下程序：

```

#include <iostream.h>
class A {
public:
    A() { cout << "A"; }
};
class B {
public:
    B() { cout << "B"; }
};
class C : public A {
    B b;
public:
    C() { cout << "C"; }
};

```

```
int main( )
{
    C obj;
    return 0;
}
```

执行后的输出结果是 ()。

(a)CBA (b)BAC (c)ACB (d)ABC

解答：

本题主要考查继承中构造函数的调用顺序。对于构造函数，先执行基类的，再执行对象成员的，最后执行派生类的。本程序中在主函数里创建 C 类对象 obj，则首先执行类 C 的基类 A 的构造函数输出 A，然后调用其对象成员 b 所在的 B 类的构造函数输出 B，最后执行类 C 自己的构造函数输出 C。答案为：d

例题 5. 有如下程序：

```
#include <iostream.h>
class base1
{public:
    base1() {cout<<"base1 constructing"<<endl;}
    ~base1() {cout<<"base1 destrutcting"<<endl;}
};
class base2
{public:
    base2() {cout<<"base2 constructing"<<endl;}
    ~base2() {cout<<"base2 destrutcting"<<endl;}
};
class base3
{public:
    base3() {cout<<"base3 constructing"<<endl;}
    ~base3() {cout<<"base3 destrutcting"<<endl;}
};
class derive:public base1,virtual public base2,virtual public base3
{
    public:
    derive() {cout<<"constructing derive"<<endl;}
    ~derive() {cout<<"destructing derive"<<endl;}
};
void main()
{
    derive d1;
}
```

运行该程序结果为_____。

解答：

本题主要考查含虚基类的继承关系中构造函数的执行顺序。虚基类的构造函数在非虚基类的构造函数之前执行；若同一层次中包含多个虚基类，这些虚基类的构造函数按它们说明的先后次序执行。答案为：

```
base2 constructing
base3 constructing
base1 constructing
constructing derive
destructing derive
base1 destructing
base3 destructing
```


base2 destructing

例题 6. 运行下列程序的结果为 ()。

```
#include <iostream.h>
class A
{
    int a;
public:
    A(int i) {a=i;}
    void print(){cout<<a;}
};
class B1:virtual public A
{
    int b1;
public:
    B1(int i,int j):A(i)
    {b1=j;}
    void print()
    {
        cout<<b1;
    }
};
class B2:virtual public A
{
    int b2;
public:
    B2(int i,int j):A(i)
    {b2=j;}
    void print()
    {
        cout<<b2;
    }
};
class C:public B1,public B2
{
    int c;
public:
    C(int j,int k,int l,int m):A(l),B1(l,j),B2(k,l),c(m){ }
    void print()
    {
        A::print();
        B1::print();
        B2::print();
        cout<<c;}
};
void main()
{
    C c1(1,2,3,4);
    c1.print();
}
```

(a)3124 (b)3214 (c)1234 (d)3134

解答:

本题主要考查继承关系中对象数据的初始化问题。

C(int j,int k,int l,int m):A(l),B1(l,j),B2(k,l),c(m){ } 这条语句中, 用 l 初始化成员类 A 的成员 a; 用 j 初始化类 B1 的成员 b1; 用 l 初始化类 B2 的成员 b2, 用 m 初始化类 C 的成员 c。答案为: d

例题 7. 编写一个程序声明一个正方形类、一个桌子类和一个方桌类，方桌类由桌子类和正方形类派生，定义一格方桌类对象，并输出该桌子的颜色，价格，以及桌面尺寸。

解答：

这是一道简单的编程题，主要考查对继承机制的应用。这里方桌类由正方形类和桌子类派生而来，所以采用多重继承编写此题。

参考程序如下：

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

class square
{
    double edgelenh;
public:
    square(double i)
    {
        edgelenh=i;
    }
    double getedge()
    {
        return edgelenh;
    }
};
class table
{
    double price;
public:
    table(double p)
    {
        price=p;
    }
    double getprice()
    {
        return price;
    }
};
class squaretable:public square,public table
{
    char color[10];
public:
    squaretable(double el,double prc,char *c):square(el),table(prc)
    {
        strcpy(color,c);
    }
    void disp()
    {
        cout<<"颜色:"<<color<<endl
            <<"价格:"<<getprice()<<endl
            <<"桌面尺寸:"<<getedge()<<"*"<<getedge()<<endl;
    }
};
void main()
{
    squaretable st(60,1234,"red");
```

```
        st.disp();  
    }
```

【习题】

一、选择题

1. 在私有继承的情况下，允许派生类直接访问的基类成员包括（ ）。

- (a)公有成员和私有成员 (b)公有成员和保护成员
(c)保护成员和私有成员 (d)公有成员、私有成员和保护成员

2. 有如下类定义：

```
class base{  
public: int x;  
protected: int y;  
private: int z;  
};
```

派生类采用什么方式继承可以使 x 成为自己的公有成员（ ）。

- (a)公有继承 (b)保护继承 (c)私有继承 (d)以上三个都对

3. 派生类的对象对其基类成员中（ ）是可以访问的。

- (a)公有继承中的公有成员 (b)公有继承中的保护成员
(c)私有继承中的公有成员 (d)以上三者都对

4. 下列虚基类的声明正确的是（ ）。

- (a)class derived:virtual public base
(b)virtual class derived:public base
(c)class virtual derived:base
(d)class derived: base1 virtual

5. 有如下类声明：

```
class base{  
    int i;  
public:  
    void set(int n){ i=n;}  
    int get( )const{ return i; }  
};  
class derived: protected base{  
protected:  
    int j;  
public:  
    void set(int m, int n){ base::set(m); j=n;}  
    int get( )const{ return base::get( )+j; }  
};
```

则类 derived 中保护的数据成员和成员函数的个数是（ ）。

- (a)1 (b)2 (c)3 (d)4

6. 有如下程序：

```
#include <iostream.h>  
class base  
{  
    int n;  
public:  
    base(int a,int b,int c)
```

```

        {
            n=a;x=b;y=c;
        }
protected:
    int x,y;

};
class derived:public base
{
public:
    int m;
    derived(int a,int b,int c,int d):base(a,b,c){m=d;}
};
void main()
{
    derived s(1,2,3,4);//①
    cout<<s.n<<endl;//②
    cout<<s.x<<s.y<<endl;//③
    cout<<s.m<<endl;//④
}

```

有语法错误的语句是 ()。

- (a)①② (b)②③ (c)③④ (d)①④

7. 有如下类声明:

```

class base{
    int x;
public:
    base(int n){ x=n;}
};
class derived: public base{
    int y;
public:
    derived(int a,int b);
};

```

下列对构造函数 derived 的定义, 正确的是 ()。

- (a) derived::derived(int a,int b):base(a),y(b){}
 (b) derived::derived(int a,int b):x(a),y(b){}
 (c) derived::derived(int a,int b):base(a),derived(b){}
 (d) derived::derived(int a,int b):x(a),derived(b){}

8. 在有公派生的情况下, 有关派生类对象和基类对象的关系的关系, 不正确的叙述是 ()。

- (a)派生类的对象可以赋给基类对象
 (b)派生类的对象可以初始化基类的引用
 (c)派生类的对象可以直接访问基类中的成员
 (d)派生类的对象的地址可以赋给指向基类的指针

二、填空题

- 在继承中, 缺省的继承方式是_____。
- 派生类中的成员函数不能直接访问基类中的_____成员。
- 保护派生时, 基类中的所有非私有成员在派生类中是_____成员。
- 当创建一个派生类对象时, 先调用_____的构造函数, 然后调用_____的构造函数, 最后调用_____的构造函数。
- 对于基类数据成员的初始化必须在派生类构造函数中的_____处执行。
- 为了解决在多重继承中因公共基类带来的_____问题, C++语言提供了虚基类机制。

7. 将下列的类定义补充完整。

```
class base
{
public:
    int f();
};
class derived:public base
{
    int f();
    int g();
};
void derived::g()
{
    f();    //被调用的函数是 derived:: f()
           _____ //调用基类的成员函数 f
}
```

8. 有如下程序:

```
#include <iostream.h>
class base{
public:
    base(){cout<<"constructing base!"<<endl;}
    ~base(){ cout<<"destructing base!"<<endl;}
};
class derived: public base{
public:
    derived(){cout<<"constructing derived!"<<endl;}
    ~derived(){ cout<<"destructing derived!"<<endl;}
};
int main()
{
    derived x;
    return 1;
}
```

运行结果为_____。

9. 有如下程序:

```
#include<iostream.h>
class base
{
protected:
    int x;
public:
    base(int x1)
    { x=x1;cout<<"x="<<x<<endl;}
};
class base1:virtual public base
{
    int y;
public:
    base1(int x1,int y1):base(x1)
    { y=y1;cout<<"y="<<y<<endl;}
};
class base2:virtual public base
{ int z;
public:
    base2(int x1,int z1):base(x1)
```

```

    { z=z1;cout<<"z="<<z<<endl;}
};
class derived:public base1,public base2
{ int xyz;
public:
    { xyz=xyz1;cout<<"xyz="<<xyz<<endl;}
};
void main()
{derived obj(1,2,3,4);}
输出结果为:

```

x=1

y=2

z=3

xyz=4

请将程序补充完整。

10. 运行下列程序的结果为_____。

```

#include <iostream.h>
class vehicle
{
    int wheels;
    float weight;
public:
    void message()
    {cout<<"vehicle message\n";}
};
class car:public vehicle
{
    int passengers;
public:
    void message(){
        cout<<"car message\n";}
};
class truck:public vehicle
{
    int goods;
public:
    void message(){
        cout<<"truck message\n";}
};
void main()
{
    vehicle obj,*ptr;
    car obj1;
    truck obj2;
    ptr=&obj;
    ptr->message();
    ptr=&obj1;
    ptr->message();
    ptr=&obj2;
    ptr->message();
}

```

三、编程题

1. 编写一个输出学生和教师数据的程序，学生数据有编号、姓名、年龄、班号和成绩；教师数据有编号、姓名、年龄、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。

【参考答案】

一、选择题

1. b
2. a
3. a
4. a
5. c
6. b
7. a
8. c

二、填空题

1. 私有继承
2. 私有
3. 保护
4. 基类、对象成员所在类、派生类
5. 成员初始化列表
6. 二义性
7. base::f()
8. constructing base!
constructing derived!
destructing derived!
destructing base!
9. derived(int x1,int y1,int z1,int xyz1):base(x1),base1(x1,y1),base2(x1,z1)
其中参数名可以为任意合法变量名。
10. vehicle message
vehicle message
vehicle message

三、编程题

1. 参考程序如下：

```
#include <iostream.h>
#include <string.h>
class person
{protected:
    char name[10];
    int age;
public:
    person(char n[],int a)
    {
        strcpy(name,n);
        age=a;
    }
    void disp() {
        cout<<" 姓 名: "<<name<<endl;
        cout<<" 年 龄: "<<age<<endl;
    }
};
class teacher : public person
{
    char tid[10];
    char dept[20];
    char post[10];
public:
    teacher(char *s1,char *s2,int i,char *s3,char *s4):person(s2,i)
```

```

    {
        strcpy(tid,s1);
        strcpy(dept,s3);
        strcpy(post,s4);
    }
    void disptech()
    {
        cout<<" 编    号: "<<tid<<endl;
        person::disp();
        cout<<" 工作部门: "<<dept<<endl;
        cout<<" 职    称: "<<post<<endl<<endl;
    }
};

class student : public person
{
    char sid[12];
    char cnum[12];
    double score;
public:
    student(char *s1,char *s2,int a, char *s3,double i):person(s2,a)
    {
        strcpy(sid,s1);
        strcpy(cnum,s3);
        score=i;
    }
    void dispstu() {
        cout<<" 学    号: "<<sid<<endl;
        person::disp();
        cout<<" 班    级: "<<cnum<<endl;
        cout<<" 成    绩: "<<score<<endl;}
};

int main()
{
    teacher t1("001","张伟",38,"信息学院","教授");
    student s1("2007023","李涛",19,"07001",510);
    t1.disptech();
    s1.dispstu();
    return 0;
}

```


第 10 章 多态性与虚函数

【内容提要】

继承解决了站在前人肩膀上继续前行的可能，但社会在革新，会出现一些新的事物，在某些方面完全颠覆了它“老祖宗”的方式，那这个问题怎么解决呢？自然，我们会想到函数的重载，以重载函数来覆盖前人的实现。但因为子类继承了父类的行为，因为子类叛逆，准备了一套自己的行为（函数重载），这样的 2 个同名的函数在调用的时候要明确指明，特别繁琐。对于子类而言，肯定是希望调用这个重载的函数时，就是自己实现的行为，而不是父类的行为，那好，我们就有必要在父类的函数之前以 `virtual` 标记这个函数若在子类重载了，就以子类的函数为准。只不过，这个重载的函数，比起普通意义的函数重载更为苛刻，要求函数的返回值、名称和参数列表要一模一样。正是虚函数的这种特性，帮助子类实现了“解放”，可以标新立异，按照自己的行为来。而遵照面向对象的规范，父类指针是可以指向子类（甚至“孙、重孙”）对象的，这样，当父类指针指向子类对象时，调用虚函数时，到底是应该调用基类的，还是子类的呢？显而易见，子类对象是希望调用子类的，而 `virtual` 函数帮助子类对象实现了这种特性，这个特性就叫做多态。

- ① **多态性的概念**：见上描述
- ② **函数和运算符的重载**：在重载的函数里，有一类比较特殊的函数，称之为操作符函数，这类函数的重载比较特殊，但非常重要，并且灵活。
- ③ **虚函数和抽象类**。

【重点与难点】

10.1 多态性的概念

在面向对象的概念中，多态性是指不同对象接收到相同消息时，根据对象类的不同产生不同的动作。

由静态联编支持的多态性称为编译时的多态性或静态多态性，也就是说，确定同名操作的具体操作对象的过程是在编译过程中完成的。C++ 用函数重载和运算符重载来实现编译时的多态性。

由动态联编支持的多态性称为运行时的多态性或动态多态性，也就是说，确定同名操作的具体操作对象的过程是在运行过程中完成的。C++ 用继承和虚函数来实现运行时的多态性。

10.2 函数和运算符的重载

10.2.1 函数重载

面向对象程序设计中，函数的重载表现为两种情况：第一种是参数个数或类型有所差别重载，第二种是函数的参数完全相同但属于不同的类。

10.2.2 运算符重载

C++ 预定义的运算符只是对基本数据类型进行操作，而对于自定义的数据类型比如类，却没有类似的操作。为了实现对自定义类型的操作，就必须自己编写程序来说明某个运算符作用在这些数据类型上时，应该完成怎样的操作，这就要引入运算符重载的概念。

✚ 运算符的重载形式有两种，一种是重载为类的成员函数，一种是重载为类的友元函数。将运算符重载为它将要操作的类的成员函数，称为成员运算符函数。实际使用时，总是通过该类的某个对象访问重载的运算符。

成员运算符函数的定义：

在类内声明的一般形式为：

```
<返回类型> operator<运算符>(参数表);
```

在类外定义的一般形式为：

```
<返回类型> <类名::> operator<运算符>(参数表)
{
    函数体
}
```

其中，**operator** 是定义运算符重载函数的关键字；运算符是要重载的运算符的名称；参数表给出重载运算符所需要的参数和类型。

✚ 将重载的运算符函数定义为类的友元函数，称为友元运算符函数。友元运算符函数不是类的成员，它在类内声明原型，在类外定义函数本身。由于它不是类的成员函数，不属于任何一个类对象，所以没有 **this** 指针，因此，重载双目运算符时要有两个参数，重载单目运算符时只要一个参数就可以了。

友元运算符函数的定义：

在类内声明的一般形式为：

```
friend<返回类型> operator<运算符>(参数表);
```

在类外定义的一般形式为：

```
<返回类型> operator<运算符>(参数表)
{
    函数体
}
```

其中，**friend** 是声明友元函数的关键字，**operator** 是定义运算符重载函数的关键字；运算符是要重载的运算符的名称；参数表给出重载运算符所需要的参数和类型。

✚ 几种典型运算符的重载

- ① 加法运算符 “+” 的重载
- ② “++” 和 “--” 的重载
- ③ 赋值运算符 “=” 的重载
- ④ 函数调用运算符 “()” 的重载
- ⑤ 下标运算符 “[]” 的重载

10.3 虚函数和抽象类

虚函数是重载的另一种形式，实现的是动态的重载，即函数调用与函数体之间的联系是在运行时才建立，也就是动态联编。

10.3.1 虚函数的定义和使用

虚函数的定义是在基类中进行的，即把基类中需要定义为虚函数的成员函数声明为 **virtual**。当基类中的某个成员函数被声明为虚函数后，它就可以在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、参数个数和类型、参数的顺序都必须与基类中的原型完全一致。

虚函数定义的一般形式为：

```
virtual<函数类型><函数名>(参数表)
{
    函数体
}
```

使用虚函数时应注意如下问题：

- ① 虚函数的声明只能出现在类声明的函数原型的声明中，不能出现在函数体实现的时候，而且，基类中只有保护成员或公有成员才能被声明为虚函数。
- ② 在派生类中重新定义虚函数时，关键字 **virtual** 可以写也可不写，但在容易引起混乱时，应写上该关键字。
- ③ 动态联编只能通过成员函数来调用或通过指针、引用来访问虚函数，如果用对象名的形式来访问虚函数，将采用静态联编。
- ④ 虚函数必须是所在类的成员函数，不能是友元函数或静态成员函数。但可以在另一个类中被声明为友元函数。

- ⑤ 构造函数不能声明为虚函数，析构造函数可以声明为虚函数。
⑥ 由于内联函数不能在运行中动态确定其外治，所以它不能声明为虚函数。

10.3.2 纯虚函数和抽象类

抽象类是一种特殊的类，它为一族类提供统一的操作界面，建立抽象类就是为了通过它多态地使用其中的成员函数。抽象类是带有纯虚函数的类。

一个抽象类至少带有一个纯虚函数。纯虚函数是在一个基类中说明的虚函数，它在该基类中没有具体的操作内容，要求各派生类在重新定义时根据自己的需要定义实际的操作内容。纯虚函数的一般定义形式为：

`virtual<函数类型><函数名>(参数表)=0;`

纯虚函数与普通虚函数的定义的不同在于书写形式上加了“=0”，说明在基类中不用定义该函数的函数体，它的函数体由派生类定义。

如果一个类中至少有一个纯虚函数，这个类就成为抽象类。它的主要作用是为一个族类提供统一的公共接口，以有效地发挥多态的特性。使用时应注意以下问题：

- ① 抽象类只能用作其它类的基类，不能建立抽象类的对象。因为它的纯虚函数没有定义功能。
- ② 抽象类不能用作参数类型、函数的返回类型或显式转换的类型。
- ③ 可以声明抽象类的指针和引用，通过它们，可以指向并访问派生类对象，从而访问派生类的成员。
- ④ 若抽象类的派生类中没有给出所有纯虚函数的函数体，这个派生类仍是一个抽象类。若抽象类的派生类中给出了所有纯虚函数的函数体，这个派生类不再是一个抽象类，可以声明自己的对象。

【典型例题】

例题 1. 下面关于虚函数和函数重载的叙述不正确的是（ ）。

- (a)虚函数不是类的成员函数
(b)虚函数实现了 C++ 的多态性
(c)函数重载允许非成员函数，而虚函数则不行
(d)函数重载的调用根据参数的个数、序列来确定，而虚函数依据对象确定

解答：

函数重载和虚函数是 C++ 中实现多态性的两种手段，但是它们的实现机制是不一样的；函数重载依据调用时的参数进行区分，而虚函数则根据对象实际的指向确定调用的版本。答案为：a。

例题 2. （ ）是一个在基类中说明的虚函数，它在该基类中没有定义，但要求任何派生类都必须定义自己的版本。

- (a)纯虚函数 (b)虚析构造函数 (c)虚构造函数 (d)静态成员函数

解答：

抽象类中的纯虚函数没有具体的定义，需要在抽象类的派生类中定义。因此，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，但要求任何派生类都必须定义自己的版本。答案为：a

例题 3. 实现运行时的多态性要使用（ ）。

- (a)构造函数 (b)析构造函数 (c)重载函数 (d)虚函数

解答：

动态联编要在程序运行时才能确定调用哪个函数。虚函数是实现动态联编的必要条件之一，没有虚函数一定不能实现动态联编。答案为：d。

例题 4. 关于虚函数的描述中，（ ）是正确的。

- (a)派生类的虚函数与基类的虚函数具有不同的参数个数和类型

(b)基类中说明虚函数后，派生类中其对应的函数一定要说明为虚函数

(c)虚函数是一个成员函数

(d)虚函数是一个 static 类型的成员函数

解答：

为实现某种功能而假设的函数称为虚函数。虚函数是用关键字 `virtual` 进行说明。虚函数是动态联编的基础。虚函数只能是类中的一个成员函数，但不能是静态成员；派生类的虚函数与基类的虚函数具有相同的参数个数和类型，当派生类的虚函数与基类中的对应的虚函数的参数不同时，派生类的虚函数将丢失虚特性，变为重载函数；对于基类中的虚函数，在派生类中自然是虚函数，可不必说明。答案为：c。

例题 5. 下面的程序中，有错误的语句是_____。

```
class A //①
{
public: //②
    A()
    {
        func(); //③
    }
    virtual void func()=0; //④
};
```

解答：

在成员函数内可以调用纯虚函数，但在构造函数或析构函数内调用一个纯虚函数将导致程序运行错误，因为没有为纯虚函数定义代码。答案为：③

例题 6. 运行下列程序的结果为_____。

```
#include<iostream.h>
class base
{
public:
    void display1(){cout<<"base::display1()"<<endl;}
    virtual void display2(){cout<<"base::display2()"<<endl;}
};
class derived:public base
{
public:
    void display1(){cout<<"derived::display1()"<<endl;}
    void display2(){cout<<"derived::display2()"<<endl;}
};
void main()
{
    base * pbase;
    derived d;
    pbase=&d;
    pbase->display1 ();
    pbase->display2();
}
```

解答：

本题主要考查有关多态性的相关知识。在基类 `base` 中，定义了一个函数 `display1()` 和虚函数 `display2()`；在派生类 `derived` 中，重写了函数 `display1()`，而且重新定义的虚函数 `display2()`。由于基类指针 `pbase` 指向的是派生类的一个对象，因而会调用派生类的 `display2()` 版本，但是对于一般的成员函数 `display1()`，仍然遵循一般的调用规则，只调用基类的 `display1()` 版本。

本题答案为：

`base::display1()`

derived::display2()

例题 7. 下面的程序的输出结果为 an animal a person an animal a person, 请将程序补充完整。

```
#include<iostream.h>
class animal{
public:
    _____①_____ void speak(){cout<<"An animal"<<" ";}
};
class person:public animal{
public:
    void speak(){cout<<"a person"<<" ";}
};
void main()
{
    animal a,_____②_____;
    person p;
    a.speak();
    p.speak();
    pa=&a;
    pa->speak();
    _____③_____;
    pa->speak();
}
```

解答:

本题主要考查对多态性的理解与应用。本题通过虚函数实现多态性,所以在基类中应定义虚函数;为了实现多态性,必须定义基类的指针,然后将它指向各个派生类的对象。本题答案为: ①virtual、②*pa、③pa=&p

【习题】

一、选择题

1. 在 C++ 中,用于实现运行时多态性的是 ()。
A) 内联函数
B) 重载函数
C) 模板函数
D) 虚函数
2. 如果一个类至少有一个纯虚函数,那么就称该类为 ()。
(a)抽象类 (b)派生类 (c)虚基类 (d)以上都不对
3. 为了区分一元运算符的前缀和后缀运算,在后缀运算符进行重载时,额外添加一个参数,其类型是 ()。
(a)void (b)char (c)int (d)float
4. 下面关于运算符重载的说法中,错误的是 ()。
(a)可以对 C++ 所有运算符进行重载
(b)运算符重载保持固有的结合性和优先级顺序
(c)运算符重载不能改变操作数的个数
(d)在运算符函数中,不能使用缺省的参数值
5. 下列关于抽象类的说明中不正确的是 ()。
(a)含有纯虚函数的类称为抽象类

- (b)抽象类不能被实例化，但可声明抽象类的指针变量
- (c)抽象类的派生类可以实例化
- (d)纯虚函数可以被继承

6. 运行下列程序的输出结果为（ ）。

```
#include<iostream.h>
class base
{
public:
    void fun1(){cout<<"base"<<endl;}
    virtual void fun2(){cout<<"base"<<endl;}
};
class derived:public base
{
public:
    void fun1(){cout<<"derived"<<endl;}
    void fun2(){cout<<"derived"<<endl;}
};
void f(base &b){b.fun1();b.fun2();}
int main()
{
    derived obj;
    f(obj);
    return 0;
}
```

- (a)base (b)base (c)derived (d)derived
- Base derived base derived

7. 运行下列程序结果为（ ）。

```
#include<iostream.h>
class complex
{
    double re,im;
public:
    complex(double r,double i):re(r),im(i){}
    double real() const{return re;}
    double image() const{return im;}
    complex& operator+=(complex a)
    {
        re+=a.re ;
        im+=a.im ;
        return *this;
    }
};
ostream &operator<<(ostream&s,const complex& z)
{
    return s<< '('<<z.real ()<<','<<z.image()<<')';
}
int main()
{
    complex x(1,-2),y(2,3);
    cout<<(x+=y)<<endl;
    return 0;
}
```

- (a) (1,-2) (b) (2,3) (c) (3,5) (d) (3,1)

二、填空题

1. 多数运算符既能作为类的成员函数重载，也能作为类的非成员函数重载，但[]运算符只能作为类的_____函数重载。

2. 下列程序的输出结果为 2，请将程序补充完整。

```
#include <iostream>
using namespace std;
class Base
{
public:
    _____ void fun() { cout<<1; }
};
class Derived:public Base
{
public:
    void fun() { cout<<2; }
};
int main()
{
    Base *p= new Derived;
    p->fun();
    delete p;
    return 0;
}
```

3. 运行下列程序结果为_____。

```
#include<iostream.h>
class one
{
public:
    virtual void f(){cout<<"1";}
};
class two:public one
{
public:
    two(){cout<<"2";}
};
class three:public two
{
public:
    virtual void f(){two::f();cout<<"3"; }
};
int main()
{
    one aa,*p;
    two bb;
    three cc;
    p=&cc;
    p->f();
    return 0;
}
```

4. 运行下列程序结果为_____。

```
#include<iostream>
using namespace std;
class A{
public:
```

```

        virtual void func1(){cout<<"A1";}
        void func2(){cout<<"A2";}
};
class B: public A{
public:
    void func1(){cout<<"B1";}
    void func2(){cout<<"B2";}
};
int main(){
    A *p=new B;
    p->func1();
    p->func2();
    return 0;
}

```

5. 下面的程序通过重载运算符“+”实现了两个一维数组对应元素的相加。请将程序补充完整。

```

#include<iostream.h>
class Arr
{
    int x[20];
public:
    Arr(){ for(int i=0;i<20;i++) x[i]=0;}
    Arr(int *p)
    { for(int i=0;i<20;i++)x[i]=*p++;}
    Arr operator+(Arr a)
    {
        Arr t;
        for(int i=0;i<20;i++)
            t.x[i]=_____ ① _____;
        return _____ ② _____;
    }
    Arr operator+=(Arr a)
    {
        for(int i=0;i<20;i++)x[i]=_____ ③ _____;
        return _____ ④ _____;
    }
    void show()
    {
        for(int i=0;i<20;i++)cout<<x[i]<<"\t";
        cout<<endl;
    }
};
void main()
{
    int array[20];
    for(int i=0;i<20;i++) array[i]=i;
    Arr a1(array),a2(array),a3;
    a3=a1+a2;a3.show ();
    a1+=a3;a1.show();
}

```

6. 运行下列程序，分别输入“tom”、“m”、“23”、“321456”输出结果为_____。

```

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

```



```

{
public:
    Employee(void) {};
    Employee(char *name, char sex, int age, char *phone)
    {
        strcpy(Employee::name, name);
        Employee::sex = sex;
        Employee::age = age;
        strcpy(Employee::phone, phone);
    };
    friend ostream &operator<<(ostream &cout, Employee emp);
    friend istream &operator>>(istream &stream, Employee &emp);
private:
    char name[256];
    char phone[64];
    int age;
    char sex;
};
ostream &operator<<(ostream &cout, Employee emp)
{
    cout << "Name: " << emp.name << "; Sex: " << emp.sex;
    cout << "; Age: " << emp.age << "; Phone: " << emp.phone << endl;
    return cout;
}
istream &operator>>(istream &stream, Employee &emp)
{
    cout << "Enter Name: ";
    stream >> emp.name;
    cout << "Enter Sex: ";
    stream >> emp.sex;
    cout << "Enter Age: ";
    stream >> emp.age;
    cout << "Enter Phone: ";
    stream >> emp.phone;
    return stream;
}
void main(void)
{
    Employee worker;
    cin >> worker;
    cout << worker ;
}

```

7. 运行下列程序结果为_____。

```

#include <iostream.h>
class sample {
public:
    int i;
    sample *operator->(void) {return this;}
};
void main(void)
{
    sample obj;
    obj->i = 10;
    cout << obj.i << " " << obj->i;
}

```

8. 运行下列程序结果为_____。

```

#include <iostream.h>
#include <stdlib.h>
class Base
{
public:
    virtual int add(int a, int b) { return(a + b); };
    virtual int sub(int a, int b) { return(a - b); };
    virtual int mult(int a, int b) { return(a * b); };
};
class ShowMath : public Base
{
    virtual int mult(int a, int b)
    {
        cout << a * b << endl;
        return(a * b); };
};
class PositiveSubt : public Base
{
    virtual int sub(int a, int b) { return(abs(a - b)); };
};
void main(void)
{
    Base *poly = new ShowMath;
    cout << poly->add(562, 531) << ' ' << poly->sub(1500, 407) << ' ';
    poly->mult(1093, 1);
    poly = new PositiveSubt;
    cout << poly->add(892, 201) << ' ' << poly->sub(0, 1093) << ' ';
    cout << poly->mult(1, 1093);
}

```

三、编程题

1. 下面的 shape 类是一个表示形状的抽象类，area()为求图形面积的函数，total()是一个求不同形状的图形面积总和的函数。请从 shape 类派生三角形类（triangle）和矩形类(rectangle)，并给出具体的求面积函数。

```

class shape
{
public:
    virtual float area()=0;
};
float total(shape *s[],int n)
{
    float sum=0.0;
    for(int i=0;i<n;i++)
        sum+=s[i]->area();
    return sum;
}

```

2.编写程序，定义一个描述三维空间坐标点的类，用成员函数重载“+”运算符以实现两个三维坐标的相加，用友元函数重载前置“++”运算符以实现三维坐标的加一操作。

【参考答案】

一、选择题

1. d
2. a
3. c
4. a
5. c

6. b
7. d

二、填空题

1. 成员
2. virtual
3. 2213
4. A1A2
5. ①x[i]+a.x[i] 、 ②t 、 ③x[i]+a.x[i] 、 ④*this
6. Name: tom; Sex: m; Age: 23; Phone: 321456
7. 10 10
8. 1093 1093 1093
1093 1093 1093

三、编程题

1. 参考程序如下:

```
#include<iostream.h>
#include<math.h>
class shape
{
    public:
        virtual float area()=0;
};
class rectangle:public shape
{
    float width,height;
public:
    rectangle(float w,float h){ width=w;height=h;}
    void show()
    {
        cout<<"width:"<<width<<","<<"height:"<<height<<endl;
        cout<<"area:"<<area()<<endl;
    }
    float area(){return width*height;}
};
class triangle:public shape
{
    float a,b,c;
public:
    triangle(float aa,float bb,float cc)
    {a=aa;b=bb;c=cc;}
    float show()
    {
        cout<<"a:"<<a<<","b:"<<b<<","c:"<<c<<endl;
        cout<<"area:"<<area()<<endl;
    }
    float area()
    {
        float s=(a+b+c)/2;
        return sqrt(s*(s-a)*(s-b)*(s-c));
    }
};
float total(shape *s[],int n)
{
    float sum=0.0;
    for(int i=0;i<n;i++)
        sum+=s[i]->area();
    return sum;
```

```

}
void main()
{
    float w,h,aa,bb,cc;
    cout<<"请输入三角形的三条边: "<<endl;
    cin>>aa>>bb>>cc;
    triangle t(aa,bb,cc);
    cout<<"请输入矩形的长和宽: "<<endl;
    cin>>w>>h;
    rectangle r(w,h);
    shape *p[2];
    p[0]=&t;
    p[1]=&r;
    cout<<"两个图形面积的和为: "<<total(p,2)<<endl;
}

```

2. 参考程序如下:

```

#include<iostream.h>
class threedime
{
    float x,y,z;
public:
    threedime(float a=0,float b=0,float c=0){x=a;y=b;z=c;}
    void show(){cout<<"x:"<<x<<"y:"<<y<<"z:"<<z<<endl;}
    threedime operator+(threedime c);
    friend threedime operator++(threedime& c);
};

threedime threedime::operator+(threedime c)
{
    threedime t=*this;
    t.x+=c.x;
    t.y+=c.y;
    t.z+=c.z;
    return t;
}

threedime operator++(threedime& c)
{
    c.x++;
    c.y++;
    c.z++;
    return c;
}

void main()
{
    threedime td1(1,2,3),td2(1,1,1),td3,td4;
    td1.show();td2.show();
    td3=++td1;td4=td1+td2;
    td3.show();td4.show();
}

```

第 11 章 模板

【内容提要】

模板是实现“泛型编程”的重要技术之一，可以说没有模板，C++就没法支持泛型编程。何为模板呢？事实上，在面向过程的程序设计里，我们是通过“函数复用”来提高软件开发效率的。我们知道，函数严格地约定了每个形参的数据类型，尽管我们可以用 `void*` 这种无比强大的指针，但这种做法太不自然，晦涩难懂。那我们能否写一个函数，与数据类型无关呢？这就是泛型编程的初衷。幸运地是，一批计算机天才们使用模板技术实现了这个想法。模板技术的想法就是在函数定义和实现的阶段，我们用抽象的 `class T`（或者 `typename T`）来代指形参类型，并且实现相应的功能代码，待到这个模板函数被调用的时，有了具体的参数类型 `T` 之后，我们再把先前的代码在此处展开成一个具体参数类型的函数，并编译、链接到最后的程序中来。这可以视为一种“动态绑定”的技术。简而言之，模板就是将参数类型进行抽象化的一种技术。本章主要包括：

- ① 模板的概念：如上所述。
- ② 函数模板的定义和使用：语法
- ③ 类模板的定义和使用：语法

【重点与难点】

11.1 模板的概念

模板是实现代码复用的一种工具，它可以实现类型参数化，把类型定义为参数，实现代码的真正复用。

模板分两类：函数模板和类模板。用一个代码段指定一组函数称为函数模板，或用一个代码段指定一组相关类称为类模板。

11.2 函数模板的定义和使用

11.2.1 函数模板的定义

格式为：

```
template <typename(或 class) 数据类型参数标识符>
<返回类型><函数名>(参数表)
{
    函数体
}
```

说明：

- ① `template` 是定义模板函数的关键字；`template` 后面的尖括号不能省略；
- ② `typename` 是声明数据类型参数标识符的关键字，也可用 `class`。它用以说明其后面的标识符是数据类型标识符。在这个函数定义中，凡希望根据实参数据类型来确定数据类型的变量，都可以用数据类型参数标识符来说明，从而使这个变量可以适应不同的数据类型。
- ③ 定义函数模板时，可以声明多个类型参数标识符，各标识符之间用逗号分开。
- ④ 函数模板只是声明了一个函数的描述即模板，不是一个可以直接执行的函数，只有根据实际情况用实参的数据类型代替类型参数标识符之后，才能产生真正的函数。

11.2.2 模板函数

在使用函数模板时，要将形参“数据类型参数标识符”实例化为确定的数据类型。将类型形参实例化的参数称为模板实参，用模板实参实例化的函数称为模板函数。

11.2.3 重载模板函数

函数模板可使用多种方式重载。可以使用其它函数模板，指定不同参数的相同函数名。也可以用非模板函数重载。

✚ 用非模板函数重载函数模板有两种方法：

- ① 借用函数模板的函数体，只声明非模板函数的原型，它的函数体借用函数模板的函数体。
- ② 重新定义函数体。即重新定义一个完整的非模板函数，它所带的参数可以随意。

✚ 在 C++ 中，函数模板与同名的非模板函数重载时，应遵循下列调用原则：

首先寻找一个参数完全匹配的函数，若找到就调用它。若找不到，则寻找一个函数模板，将其实例化生成一个匹配的模板函数，若找到就调用它。若找不到，则从第一步中通过类型转换产生参数匹配，若找到就调用它。否则调用失败。

11.3 类模板的定义和使用

11.3.1 类模板的定义

格式为：

```
template<class 数据类型参数标识符>
class 类名
{
    //.....
};
```

说明：

- ① `template` 是声明类模板的关键字，`template` 后面的尖括号不能省略。
- ② 定义类模板时，可以声明多个类型参数标识符，各标识符之间用逗号分开。
- ③ 类模板定义中，凡要采用标准数据类型的数据成员、成员函数的参数或返回类型的前面都要加上类型标识符。
- ④ 如果类中的成员函数要在类的声明之外定义，则它必须是模板函数。其定义形式为：
`template<class 数据类型参数标识符>`
`函数返回类型 类名<数据类型参数标识符>::函数名(数据类型参数标识符 形参 1, ……，数据类型参数标识符 形参 n)`
{
 函数体
}
- ⑤ 类模板使类中的一些数据成员和成员函数的参数或返回值可以取任意的数据类型。类模板不是一个具体的类，它代表着一族类，是这一族类的统一模式。使用类模板就是要将它实例化为具体的类。

11.3.2 模板类

将类模板的模板参数实例化后生成的具体的类，就是模板类。由类模板生成模板类的一般形式为：

类名<模板实参表> 对象名 1，对象名 2，…，对象名 n；

【典型例题】

例题 1. 有如下函数模板定义：

```
template <class T>
T func(T x, T y) { return x*x*x+y*y*y; }
```

在下列对 `func` 的调用中，错误的是（ ）。

- (a) `func(3, 5);` (b) `func(3.0, 5.5);`
(c) `func(3, 5.5);` (d) `func<int>(3, 5.5);`

解答：

本题主要考查函数模板的使用方法。这里选项 a 是将函数模板中的类型形参实例化为 `int` 型，选项 b 是将函数模板中的类型形参实例化为 `double` 型。选项 c 中函数 `func` 的两个实参一个为 `int` 型，一个为 `double` 型，无法使用该函数模板。选项 d 中模板实参被显式指定，显示指定模板实参的方法是用尖括号 `<>` 将用逗号隔开的实参类型列表括起来紧跟在函数模板实例

的名字后面。如选项 d 将实参类型指定为 int 型。本题答案为：c。

例题 2. 有如下程序：

```
template <class T>
class Array
{
    protected:
        int num
        T *p;
    public:
        Array(int);
        ~Array();
};
Array::Array(int x)// ①
{
    num=x;// ②
    p=new T[num];} // ③
Array::~~Array()//④
{
    delete []p; // ⑤
}
void main()
{
    Array a(10); // ⑥
}
```

其中有错误的语句为_____，应改正为_____。

解答：

本题主要考查类模板的定义和使用。如果类中的成员函数要在类的声明之外定义，则它必须是模板函数。其定义形式为：

template<class 数据类型参数标识符>

函数返回类型 类名<数据类型参数标识符>::函数名(数据类型参数标识符 形参 1, ……，数据类型参数标识符 形参 n)

```
{
    函数体
}
```

本程序中类的构造函数和析构函数均在类中声明，类外定义。所以①、④语句关于这些函数的定义均错误，应遵循上边所述形式。由类模板生成模板类的一般形式为：

类名<模板实参表> 对象名 1, 对象名 2, …, 对象名 n;

⑥语句对类模板的使用错误。所以本题答案为：①、④、⑥

改正程序：

```
①template<class T>
    Array<T>::Array(int x)
④template<class T>
    Array<T>::~~Array()
⑥Array<int> a(10);
```

例题 3. 运行下列程序结果为_____。

```
#include <iostream.h>
#include<string.h>
template<class T,class U>
T add(T a, U b)
{
    return(a + b);
}
```

```

char* add(char *a,char *b )
{
    return strcat(a,b);
}
void main()
{
    int x=1,y=2;
    double x1=1.1,y1=2.2;
    char p[10]="C++";
    cout<<add(x,y)<<" ";
    cout<<add(x1,y1)<<" ";
    cout<<add(x1,y)<<" ";
    cout<<add(p," program")<<endl;
}

```

解答:

本题主要考查对函数模板的定义与使用以及重载模板函数的理解。在 C++ 中,函数模板与同名的非模板函数重载时,应遵循下列调用原则:

首先寻找一个参数完全匹配的函数,若找到就调用它。若找不到,则寻找一个函数模板,将其实例化生成一个匹配的模板函数,若找到就调用它。若找不到,则从第一步中通过类型转换产生参数匹配,若找到就调用它。否则调用失败。本题中主函数对函数 add 的四次调用中前三次都是未找到参数完全匹配的函数,于是找到一个函数模板,将该函数模板实例化为模板函数。第四次调用时找到了参数完全匹配的函数,于是调用了该函数。所以本题答案为: 3, 3.3, 3.1, C++ program。

例题 4. 下列关于类模板的模板参数说法正确的是 ()。

- (a) 只可作为数据成员的类型
- (b) 只可作为成员函数的参数类型
- (c) 只可作为成员函数的返回值类型
- (d) 以上三者都可以

解答:

类模板中的模板参数既可以作为类成员的类型,也可作为成员函数的参数类型还可作为成员函数的返回值类型。本题答案为: d。

例题 5. 有如下类模板的定义,请指出其中的错误_____。

```

template <class T>
class A {public: T num;}

```

解答:

类模板的定义还是在定义类,所以类定义结束的分号不能少。答案为: 定义语句的最后缺少一个分号。

例题 6. 下列说法正确的是 ()。

- (a) 如果从一个带单个 static 数据成员类模板产生几个模板类,则每个模板类共享类模板 static 数据成员的一个副本。
- (b) 模板函数可以用同名的另一个模板函数重载。
- (c) 同一个形参名只能用于一个模板函数。
- (d) 关键字 class 指定函数模板类型参数,实际上表示“任何用户自定义类型”。

解答:

在非模板类中,类的所有对象共享一个 static 数据成员。从类模板实例化的每个模板类有自己的类模板 static 数据成员,该模板类的所有对象共享一个 static 数据成员。每个模板类有自己的类模板的 static 数据成员副本。每个模板类有自己的静态数据成员副本。所以选项 a 错误。同一形式参数名可以用于多个模板函数。选项 c 错误。关键字 class 指定函数模板的类型参数,实际上表示“任何内部类型或用户自定义类型”。选项 d 错误。答案为: b

例题 7. 下面是 3 个数字求和的类模板程序。请将程序补充完整。

```

#include<iostream.h>

```



```

template <class T, _____>
class sum{
T array[size];
public:
    sum(T a,T b,T c){array[0]=a;array[1]=b;array[2]=c;}
    T s(){return _____;}
};
void main()
{
    _____ s1(1,2,3); //定义类对象
    cout<<s1.s()<<endl;

}

```

解答:

本题主要考查对类模板的应用。答案为: int size, array[0]+array[1]+array[2], sum <int,3>

例题 8. 定义一个求幂函数的函数模板。

解答:

参考程序:

```

#include <iostream.h>
template <class T>
T Power(T a, int exp)
{
    T ans = a;
    while(--exp>0) ans*=a;
    return ans;
}
int main()
{
    cout << "2^3= " <<Power(2, 3) << endl;
    cout << "1.1^2= " << Power(1.1, 2) << endl;
    return 0;
}

```

【习题】

一、选择题

1. 模板对类型的参数化提供了很好的支持, 因此 ()。

- (a) 类模板的主要作用是生成抽象类
- (b) 类模板实例化时, 编辑器将根据给出的模板实参生成一个类
- (c) 在类模板中的数据成员都具有同样类型
- (d) 模板中的成员函数都没有返回值

2. 有如下模板:

```

template<typename T,typename U>
T fun(U u){return u;}

```

下列对模板函数 fun 的调用中错误的是 ()。

- (a) a<int,int>(97);
- (b) a<char,int>(97);
- (c) a<double,int>(97);
- (d) a(97);

3. 关于关键字 class 和 typename, 下列表述中正确的是 ()。

- (a) 程序中的 typename 都可以替换为 class
- (b) 程序中的 class 都可以替换为 typename

- (c)在模板形参表中只能用 `typename` 来声明参数的类型
 (d)在模板形参表中只能用 `class` 或 `typename` 来声明参数的类型

4. 运行下列程序的结果为 ()。

```
#include <iostream.h>
template <class T>
void swap(T &a, T &b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}
void swap(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    cout<<"swap two int!"<<endl;
}
int main(void)
{
    int i=1, j=2;
    double x=1.1, y=2.2;
    char a='x', b='z';
    swap(i,j);
    swap(a,b);
    swap(x,y);
    cout<<i<<" "<<j<<endl;
    cout<<a<<" "<<b<<endl;
    cout<<x<<" "<<y<<endl;
}
```

- | | | | |
|----------|--------------|--------------|--------------|
| (a) | (b) | (c) | (d) |
| 2, 1 | swap two int | swap tow int | swap tow int |
| z, x | 2, 1 | swap tow int | swap tow int |
| 2.2, 1.1 | z, x | 2, 1 | swap tow int |
| | 2.2, 1.1 | z, x | 2, 1 |
| | | 2.2, 1.1 | z, x |
| | | | 2.2, 1.1 |

5. 设有函数模板

```
template <class T>
T max(T a,T b)
{
    return (a>b)?a:b;
}
```

则下列语句中对该函数模板的错误使用是 ()。

- (a)`max(1,2)` (b)`max(2.3,4.5)` (c)`max(1,2.3)` (d)`max('a','b')`

6. 关于在调用模板函数时模板实参的使用, 下列描述正确的是 ()。

- (a)对于虚拟类型参数所对应的模板实参, 如果能从模板函数的实参中获得相同的信息, 则都可以省略。
 (b)对于虚拟类型参数所对应的模板实参, 如果它们是参数表中的最后的若干个参数, 则都可以省略。

- (c)对于虚拟类型参数所对应的模板实参，若能够省略则必须省略。
(d)对于常规参数所对应的模板实参，任何情况下都不能省略。

二、填空题

1. 模板是实现类属机制的一种工具，其功能非常强大，它既允许用户构造类属函数，即_____；也允许用户构造类属类，即_____。

2. 有如下函数模板定义，错误的语句是_____，应该正为_____。

```
template <class T>;//①
T fun(T x)//②
{
    return x;//③
}
```

3. 运行下列程序的输出结果为_____。

```
#include<iostream.h>
template <class T >
class Tstack
{
    enum {size=1000};
    T stack[size];
    int top;
public:
    Tstack():top(0){ }
    bool push(const T &i){
        if(top<size)
        {
            stack[top++]=i;
            return true;
        }
        else
            return false;
    }
    bool pop(T &value){
        if(top!=0)
        {
            value=stack[--top];
            return true;
        }
        else
            return false;
    }
};
void main()
{
    Tstack<int> a;
    int x;
    a.push(1);
    a.push(2);
    a.pop(x);
    cout<<x;
    a.pop(x);
    cout<<x<<endl;
}
```

4. 下面的程序输出结果是_____。

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

```

template<class T>
T sum(T *x,int size=0)
{
    T a=0;
    for(int i=0;i<size;i++)
        a+=x[i];
    return a;
}
void main()
{
    int array[]={ 1,2,3,4};
    int arraysum=sum(array,4);
    cout<<arraysum<<endl;
}

```

5. 下面是一个函数模板，用于实现从 3 个整数或浮点数中找出最大值。请将函数模板的定义补充完整。

```

template<class T>
T max(_____①_____,T c)
{
    T x;
    x=(a>b)?(a):(b);
    _____②_____;
}

```

6. 有如下程序段，错误的语句是_____，应该正为_____。

```

template<class T> //①
class A
{
    T x,y;//②
public:
    A(T i,T j):x(i),y(j){} //③
};
A(double) obj(1.1,2.2);// ④

```

7. 下面是 3 个数字求和的类模板程序。请将该定义补充完整。

```

template <class T,_____①_______>
class sum
{
    T temp,m[size];
public:
    sum(T a,T b,T c){m[0]=a;m[1]=b;m[2]=c;}
    T s(){return _____②_____;} //计算三个数的和
};

```

三、编程题

编写一个处理冒泡排序的函数模板，并设计主程序完成输入、排序和输出 int 型和 double 型数组的功能。

【参考答案】

一、选择题

1. b
2. d
3. d
4. b
5. c
6. d

二、填空题

1. 函数模板、类模板
2. ①、`template <class T>` 即去掉分号
3. 21
4. 10
5. ①`T a, T b` 、 ②`return (x>c)?(x):(c)`
6. ④、`A<double> obj(1.1,2.2);`
7. ①`int size=3` 、 ② `m[0]+m[1]+m[2]`

三、编程题

参考程序如下：

```
#include <iostream.h>
template <class X> void bubble_sort(X *items, int size);
template <class X> void show_items(X *items, int size);
void main(void)
{
    int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
    double darray[5] = {4.2, 2.5, -0.9, 100.2, 3.0};
    cout << "Here is unsorted integer array: " << endl;
    show_items(iarray, 7);
    cout << "Here is unsorted double array: " << endl;
    show_items(darray, 5);
    bubble_sort(iarray, 7);
    bubble_sort(darray, 5);
    cout << "Here is sorted integer array: " << endl;
    show_items(iarray, 7);
    cout << "Here is sorted double array: " << endl;
    show_items(darray, 5);
}
template <class X> void bubble_sort(X *items, int size)
{
    register int i, j;
    X temp;
    for (i = 1; i < size; i++)
        for (j = size-1; j >= i; j--)
            if (items[j-1] > items[j])
            {
                temp = items[j-1];
                items[j-1] = items[j];
                items[j] = temp;
            }
}
template <class X> void show_items(X *items, int size)
{
    int i;
    for(i=0; i < size; i++)
        cout << items[i] << ", ";
    cout << endl;
}
```

第 12 章 IO & Stream

【内容提要】

stream 的中文解释是“流”，那我们就从中文“流”的角度解释下何为“流”。我们生活中最熟悉的要属“水流”，你打开水龙头，里面就开始有自来水流出，可以使用。事实上，C++里提供的 stream 与之类似。那我们来对“水流”稍加分析，首先，自来水管流出的水流来自于哪里，我们就假定是自来水厂，那说明“水流”是有输入一端的；其次，自来水厂一供水，你家水龙头就有水，这似乎不可能，出发你家就住在自来水厂里，这说明水流到你家里，是有一个缓冲的过程的；最后，我们从水龙头接水的容器也是可以选择的，我既可以用盆接水，也可以用烧水壶接水，也可以用碗接水。上述的 3 个方面，正是“流”的基本特性。那我们以此联想到 C++ 的 IOStream，首先语句 `cout << a << b << c;` 是往 iostream 中注入内容；其次它有个缓冲池来装入这个内容，待这个池中的内容比较多了之后，再一次性的通过管道输出到用户终端，你若要强制输出缓冲池的内容，可以用 `cout<<flush;` 事实上 `endl` 也是强制输出符。最后，你得告诉计算机，内容输出到哪里，输出到标准输出设备，就用 `cout`，输出到文件，就用 `fstream`，要输出到字符串，就用 `stringstream`，记住：iostream 的 `cout`，`ifstream`、`stringstream` 仅仅是盛装内容的器具而已。真是基于此，C++ 提供了 iostream，fstream，sstream 这三类容器，为了区分输入和输出，进一步细分了 `istream`，`ostream`，`ifstream`，`ofstream`，`istringstream`，`ostringstream` 共 6 类容器。主要内容如下：

- ① 流的概念及流类库：见上。
- ② 输入输出的格式控制：掌握相关的设置函数。
- ③ 输入与输出运算符的重载：前面章节已有描述。
- ④ 文件操作：注意区分二进制和 ASCII 的区别，本质上，二者一致。

【重点与难点】

12.1 流的概念及流类库

12.1.1 流的概念

C++ 中的流是指数据从一个对象传递到另一个对象的操作。从流中读取数据称为提取操作，向流内添加数据称为插入操作。流在使用前要建立，使用后要删除。如果数据的传递是在设备之间进行，这种流就称为 I/O 流。C++ 专门内置了一些供用户使用的类，在这些类中封装了可以实现输入输出操作的函数，这些类统称为 I/O 流类。

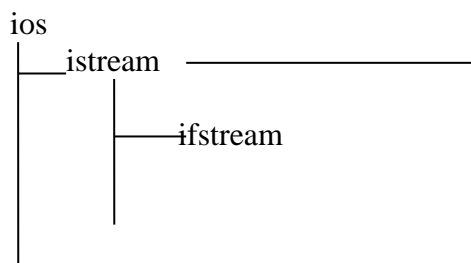
C++ 流预定义了 4 个标准流类对象，它们可以直接用来输入输出。这些标准流类对象都包含在头文件 `iostream.h` 中。

```
istream cin;           //与标准输入设备相关联的标准输入流。
ostream cout;          //与标准输出设备相关联的标准输出流。
ostream cerr;          //与标准错误输出设备相关联的非缓冲方式的标准输出流。
ostream clog;          //与标准错误输出设备相关联的缓冲方式的标准输出流。
```

12.1.2 流类库

C++ 的流类库有两个平行的基类 `streambuf` 和 `ios`，其它的流类都是从这两个基类直接或间接派生的。使用这些流类库时，必须包含相应的头文件。

`ios` 类及其派生类为用户提供了使用流类的接口。以下给出 `ios` 类及其派生类的层次结构图。



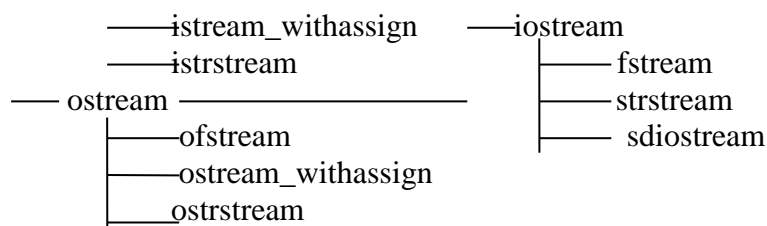


图 12.1 ios 类的层次关系

12.2 输入输出的格式控制

C++ 仍可使用 C 中的 `printf()` 和 `scanf()` 进行格式化控制，同时又提供了两种格式化控制的方法，一是使用 ios 类中的有关格式控制的成员函数，二是使用执行格式化输入/输出的格式控制符。

12.2.1 用 ios 类的成员函数进行格式控制

使用这种方式进行格式控制主要是通过设置及清除格式状态标志、设置域宽、设置填充字符和设置输出精度的操作来完成的。

✚ 设置及清除格式状态标志

① 格式状态标志

格式状态标志	说明
<code>ios::skipws</code>	跳过输入中的空白，用于输入
<code>ios::left</code>	左对齐输出，用于输出
<code>ios::right</code>	右对齐输出，用于输出
<code>ios::internal</code>	在符号和数值之间填充字符，用于输出
<code>ios::dec</code>	转换基数为十进制，用于输入或输出
<code>ios::oct</code>	转换基数为八进制，用于输入或输出
<code>ios::hex</code>	转换基数为十六进制，用于输入或输出
<code>ios::showbase</code>	输出时显示基指示符 (0 表示八进制，0x 或 0X 表示十六进制)，用于输入或输出
<code>ios::showpoint</code>	输出时显示小数点，用于输出
<code>ios::uppercase</code>	输出时表示十六进制的 x 为大写，表示浮点数科学计数法的 e 为大写，用于输出
<code>ios::showpos</code>	正整数前显示 “+” 符号，用于输出
<code>ios::scientific</code>	用科学表示法显示浮点数，用于输出
<code>ios::fixed</code>	用定点形式显示浮点数，用于输出
<code>ios::unitbuf</code>	在输出操作后立即刷新所有流，用于输出
<code>ios::stdio</code>	在输出操作后刷新 stdout 和 stderr，用于输出

图 12.2 格式状态标志

图 12.2 中的格式状态标志在类 ios 中被定义为枚举值，这些枚举元素的值的共同特点是，使状态标志字二进制表示中的不同位为 1，它们共同组成状态标志字，存放在数据成员 `long x_flags` 中。在设置格式状态时可以通过按位或 (|) 运算将多个标志选项组合在一个 `long int` 型的值中。

② 设置状态标志

使用成员函数 `setf` 设置参数所指定的格式标志，并返回 `long int` 型的标志设置值。

一般调用形式为：

流对象.`setf`(格式状态标志);

③ 清除状态标志

使用成员函数 `unsetf` 清除指定的标志并返回清除前的标志值。

一般调用形式为:

流对象.unsetf(格式状态标志);

④ 取状态标志

用成员函数 `flag`, 其不带参数时返回格式状态标志的当前设置 (`long int` 型值); 带参数时按参数指定的格式设置标志, 返回以前的标志设置值。

一般调用形式为:

流对象.flags();

流对象.flags(格式状态标志);

注意: `flags()`与 `setf()`的区别是: `setf()`是在原有的基础上追加设置, 不改变原有设置; `flags()`使用新的设置覆盖原有的设置, 改变了原有设置。

✚ 设置域宽

域宽是指输出字符的长度。用成员函数 `width()`完成域宽的设置。

一般调用形式为:

流对象.width();

流对象.width(int w);

说明:

- ① 不带参数时返回当前的域宽值;
- ② 带参数时将域宽值设置为 `w`, 并返回设置前的域宽值。

✚ 设置填充字符

当输出值长度小于域宽时将剩余部分用设定的填充字符填满, 缺省的填充字符为空格。

用成员函数 `fill` 设置填充字符。

一般调用形式为:

流对象.fill();

流对象.fill(char ch);

说明:

- ① 不带参数时返回当前的填充字符;
- ② 带参数时将填充字符设置为 `ch`, 并返回设置前的填充字符。
- ③ 使用填充字符函数时, 必须与设置域宽函数配合使用, 否则没有意义。

✚ 设置输出精度

用函数 `precision()`设置浮点数输出时的显示精度。

一般调用形式为:

流对象.precision(int p);

12.2.2 用格式控制符进行格式控制

✚ 预定义的格式控制符 (在第二章中已介绍)。

✚ 自定义的格式控制符

为输出流自定义格式控制符的一般形式为:

```
ostream &格式控制符名(ostream &stream)
{
    //自定义代码
    return stream;
}
```

为输入流自定义格式控制符的一般形式为:

```
istream &格式控制符名(istream &stream)
{
    //自定义代码
    return stream;
}
```

12.3 输入与输出运算符的重载

✚ 定义运算符 “<<” 重载函数的一般形式为:

```
ostream &operator<<(ostream &stream, 类名 对象名)
{
```



```

        //操作代码
        return stream;
    }
    定义运算符 ">>" 重载函数的一般形式为:
    istream &operator>>(istream &stream,类名 &对象名)
    {
        //操作代码
        return stream;
    }

```

说明：重载输入/输出运算符函数只能为非成员函数，一般被定义为类的友元。

12.4 文件操作

12.4.1 创建文件流

在 C++ 中进行文件操作，要包含 <fstream.h>。该头文件包括了 ifstream（从文件输入）、ofstream（向文件输出）、fstream（从文件输入输出）流类的定义。它们分别派生自 istream、ostream、iostream。建立文件流就是定义文件流类的对象。如

```

ifstream fin;
ofstream fout;
fstream fio;

```

建立了流以后，就可以把某一个流与文件建立联系，进行文件的读写操作了。

12.4.2 文件的打开与关闭

可以在定义文件流对象的同时完成打开文件的工作，也可在定义文件流对象以后通过 open() 函数完成。

open() 函数是上述三个流类的成员函数，定义在 fstream.h 头文件中。

函数原型为：

```
void open(const unsigned char *,int mode,int dces=filebuf::openprot);
```

说明：

- ① 第一个参数用来传递文件名；
- ② 第二个参数的值决定文件打开的方式，必须从下列值中选取：

ios::app	//使输出追加到文件尾部，只用于输出
ios::ate	//查找文件尾
ios::in	//打开一个文件进行读操作，只用于输入
ios::nocreate	//文件不存在，导致 open() 失败
ios::noreplace	//若文件存在，则 open() 失败
ios::out	//打开一个文件进行写操作，只用于输出
ios::trunc	//删除同名文件
ios::binary	//以二进制方式打开文件，缺省为文本方式

以上各值可以组合使用，之间用 “|” 分开。

- ③ 第三个参数的值决定文件的访问方式及文件的类别。

一般情况下，ifstream 和 ofstream 流类的析构函数就可以自动关闭已打开的文件，也可用函数 close() 关闭文件，它是流类中的成员函数，没有参数，没有发返回值。

12.4.3 文件的读写

文件的顺序读写

- ① get() 函数

函数原型：istream &get(unsigned char &ch);

作用：它从流中每次读出一个字节或一个字符放入引用 ch& 中。
- ② put() 函数

函数原型：istream &put(char ch);

作用：它将一个字节或一个字符写入流中。
- ③ read() 函数

函数原型：istream &read(unsigned char *buf,int num);

作用：从相应的流读出 `num` 个字节或字符的数据，把他们放入指针所指向的缓冲区中。`buf` 是一个指向读入数据存放空间的指针，`num` 说明要读入数据的字节或字符数。

④ `write()`函数

函数原型：`ostream &write(const unsigned char *buf,int num);`

作用：从 `buf` 所指向的缓冲区把 `num` 个字节的数据写到相应的流中。参数的含义、调用及注意事项与 `read()`相同。



文件的随机读写

完成定位操作的函数如下：

① 函数 `seekg()`

函数原型：`istream &seekg(streamoff offset,seek_dir origin);`

作用：用于输入文件，将文件的读指针从 `origin` 说明的位置移动 `offset` 个字节；

② 函数 `seekp()`

函数原型：`ostream &seekp(streamoff offset,seek_dir origin);`

作用：用于输出文件，将文件的写指针从 `origin` 说明的位置移动 `offset` 个字节。

说明：`origin` 的取值由三种情况：

`ios::beg` 从文件头开始，把文件指针移动由 `offset` 指定的距离。

`ios::cur` 从文件当前位置开始，把文件指针移动 `offset` 指定的距离。

`ios::end` 从文件尾开始，把文件指针移动由 `offset` 指定的距离。

③ 函数 `tellg()`

函数原型：`streampos tellg();`

作用：用于输入文件，确定文件当前指针的位置

④ 函数 `tellp()`

函数原型：`streampos tellp();`

作用：用于输出文件，确定文件当前指针的位置。

【典型例题】

例题 1. 在 C++中，打开一个文件就是将一个文件与一个_____建立关联；关闭一个文件就是取消这种关联。

解答：

C++的 I/O 系统是通过一种称为流的机制来实现文件和控制台的 I/O 操作。简单地说，在 C++ 中，输入输出操作是通过流来完成的。而文件流以磁盘文件以及其他可按文件方式进行管理的外部设备为输入输出对象。每个文件流都应当与一个打开的文件相联系；关闭一个文件就是取消这种联系。答案为：流。

例题 2. 进行外交操作时需要包含（ ）头文件。

(a) `iostream` (b) `fstream` (c) `stdio` (d) `stdlib`

解答：

文件流是文件流类的实例对象。C++系统在头文件 `fstream` 中定义了 3 个文件流类：`ifstream`，`ofstream` 和 `fstream`。要创建文件流，必须包含头文件 `fstream`。并声明所创建的文件流是上述哪个类的实例对象。因此，有关文件 I/O 类的说明包含在 `fstream` 头文件中。答案为：b。

例题 3. 当使用 `ifstream` 流类定义一个流对象并打开一个磁盘文件时，文件的隐含打开方式为（ ）。

(a) `ios::in` (b) `ios::out` (c) `ios::trunc` (d) `ios::binary`

解答：

当使用 `ifstream` 流类定义一个流对象并打开一个磁盘文件时，文件的隐含打开方式为 `ios::in`；

当使用 `ofstream` 流类定义一个流对象并打开一个磁盘文件时，文件的隐含打开方式为 `ios::out`。答案为：a。

例题 4. 下列函数中，() 是对文件进行写操作的。

(a) `read()` (b) `seekg()` (c) `get()` (d) `put()`

解答：

`read()` 的功能是输入指定数量的字节存入指定的字符空间中；`seekg()` 的功能是按指定方式将输入定位于指定的相对位置处；`get()` 的功能是提取并返回当前输入位置的字符代码，或将提取的字符存入指定变量中；`put()` 的功能是把一个字符写到输出流中。答案为：d。

例题 5. `cin` 是_____的一个对象，处理标准输入。`cout`, `cerr` 和 `clog` 是_____的对象，`cout` 处理标准输出，`cerr` 和 `clog` 都处理标准出错信息，只是_____输出不带缓冲，_____输出带缓冲。

解答：

C++ 流预定义了 4 个流，它们是 `cin`, `cout`, `cerr` 和 `clog`。它们可以直接用来输入输出，与这 4 个流所联结起的具体设备是：(1) `cin`：与标准输入设备相关联。(2) `cout`：与标准输出设备相关联。(3) `cerr`：与标准错误输出设备相关联（非缓冲）。(4) `clog`：与标准错误输出设备相关联（缓冲）。因此，可知 `cin` 是 `istream` 的一个对象，处理标准输入。`cout`, `cerr` 和 `clog` 是 `ostream` 的对象，`cout` 处理标准输出，`cerr` 和 `clog` 都处理标准出错信息，只是 `cerr` 输出不带缓冲，`clog` 输出带缓冲。答案为：`istream`、`ostream`、`cerr`、`clog`。

例题 6. 下面是一个将文本文件 `readme` 中的内容读出并显示在屏幕上的示例，请完成该程序。

```
#include<fstream.h>
void main()
{
    char buf[80];
    ifstream me("c:\\readme");
    while(____①____)
    {
        me.getline(____②____,80);
        cout<<buf<<endl;
    }
    me.close();
}
```

解答：

`while` 循环中需要加循环终止条件，在①处应该判断文件是否结束，应填入 “`!me.eof()`”；根据 `getline()` 函数的原型可知，此处应填入一个字符串变量，故填入 “`buf`”。即答案为：① `!me.eof()`、② `buf`。

例题 7. 下面的程序向 C 盘的 `new` 文件写入内容，然后把该内容显示出来，试完成该程序。

```
#include<fstream.h>
void main()
{
    char str[100];
    fstream f;
    _____①_____;
    f<<"hello world";
    f.put('\n');
    f.seekg(0);
    while( _____②_____)
    {
        f.getline(str,100);cout<<str;
    }
}
```

_____③_____;

}

解答:

向文件写内容前首先需要打开该文件并指明需要的输入输出操作;在使用完毕后应当关闭该文件。在读到文件结尾时应当结束读取操作。故本题答案为:

①f.open("c:\\new",ios::in|ios::out)、②!f.eof()、③f.close()。

例题 8. 运行下列程序结果为 ()。

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    int a=12;
    double b=314159.26;
    cout<<a<<endl;
    cout.setf(ios::oct|ios::showbase);
    cout<<a<<endl;
    cout.precision(8);
    cout<<-b<<endl;
    cout.setf(ios::scientific|ios::uppercase|ios::left);
    cout.width(20);
    cout.fill('*');
    cout<<-b<<endl;
    cout.width(20);
    cout.setf(ios::internal);
    cout<<-b<<endl;
    cout.precision(6);
    cout.fill(' ');
    cout.unsetf(ios::scientific|ios::uppercase|ios::left|ios::internal);
}
```

解答:

本题主要考查用 ios 类的成员函数进行格式控制的使用方法。参见本章重点与难点部分的介绍。本题答案为:

12
014
-314159.26
-314159.260E+005****
-****3.14159260E+005

【习题】

一、选择题

1. 运行下列程序结果为 ()。

```
#include <iostream.h>
int main( )
{
    cout.width(6);
    cout.fill('*');
    cout << 'a'<<1 << endl;
    return 0;
}
```

(a)*****a*****1

(b)*****a1

(c)a*****1*****

(d)a*****1

2. 下面关于 C++流的叙述中，正确的是（ ）。

- (a) cin 是一个输入流对象。
- (b) 可以用 ifstream 定义一个输出流对象。
- (c) 执行语句序列 `char *y="PQMN"; cout<<y;` 将输出字符串 "PQMN"的地址。
- (d) 执行语句序列 `char x[80]; cin.getline(x,80);` 时，若键入 Happy new year，则 x 中的字符串是 "Happy"。

3. 若磁盘上已存在某个文本文件，它的全路径文件名为 `d:\kaoshi\test.txt`，则下列语句中不能打开这个文件的是（ ）。

- (a) `ifstream file("d:\kaoshi\test.txt");`
- (b) `ifstream file("d:\\kaoshi\\test.txt");`
- (c) `ifstream file;file.open("d:\\kaoshi\\test.txt");`
- (d) `ifstream *pFile=new ifstream("d:\\kaoshi\\test.txt");`

4. 语句 `ofstream f("SALARY.DAT",ios_base::app)`的功能是建立流对象 f，并试图打开文件 SALARY.DAT 与 f 关联，而且（ ）。

- (a) 若文件存在，将其置为空文件；若文件不存在，打开失败
- (b) 若文件存在，将文件指针定位于文件尾；若文件不存在，建立一个新文件
- (c) 若文件存在，将文件指针定位于文件首；若文件不存在，打开失败
- (d) 若文件存在，打开失败；若文件不存在，建立一个新文件

5. C++流中重载了运算符<<，它是一个（ ）。

- (a) 用于输出操作的成员函数 (b) 用于输入操作的非成员函数
- (c) 用于输入操作的成员函数 (d) 用于输出操作的非成员函数

二、填空题

1. 运行以下程序输出结果为_____。

```
#include <iostream>
using namespace std;
int main( )
{
    cout<<true<<endl;
    cout<<boolalpha;
    cout<<true<<endl;
    return 1;
}
```

2. 运行下列程序输出结果为_____。

```
#include <iostream>
using namespace std;
class point
{
    double x,y;
public:
    void set(double,double);
    double getx();
    double gety();
};
void point::set(double i,double j)
{
    x=i;y=j;
}
double point::getx(){return x;}
double point::gety(){return y;}
```

```
ostream& operator<<(ostream &out,point &p1)
{
    return out<<'('<<p1.getx()<<','<<p1.gety()<<');
}
int main( )
{
    point p1;
    double x=3.1,y=4.5;
    p1.set(x,y);
    cout<<"p1="<<p1<<endl;
    return 1;
}
```

3. 重载输入/输出运算符函数一般被定义为类的_____函数。

4. 成员函数_____和_____用于设置和清除格式状态标志。

5. 标准错误流的输出发送给流对象_____或_____。

6. 运行如下程序，当输入”hello mary!”时，输出为_____。

```
#include<iostream.h>
void main()
{
    char str[100];
    cin>>str;
    cout<<str;
}
```

7. 进行文件操作时需要包含头文件_____。

8. cin 是_____的一个对象，处理标准输入。cout、cerr、clog 是_____的对象。

9. 文件流的成员函数 good() 的作用是 _____;fail() 的作用是 _____;bad()的作用是_____。

10. 运行下列程序的结果为_____。

```
#include <iostream.h>
void main(void)
{
    int i;
    for (i = 0; i < 3; i++)
    {
        cout.fill('*');
        cout.width(5 + i);
        cout << i << '\n';
    }
}
```

11. 运行下列程序的结果为_____。

```
#include <iostream.h>
#include <iomanip.h>
void main(void)
{
    int i;
    cout << "Right justification\n";
    for (i = 0; i < 3; i++)
    {
        cout.width(5);
        cout << setiosflags(ios::right) << i;
    }
}
```

```

cout << "\nLeft justification\n";
for (i = 0; i < 3; i++)
{
    cout.width(5);
    cout << setiosflags(ios::left) << i;
}
}

```

12. 运行下列程序结果为_____。

```

#include <iostream.h>
void main(void)
{
    cout.setf(ios::hex);
    cout.setf(ios::showbase);
    cout << 100;
}

```

三、编程题

1. 编写程序，将 0-255 之间的所有字符写入文件 chars.txt 中。
2. 编写程序，将文件 old.txt 中的所有行后加上句号后写入文件 new.txt 中。

【参考答案】

一、选择题

1. b
2. a
3. a
4. b
5. d

二、填空题

1. 1
true
2. p1=(3.1,4.5)
3. 友元
4. setf、unsetf
5. cerr、clog
6. hello
7. fstream.h
8. istream、ostream
9. 判断刚进行的操作是否成功,判断刚进行的操作是否失败,判断是否进行了非法操作。
10. ****0
*****1
*****2

11.

```

Right justification
    0    1    2

```

```

Left justification
0    1    2

```

12. 0x64

三、编程题

1. 参考程序如下:

```

#include<process.h>
#include<fstream.h>
void main()
{
    ofstream out("chars.txt");
    if(!out)
    {

```

```

        cout<<"不能打开输出文件"<<endl;exit(0);
    }
    int ch;
    for(ch=0;ch<=255;ch++)out.put((char)ch);
    out.close();
}

```

2. 参考程序如下:

```

#include<fstream.h>
#include<stdlib.h>
void main()
{
    fstream file1,file2;
    char buffer[100];
    int i=0;
    file1.open ("old.txt",ios::in);
    if(!file1)
    {cout<<"can not open this file!"<<endl;exit(2);}
    file2.open("new.txt",ios::out);
    while(!file1.eof())
    {
        file1.getline(buffer,sizeof(buffer));
        file2<<buffer<<". "<<endl;
    }
    cout<<"OK!"<<endl;
}

```


第 13 章 异常处理

【内容提要】

异常是软件不可回避的问题，正如我们人类生活在大自然中一样，不可以不考虑风险，并对风险进行规避，否则，稍微大点的意外就会无法让我们幸福地生活。程序设计中的异常处理也就是用于捕捉（感知）异常，并对相应的异常进行处理的机制，异常是程序未按预期执行的紧急预案。举个例子来说明我们现实生活中的异常，比如你的室友请你帮忙取个快递，他告诉你个快递号，你去快递服务中心取件。如果一切顺利的话，你很快就能拿回快件；但若遇到一些麻烦，你未能取回快件，这就是一个异常。对于这么一个异常，取快件的你有什么办法处理吗，因为你只知道很少的信息（快递号），对货物的类型，什么时候的快递，姓名，电话一无所知，那怎么办呢？比较好的办法就是通知你的室友，你未能取回快递。光这样说似乎也不合适，你是不是应该把相应的细节告知你的室友，这个过程称之为抛出异常（throw exception）。异常抛出后，你的室友应该要对异常进行捕获（catch），并进行处理。比如：是快递号不正确，那我可以告诉你运单号，去查询；若是快递名称搞错了，告诉你正确的名称；等等其他的异常。同时呢，捕获异常是有性能代价的，我们只监听部分语句抛出的异常，因此我们需要用 try 包含这些语句。以上就是异常处理的思想，简而言之，异常处理就是增强了程序的稳健性，处理提供程序在正常运行状态下的代码（ordinary code），还提供了面对各种异常（突发情况）准备的修复代码（error code），以确保软件更稳健的运行。本章内容如下：

- ①异常处理的基本思想
- ②异常处理的实现方法

【重点与难点】

13.1 异常处理的基本思想

异常就是程序在运行的过程中，由于使用环境的变化以及用户的操作而产生的错误。常见的异常有 new 无法取得所需的内存、请求打开硬盘上不存在的文件、程序中出现了以零为除数的错误、数组下标越界、运算溢出、无效函数参数、打印机未打开导致程序运行中挂接设备失败等。一个应用程序，既要保证其正确性，还应有容错能力。

异常处理机制保证了当应用环境出现意外或用户操作不当时，程序能够作出合理的反应。其基本思想是当一个函数在发现了（try）自己无法处理的错误时抛出（throw）一个异常，希望它的（直接或间接）调用者能够处理这个异常。能够处理这类异常的函数可以表明它将捕获（catch）这类异常。

异常处理的过程是：在底层发生的问题，逐级上报，直到有能力可以处理异常的那一级为止。如果程序最终没有相应的代码处理该异常，那么该异常最后被 C++ 系统所接受，C++ 系统就简单地终止程序运行。

13.2 异常处理的实现方法

C++ 中异常处理的具体步骤：

- ① 定义异常（try 语句块）

格式为：

```
try
{
    可能产生错误的语句
```

- }
- ② 定义异常处理（**catch** 语句块）
- 格式为：
- ```
catch（异常类型声明 1）
{
 异常处理语句块 1
}
catch（异常类型声明 2）
{
 异常处理语句块 2
}
.....
catch（异常类型声明 n）
{
 异常处理语句块 n
}
```
- ③ 抛弃异常（**throw** 语句）
- 格式为：
- throw** 表达式；
- 说明：
- ① **try** 块之后必须紧跟一个或多个 **catch**，对异常进行相应得处理。
  - ② **catch** 后的括号中只能有一个形参，且该形参的类型是必要的。因为抛出异常后，该异常会按照其数据类型与 **catch** 后括号中的形参类型严格匹配从而被捕获。如果找不到数据类型匹配的 **catch** 块，则系统调用其默认的异常处理程序。
  - ③ **catch(...)**语句可以捕获全部异常，因此，若使用这个语句，应将它放置在所有的 **catch** 语句之后。
  - ④ 当 **catch** 捕获了某个异常后，**catch()**语句的异常类型参数被初始化，然后开始栈的展开过程，包括从对应的 **try** 语句块开始到异常被抛出之间对构造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。
  - ⑤ **throw** 语句用于抛出异常，由于 C++使用数据类型来区分不同的异常，因此在判断异常时，**throw** 语句中的表达式的值没有实际意义而表达式的类型非常重要。
  - ⑥ 抛出异常和异常处理可以放在不同的函数中。

## 【典型例题】

例题 1. 运行下列程序的结果为\_\_\_\_\_。

```
#include <iostream.h>
void testfun(int test)
{
 try
 {
 if(test)
 throw test;
 else
 throw "it is a zero";
 }
 catch(int i)
 {
 cout<<"Except occurred: "<<i<<endl;
 }
}
```

```

 catch(const char *s)
 {
 cout<<"Except occurred: "<<s<<endl;
 }
 }
}
int main()
{
 testfun(10);
 testfun(100);
 testfun(0);
 return 0;
}

```

解答:

本题主要考查对异常处理过程的理解。主函数第一次调用 testfun 函数时传递参数 10，10 不等于 0，所以执行抛出异常语句 throw 10；10 为 int 型，所以被 catch(int i)捕获，所以输出 Except occurred:10。同理第二次调用 testfun 函数时输出 Except occurred:100。第三次调用 testfun 函数时参数为 0，所以执行抛出异常语句 throw "it is a zero";该异常被 catch(const char \*s)捕获，输出 Except occurred: it is a zero。因此，本题答案为：

Except occurred:10

Except occurred:100

Except occurred: it is a zero

例题 2. 求一元二次方程  $ax^2+bx+c=0$  的根，其中系数 a,b,c 为实数，由键盘输入。要求使用异常机制。

解答:

当 a 为 0 时，则该方程不是一元二次方程，而是一元一次方程。当  $\Delta < 0$  时，方程无实根。程序应当能够捕获这两种异常。在设计程序时应在这两种情况发生时抛出异常，并能够捕获所抛出的异常并对不同异常做出相应的处理。

参考程序如下:

```

#include <iostream.h>
#include <math.h>
void Root(double a,double b,double c)
{
 double x1, x2, delta;
 delta=b*b-4*a*c;
 if(a==0) throw "divide by zero";
 if(delta<0) throw 0;
 x1=(-b+sqrt(delta))/(2*a);
 x2=(-b-sqrt(delta))/(2*a);
 cout<<"x1="<<x1<<endl<<"x2="<<x2<<endl;
}
int main()
{
 double a, b, c;
 cout << "please input a, b, c = ? ";
 cin >> a >> b >> c;
 try
 {
 Root(a,b,c);
 }
 catch(char *)
 {
 cout<<"Except occurred: it is not a quadratic equation. "<<endl;
 }
 catch(int)

```

```

 {
 cout<<"Except occurred: the real root of this equation does not exist. "<<endl;
 }
 return 0;
}

```

例题 3. 编写程序，打开用户指定的文件，读出其中的各行内容并显示在屏幕上。要求进行如下异常处理：当文件不存在而打不开时，认为是一种错误，此时通过抛掷与捕获异常来处理该错误。要求程序抛掷、捕获“char\*”类型的异常并进行处理，输出相关的提示警告信息后不再进行相关文件的后继处理，通过 exit 退出程序而结束。若打开文件成功，则将文件中各行内容读出并显示在屏幕上。

解答：

参考程序如下：

```

#include<fstream.h>
#include<process.h>
void main()
{
 char fname[30],line[129];
 cout<<"input file name:";
 cin>>fname;
 ifstream inf(fname,ios::nocreate);
 try{
 if(inf.fail())
 throw fname;
 }
 catch(char *str)
 {
 cout<<"can not open file "<<str<<endl;
 exit(1);
 }
 while(!inf.eof())
 {
 inf.getline(line,128);
 cout<<line<<endl;
 }
 inf.close();
}

```

## 【习题】

### 一、选择题

1. 下列说法错误的是（ ）。
  - (a) 如果 try 块中没有抛出异常，则 try 块执行完后忽略该 try 块的异常处理器 catch 块，程序在最后一个 catch 块后恢复执行。
  - (b) 如果在 try 块以外抛出异常，程序将被终止。
  - (c) try 块抛出异常后，从对应的 try 块开始到异常被抛出之间所构造的所有自动对象将被析构。
  - (d) 抛出异常和异常处理必须放在同一个函数中。

### 二、填空题

1. throw;语句的作用是\_\_\_\_\_。
2. catch(...)的作用是\_\_\_\_\_,其缺点是\_\_\_\_\_。
3. 运行下列程序结果为\_\_\_\_\_。

```

#include <iostream.h>
void fun(int i)
{
 if(i==0)
 throw i;
 if(i==1)
 throw 'a';
 if(i==2)
 throw "china";
}
void main(void)
{
 try
 {
 fun(1);
 }
 catch(int i)
 {
 cout << "Caught an integer." << endl;
 }
 catch(char c)
 {
 cout << "Caught a character." << endl;
 }
 catch(char* s)
 {
 cout << "Caught a string." << endl;
 }
}

```

4. 运行下列程序结果为\_\_\_\_\_。

```

#include <iostream>
#include <exception>
using namespace std;
void fun() throw(exception)
{
 try
 {
 throw exception();
 }
 catch(exception e)
 {
 cout<<"exception handle in fun "<<endl;
 throw;
 }
}
void main()
{
 try{
 fun();
 cout<<"in main"<<endl;}
 catch(exception e)
 {
 cout<<"exception handle in main"<<endl;
 }
 cout<<"finish"<<endl;
}

```

5. 运行下列程序结果为\_\_\_\_\_。

```
#include <iostream.h>
void fun(int &i)
{
 try {
 if(i==0)
 throw "divide by zero! ";
 else
 i/=i;
 }
 catch(char * a) {
 cout << "In fun:" <<a<<endl;
 throw;
 }
 cout<<"fun finished!"<<endl;
}
void main()
{
 int j=0;
 try {
 fun(j);
 }
 catch(char *)
 {
 cout << "In main." << endl;
 }
 cout << "finished!"<<endl;
}
```

6. 运行下列程序结果为\_\_\_\_\_。

```
#include <iostream.h>
void fc()
{
 try
 {throw "sos";}
 catch(int)
 {cout<<"sos int"<<endl;}
 try{throw 1;}
 catch(const char * p){cout<<"sos string"<<endl;}
}
void fb()
{
 int *q=new int[100];
 try{fc();}
 catch(...){
 delete []q;
 throw;}
}
void fa()
{
 int *p=new int[100];
 try{fb();}
 catch(...){
 delete []p;
 throw;
 }
}
```

```

}
void main()
{
 try{fa();}
 catch(...)
 {cout<<"an error occurred while running"<<endl;}
}

```

7. 运行下列程序结果为\_\_\_\_\_。

```

#include<iostream.h>
#include<string.h>
class student
{
 char name[20];
 int age;
public:
 student(char *p,int n)
 {
 strcpy(name,p);
 age=n;
 cout<<"constructing student:"<<name<<endl;
 }
 ~student()
 {
 cout<<"destructing student:"<<name<<endl;
 }
};

```

```

void fun()
{
 try{
 student st1("tom",23),st2("mary",13);
 throw "throw exception in try block!";
 }

 catch(char * s)
 {
 cout<<"handle char * exception:"<<s<<endl;
 }
 cout<<"fun end!"<<endl;
}

```

```

void main()
{
 fun();
}

```

### 三、编程题

编写一个程序完成由于内存不足引起的动态分配存储空间失败的异常处理。

#### 【参考答案】

#### 一、选择题

1. d

#### 二、填空题

1. 再抛出异常

2. 捕获所有异常，catch 没有参数则无法知道错误的原因并且无法引用抛出的错误信息。

3. Caught a character.

4.

exception handle in fun

exception handle in main  
 finish  
 5.  
 In fun:divide by zero!  
 In main.  
 finished!  
 6. an error occurred while running  
 7.  
 constructing student:tom  
 constructing student:mary  
 destructing student:mary  
 destructing student:tom  
 handle char \* exception: throw exception in try block!  
 fun end!

### 三、编程题

参考程序如下:

```
#include<iostream.h>
void main()
{
 int times,count=0;
 struct stru{
 long array[32767];};
 stru *p;
 cout<<"input times:";
 cin>>times;
 try{
 for(int i=0;i<times;i++)
 {
 p=new stru[2000];
 count++;
 if(!p)
 throw "Memory allocation error!";
 else
 cout<<"OK!---"<<count<<" ,Addr="<<p<<endl;
 }
 }
 catch(char *str)
 {
 cout<<"Error occurring:"<<str<<endl;
 }
 cout<<"End main function!"<<endl;
}
```

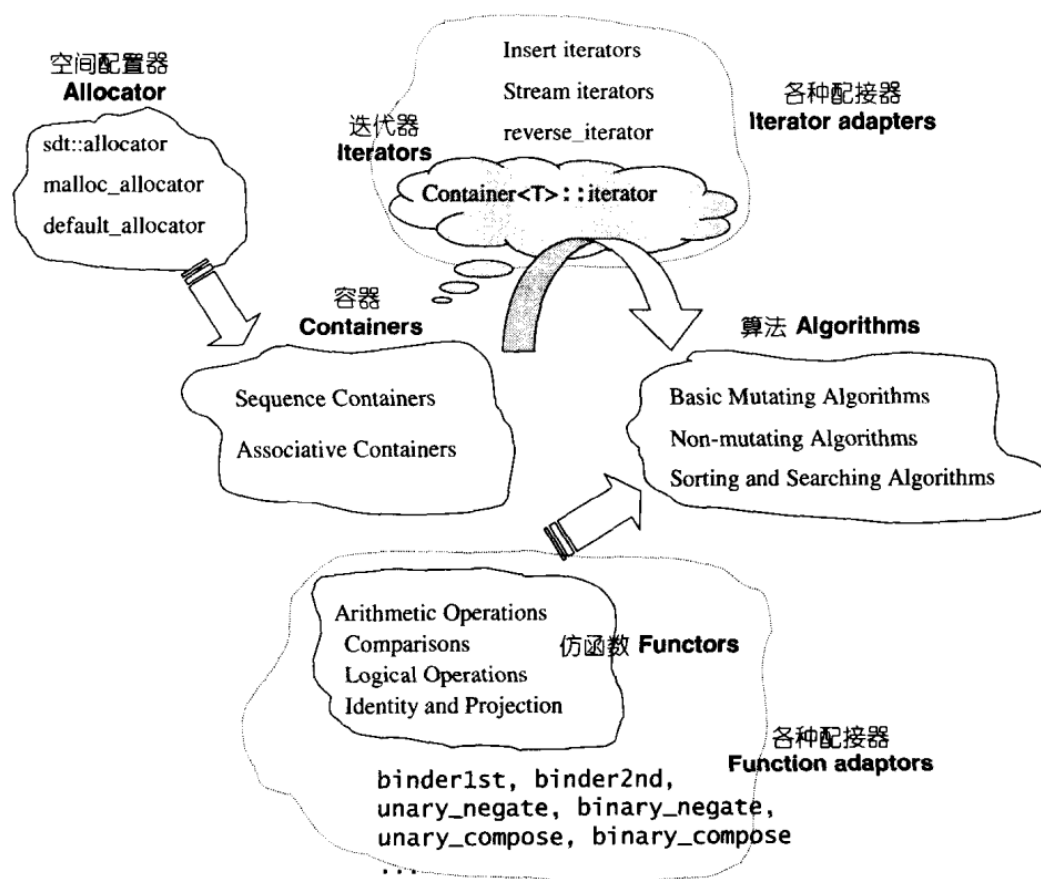


# 第 14 章 STL

## 【内容提要】

STL – Standard Templates Library 标准模板库是 C++ 的标准库之一，尽管 STL 成为 C++ 的标准比较侥幸，但 STL 所具有的泛型编程思维，为面向对象程序设计注入了一股清流。

从使用者的角度来说，STL 包含 2 个重要的部分：容器(containers)和算法(Algorithms)。从 STL 的组成来说，除了上述 2 个重要的部分外，还包括 allocator (内存管理器)、Adaptors (各类适配器，包括函数适配器、容器适配器等)、Iterators (迭代器) 以及 Functors (仿函数)。下图是 STL 各部分的联系图。



从容器说起，容器包含 2 大类，一类是顺序容器 (Sequence containers)，包括 `vector`、`list`、`deque` 等，`stack`、`queue` 属于容器适配器的范畴；另一类是关联容器 (Associative Containers)，包括：`set`、`map` 和 `multi_set`、`multi_map` 等。