自考本科专业课
软件开发工具（课程代码:04737）
通关宝典（讲义）

C++程序设计

通关宝典（又称全书讲义），是尚德老师们结合近年考试动态，精细研究官方教材和考试大纲，为学员们准备的一份备考利器。相比官方教材，内容精简了三分之二，保留了重要的考点、知识点，极大方便学员在最短时间掌握最核心内容的需要。

所有的馥郁花香，都来自沉潜酝酿。

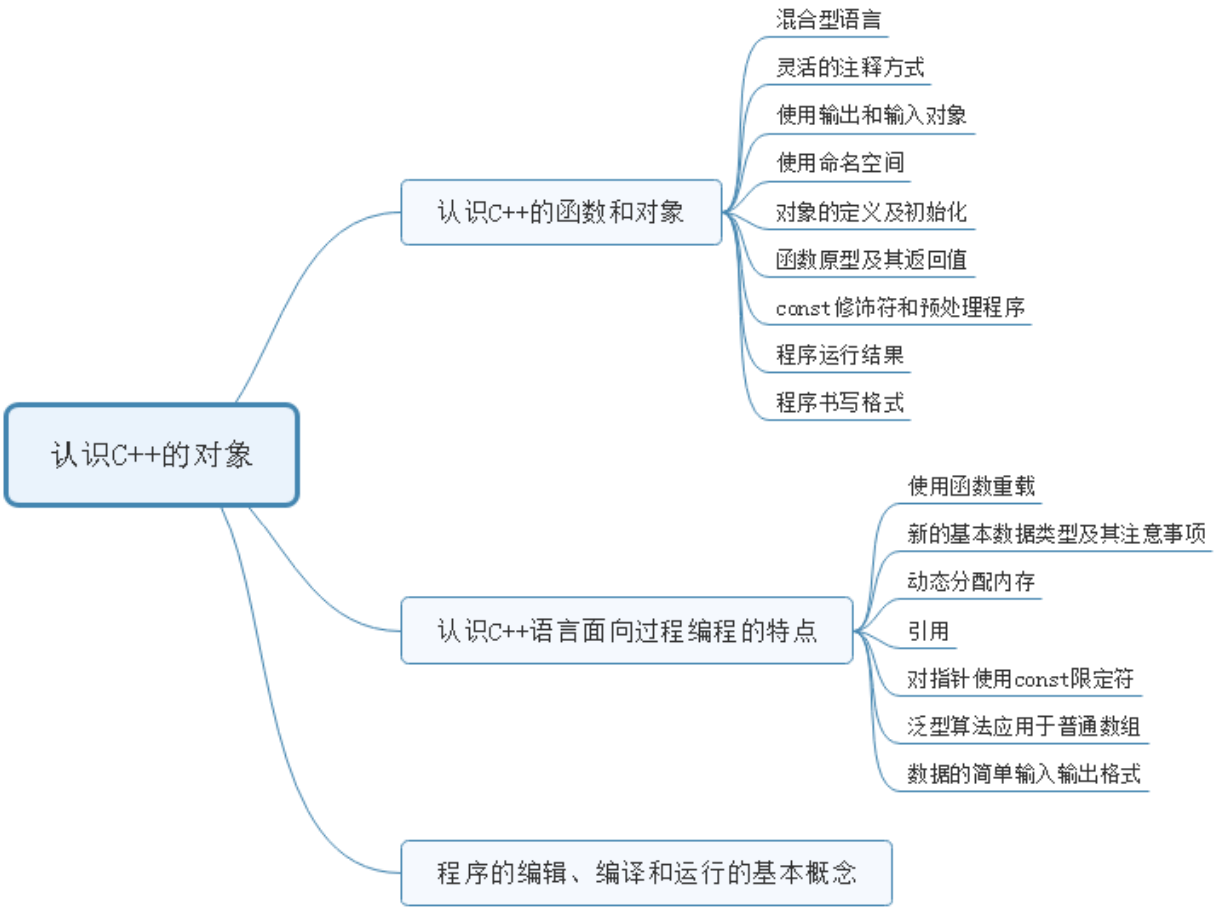
学习是一种信仰，为所有正在阅读本书的城市奋斗者喝彩，祝所有的尚德学子学有所获，期待我们举杯相庆的那一天。

目 录

目录

第一章　认识 C++的对象

思维导图



模块一　认识 C++的函数和对象

知识点一　混合型语言

C++以 .cpp 为文件扩展名，有且只有一个名为 main 的主函数，因保留了这个面向过程的主函数，所以被称为混合型语言。

知识点二　注释方式

（1）“/*”开始，到“*/”结束，如：/* …… */
（2）从“//”开始到本行结束，如：　　//……

知识点三　输入输出对象

（1）提取操作：用提取操作符“>>”从 cin 输入流中提取字符，如：　cin >> a.x；
（2）插入操作：用插入操作符“<<”向 cout 输出流中插入字符，如：cout << “we”；　cout << endl；
（3）使用标准输入（键盘输入）cin 及标准输出（屏幕输出）cout 前，要在主函数前使用 # include <iostream> 将 C + + 标准输入输出库头文件 iostream 将其包括。
（4）换行操作：用语句 cout<<endl；或 cout<<“\n”；实现，其中 endl 可以插在流的中间。如：cout<<a.x<<endl<<a.y<<endl；

知识点四　使用命名空间

C + + 相比 C 而言，可以省略“.h”标识头文件，但必须使用语句 using namespace std；来让命名空间中的对象名称曝光。因此一般的程序都要具有下面的两条语句：

```
#include <iostream>              //包含头文件
using namespace std；            //使用命名空间
```

注意 C + + 库中代替 C 库中头文件的正确名称，例如下面两个语句效果一样：

（1）include <math.h>
（2）#include <cmath>
using namespace std;

知识点五　对象的定义和初始化

定义对象包括为它命名并赋予它数据类型，一般即使初值只用来表示该对象尚未具有真正意义的值，也应将每个对象初始化。

C + + 中利用构造函数语法实现初始化，如：

int z(0)；　　//等同于 int z = 0；

知识点六　函数原型及其返回值

（1）C + + 使用变量和函数的基本规则都是：先声明，后使用。变量有时也可边声明边使用，但必须声明，否则出错。

比如对函数的调用，要在主函数之前先对调用的函数进行原型声明，如：int result （int，int）； 它向编译系统声明，后面有一个 result 函数，该函数有两个整型类型的参数，函数返回整型值。

函数声明时，除了默认参数（需给出默认参数的默认值）和内联函数（需给出函数体及其内语句）外，不需给出参数的变量名称，如果给出，效果也一样，如：int result （int a，int b）； 和上面的声明效果一样。

（2）除构造函数与析构函数外，函数都需要有类型声明。

如 int main（ ），指出 main 是整数类型，返回值由 return 后面的表达式决定，且表达式的值必须与声明函数的类型一致。

如果函数确实不需要返回值，还可用 void 标识，一旦使用 void 标识，函数体内就不再需要使用 return 语句，否则会编译出错，但可使用 return；语句。

（3）C + + 函数有库函数（标准函数，引用时函数名外加< >）和自定义函数（引用时函数名外加“ ”）两类。

知识点七　const（常量）修饰符及预处理程序

（1）const 修饰符：用于定义符号常量。

C 中一般使用宏定义“# define”定义常量，而 C + + 中除此外，建议使用 const 代替宏定义，用关键字 const 修饰的标识符称为符号常量。

因 const 是放在语句定义之前的，因此可以进行类型判别，这比用宏定义更安全一些。如下面两个语句是等同的，但是后者可以比前者避免一些很难发现的错误。

difine BOFSIZE 100

const int BUFSIZE 100；

常量定义也可使用构造函数的初始化方法，如：

const int k（2）；　　//等同于 const int k = 2；

因被 const 修饰的变量的值在程序中不能被改变，所以在声明符号常量时，必须对符号常量进行初始化，除非这个变量是用 extern 修饰的外部变量，如：

const int d；× const int d=2; √ extern const int d；√

const 的用处不仅是在常量表达式中代替宏定义，如果一个变量在生存期内的值不会改变，就应该用 const 来修饰这个变量，以提高程序安全性。

（2）预处理程序

C＋＋的预处理程序不是 C＋＋编译程序的一部分，它负责在编译程序的其他部分之前分析处理预处理语句，为与一般的 C＋＋语句区别，所有预处理语句都以位于行首的符号“#”开始，作用是把所有出现的、被定义的名字全部替换成对应的“字符序列”。

预处理语句有三种：宏定义、文件包含（也成嵌入指令）和条件编译。

文件包含是指一个程序把另一个指定文件的内容包含进来，书写时可以使用引号也可以使用尖括号，前者引用自己定义的包含文件，如：# include “E:\prog\myfile.h” ,后者引用系统提供的包含文件，如标准输入输出是定义在标准库 iostream 中的，引用时要包括以下两条语句：

```
#include <iostream>                //包含头文件
using namespace std；              //使用命名空间
```

知识点八 程序书写格式

C＋＋的格式和 C 一样，都很自由，一行可以写几条语句，但也要注意以下规则，增加可读性：

（1）括号紧跟函数名后面，但在 for 和 while 后面，应用一个空格与左括号隔开；

（2）数学运算符左右各留一个空格，以与表达式区别；

（3）在表示参数时，逗号后面留一个空格；

（4）在 for、do…while 和 while 语句中，合理使用缩进、一对花括号和空行；

（5）适当增加空行和程序注释以增加可读性；

（6）太长的程序分为两行或几行，并注意选取合适的分行和缩进位置。

模块二 认识 C＋＋面向过程编译的特点

知识点一 使用函数重载

C＋＋允许为同一个函数定义几个版本，从而使一个函数名具有多种功能，这称为函数重载。

假设有一个函数 max，分别具有以下函数原型：

```
int max（int，int）；        //2 个整型参数的函数原型
```

int max（int，int，int）；//3 个整型参数的函数原型

只要分别为不同参数的 max 编制相应的函数体，就可以实现各自的功能。

知识点二　新的基本数据类型及其注意事项

（1）void 是无类型标识符，只能声明函数的返回值类型，不能声明变量。

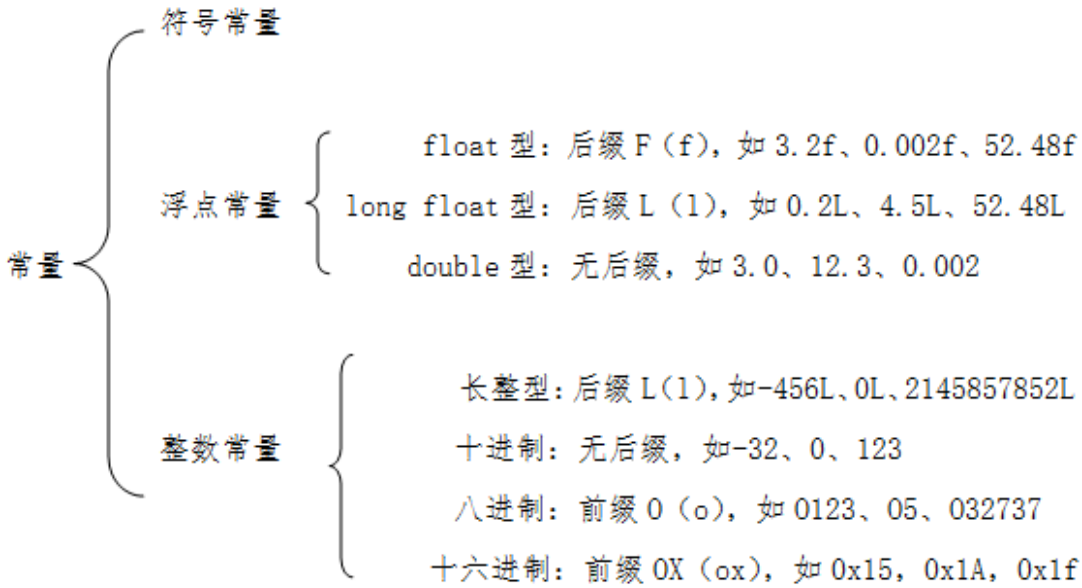
（2）C++还比 C 多了 bool（布尔）型。

（3）C++只限定 int 和 short 至少要有 16 位，而 long 至少 32 位，short 不得长于 int，int 不能长于 long，VC++6.0 规定 int 使用 4 字节，这与 C 使用 2 字节不同。

（4）地址运算符“&”用来取对象存储的首地址，对于数组，则数组名就是数组的首地址。

如：int x=56；定义 x，VC++6.0 使用 4 个字节存储对象 56，假设存放的内存首地址用十六进制表示为 006AFDEC，则语句 cout<<&x；自动使用十六进制输出存储的首地址 006AFDEC。

（5）C++中的常量分三种，第一种为符号常量；第二种为整数常量，有 4 种类型，分别为十进制、长整型（后缀 L（l））、八进制(前缀 O)、十六进制(前缀 OX)，并用前缀和后缀进行分类标识；第三种为浮点常量，有三种类型，分别为 float 型、long float 型、double 型，并用后缀进行分类识别。



（6）C++与 C 一样，也使用转义序列。如：’\0’表示 ASCII 码值为零的空字符（NULL），’\101’表示字符 A。

知识点三 动态分配内存

（1）在使用指针时，如果不使用对象地址初始化指针，可以自己给它分配地址。

对于只存储一个基本类型数据的指针，申请方式如下：

new 类型名[size] //申请可以存储 size 个该数据类型的对象

不再使用时，必须使用 delete 指针名；来释放已经申请的存储空间。

如：……

```
double *p；                    //声明 double 型指针
p = new double[3]          //分配 3 个 double 型数据的存储空间
……
delete p；                     //释放已申请的存储空间
……
```

（2）C 必须在可执行语句之前集中声明变量，而 C + + 可以在使用对象时再声明或定义。

（3）C + + 为结构动态分配内存一般格式为：

```
指针名 = new 结构名；        //分配
delete  指针名；               //释放
```

例如给书中例 1.1 的 Point 结构指针分配内存：

```
p = new Point；
```

当不再使用这个空间时，必须使用 delete p；释放空间。

知识点四 引用

（1）引用简单的说，就是为现有的对象起个别名，别名的地址与引用对象的地址是一样的。

引用的声明方式为： 数据类型 & 别名 = 对象名； ，注意对象在引用前必须先初始化，另外声明中符号“&”的位置无关紧要，比如 int& a = x； 、int & a = x； 和 int &a = x； 等效。

例：……

```
int x = 56；        //定义并初始化 x
int & a = x；       //声明 a 是 x 的引用，二者地址相同
int &r = a；        //声明 r 是 a 的引用，二者地址相同
……
r = 25；             //改变 r，则 a 和 x 都同步变化
……
```

（2）所谓“引用”，就是将一个新标识符和一块已经存在的存储区域相关联。因此，使用引用时没有分配新的存储区域，它本身不是新的数据类型。可以通过修改引用来修改原对象，但是不能有空引用，在程序中必须确保引用是和一块正确的存储区域关联。

引用通常用于函数的参数表中或作为函数的返回值。前者因为使用引用作为函数参数不产生临时对象，可提高程序执行效率和安全性（§4.4.3），后者则是因为引用作为函数返回值可用于赋值运算符的左边。

（3）引用实际上就是变量的别名，使用引用就如同直接使用变量一样，引用与变量名在使用的形式上完全一样，引用只是作为一种标识对象的手段。但要注意，可以声明指向变量或引用的指针，如：int *p=&x; √ int &a=x； int * p=&a；√；也可以声明指针对指针的引用，如：int * & p2 =p1； √（式中 p1、p2 是指针，* 声明 p2 是指针，&声明 p2 是 p1 的引用）；但不能声明指针对变量的引用，如：int * &P=&x； × ；不能声明引用的引用，如：int & & r=x； ×；也不能直接声明对数组的引用。

（4）引用的作用与指针有相似之处，它会对内存地址上存在的变量进行修改，但它不占用新的地址，从而节省开销。二者除使用形式不同，本质也不同：指针是低级的直接操作内存地址的机制，可由整型数强制类型转换得到，功能强大但易出错；引用则是较高级的封装了指针的特性，不直接操作内存地址，不可由强制类型转换而得，安全性较高。

（5）虽然不能直接定义对数组的引用，但可以通过 typedef 来间接的建立对数组的引用。如：

```
……
typedef int array[10]；    //定义 int 型数组类型 array
……
array a={12，34，……}；        //定义 array 型数组 a
array & b=a；                  //定义数组 a 的引用 b
……
```

知识点五　对指针使用 const 限定符

（1）左值和右值

左值是指某个对象的表达式，必须是可变的。左值表达式在赋值语句中即可作为左操作数，也可作为右操作数，如：x=56；和 y=x； ，而右值 56 就只能作为右操作数，不能作为左操作数。

某些运算符如指针运算符“*”和取首地址运算符“&” 也可产生左值，例如 p 是一个指针类型的表达式，则“*p”是左值表达式，代表由 p 指向的对象，且可通过“*p=”改变这个对象的值；“&p”也是左值表达式，代表由 p 指向的对象的首地址，且可通过“&p=”改变这个指针的指向。

（2）指向常量的指针（const int *p=&x； “*p=”的操作不成立）

指向常量的指针是在非常量指针声明前面使用 const ,如 :const int *p； ，它告诉编译器，*p 是常量，不能将*p 作为左值进行操作，即限定了“*p=”的操作，所以称为指向常量的指针。如：

```
const int y = 58；
const int *p1 = &y；    //指向常量的指针指向常量 y，y 不能作为左值
int x = 45；
const int *p = &x；      //只能通过左值 x 间接改变*p 的值
```

上式中*p 不能作为左值，但可以通过“x = ”改变 x 的值，间接改变*p 的值，即 const 仅是限制使用*p 的方式，*p 仍然可以作为右值使用，还可以通过运算符&改变指针所指向的地址，但不能改变指针所指向的内存地址中的内容。

（3）常量指针（int * const p = &x； “p = ”的操作不成立）

把 const 限定符放在*号的右边，就可使指针本身成为一个 const，即常量指针。如：

```
int x = 5；
int * const p = &x；
```

式中的指针本身是常量，编译器要求给它一个初始化值，这个值在指针的整个生存期中都不会改变，编译器把 p 看作常量地址，所以不能作为左值，即“p = ”不成立，也就是说不能改变指针 p 所指向的地址。但这个内存地址里的内容可以使用间接引用运算符*改变其值，例如语句 *p = 56； 将上面的 x 的值改变为 56。

（4）指向常量的常量指针

也可以声明指针本身和所指向的对象都不能改变的“指向常量的常量指针”，这时必须要初始化指针。如：

```
int x = 2；
const int * const p = &x；
```

语句告诉编译器，*p 和 p 都是常量，都不能作为左值，即“*p = ”和“p = ”两操作均不成立，这种指针限制“&”和“*”运算符，在实际中很少用。

知识点六　泛型算法应用于普通数组

数组不能作为整体输出，C + + 引入 STL 库提供的泛型算法，大大简化数组操作。所谓泛型算法，就是提供的操作与元素的类型无关。

假设一维数组 a 和 b 的长度均为 Len，数据类型为 Type，则对数组内容的相关操作和语句如下：

（1）数据内容反转：

```
reverse（a，a + Len）；      //数组元素反转排列
```

（2）复制数组内容：

```
copy（a，a + Len，b）；   //将数组 a 的内容原样复制到数组 b
reverse_copy（a，a + Len，b）；  //逆向复制数组 a 中内容到数组 b
```

（3）数组升幂排序：

sort （a，a+Len）； //默认排序方式是升幂

（4）数组降幂排序：

sort （b，b+Len，greater<Type>（））； //数组降幂排序

（5）检索查找数组内容：

find（a，a+Len，value）； //查找数组 a 中是否存在值为 value 的元素

find 函数返回的是位置指针，一般使用判别语句输出查找的内容，如：

Type *x=find（a，a+Len，value）； //x 是类型为 type 的指针

if（x==a+Len） cout<<“没有值为 value 的数组元素”；

else cout<<“有值为 value 的数组元素”；

（6）输出数组的内容

copy（a，a+Len，ostream_iterator<Type>（cout，“字符串”））；可将 ostream_iterator 简单理解为输出流操作符，<Type>表示数组元素的数据类型，本语句将数组内容按正向送往屏幕，输出方式是将每个元素与“字符串”的内容组合在一起连续输出。如果使用空格“ ”或换行符“\n”，可以按格式输出。也可将数组内容按逆向方式送往屏幕，语句为：

reverse_copy(a ,a+Len ,ostream_iterator<Type>(cout ,“字符串”))；

知识点八 数据的简单输入输出格式

（1）C++提供了两种格式控制方式，一种是使用 iso_base 类提供的接口，另一种是使用一种称为操控符的特殊函数，操控符的特点是可直接包含在输入和输出表达式中，因此更为方便，不带形式参数的操控符定义在头文件<iostream>中，带形式参数的操控符定义在头文件<iomanip>中。在使用操控符时，一是要正确包含它们，二是只有与符号“<<”或“>>”连接时才起作用，三是无参数的操控符函数不能带有“（）”号。

（2）常用操控符及其作用

格式	含义	作用
dec	设置转换基数为十进制	输入/输出
oct	设置转换基数为八进制	输入/输出
hex	设置转换技术为十六进制	输入/输出
endl	输出一个换行符并刷新流	输出
Setprecision（n）	设置浮点数输出精度 n	输出
Setw（n）	设置输出数据字段宽度	输出
Setfill（'字符'）	设置 ch 为填充字符	输出
Setiosflags（flag）	设置 flag 指定的标志位	输出
resetiosflags（flag）	清除 flag 指定的标志位	输出

参数 flag 常引用的枚举常量及其含义

常量名	含义
ios_base::left	输出数据按输出域左边对齐输出
ios_base::right	输出数据按输出域右边对齐输出
ios_base::showpos	在正数前添加一个“+”号
ios_base::showpoint	浮点输出时必须带有一个小数点
ios_base::scientific	使用科学计数法表示浮点数
ios_base::fixed	使用定点形式表示浮点数

模块三　程序的编辑、编译和运行的基本概念

知识点一　C++程序编制过程

（1）先使用编辑器编辑一个 C++程序 mycpp.cpp，又称其为 C++的源程序；

（2）然后使用 C++编译器对这个 C++程序进行编译，产生文件 mycpp.obj；

（3）再使用连接程序（又称 Link），将 mycpp.obj 变成 mycpp.exe 文件。

知识点二　C++程序编制环境及使用方法

现在 C++的编制一般都使用集成环境，如 Visual C++6.0 等，所谓集成环境，就是将 C++语言的编辑、编译、连接和运行程序都集成到一个综合环境中。

利用 VC 编制 C++程序源文件的步骤如下：

（1）启动 VC6.0；

（2）File 菜单 - New 对话框 - Project 选项卡 - Win32 Console Application 选项，在右边的 Project name 输入框中输入项目名称 myfile，在右边的 Location 输入框中输入存储目录，然后单击 OK 按键，进入 Win32 Console Application 制作向导的第一步，编辑 C++程序文件是选择 An empty project 选项，单击 Finish 按钮，完成设置；

（3）选中 File View 选项卡，进入空项目，单击它展开树形结构，选中 myfile

files 节点；选中 Source File 标记，再从 File 菜单中选 new 命令，弹出 new 对话框；选中 C＋＋Source File 选项，在右方的 File 输入框中输入 C＋＋程序文件名（mycpp），系统默认的文件扩展名为.cpp，单击 OK 按钮，返回集成环境，并在 Source File 项下面产生空文件 mycpp.cpp；在右边的源代码编辑框中输入源文件；

真题演练

1、（201704 单选题）以下说法中不正确的是（ ）

A、C++程序中必须有一个主函数 main(),而且是从 main()的第一条语句开始执行

B、非主函数都是在执行主函数时，通过函数调用或嵌套调用而得以执行的

C、主函数可以在任何地方出现

D、主函数必须出现在固定位置

答案：D

解析：C++程序以.cpp 作为文件扩展名，并且必须有一个且只能有一个名为 main 的主函数。并无固定位置。D 错误。

2、（200901 单选题）C++源程序文件扩展名为( )

A、.cpp　　B、.h　　C、.lib　　D、.obj

答案：A

解析：C++程序以.cpp 作为文件扩展名。

3、（201410 单选题）用于标识十六进制常量的前缀或后缀是（ ）。

A、无　　B、后缀 L 或 l　　C、前缀零　　D、前缀 0x

答案：D

解析：十六进制常量前缀为 0x。

4、（201410 单选题）下列表达式，哪个是声明 P 为指向常量的常量指针（ ）。

A、const int*P　　B、int*const P

C、const int* const P　　D、int*P

答案：C
解析：可以声明指针和指向的对象都不能改动的“指向常量的常量指针”，例如：const int* const P。

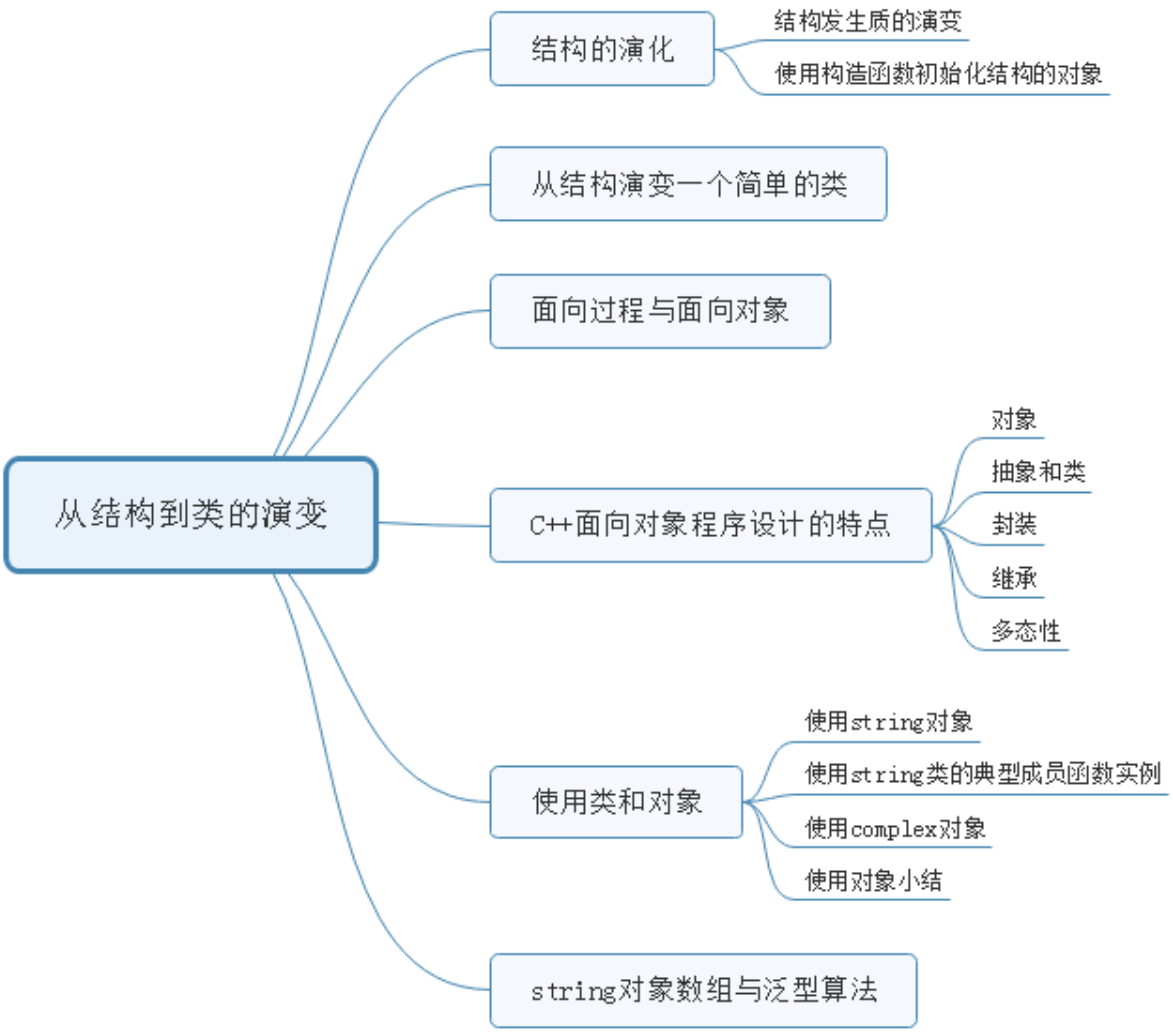
5、（200810 单选题）对使用关键字 new 所开辟的动态存储空间，释放时必须使用（ ）
A、free　　B、create　　C、delete　　D、realse

答案：C
解析：使用 delete 关键字释放空间。

本章要求熟悉 C++的基本程序结构。初步理解面向对象程序设计的思想及 C++语言中的新思想。要求初步掌握使用函数重载的方法，以便为下一章从结构引入类知识打下基础。重点：函数原型、重载、动态内存分配、引用、对指针使用 const 限定符、泛型算法、C++程序的基本结构、数据的简单输入输出格式。难点：引用、对指针使用 const 限定符、泛型算法。

第二章　从结构到类的演变

思维导图



模块一　结构的演化

知识点一　结构发生质的演变

（1）函数与数据共存

C++中首先允许结构中可以定义函数，这些函数称为成员函数，形式如下：

struct 结构名{

数据成员

成员函数

};

可以像 C 语言中结构变量使用结构成员的方法那样，通过 C++的结构对象使用成员函数，形式如下：

结构对象.成员函数

（2）封装性

如果在定义结构时，将数据成员使用 private 关键字定义，则产生封装性，没有使用 private 定义的成员函数，默认为 public。

要注意，私有的数据成员，必须通过公有的成员函数才能使用，而不能不通过公有的成员函数直接来使用，否则就会出错，这就称为数据的封装性。

知识点二　使用构造函数初始化结构的对象

函数名与结构同名，称为构造函数，专门用于初始化结构的对象，构造函数使用的一般形式为：

构造函数名 对象名（初始化参数）；

程序在运行时，会自动完成初始化任务。

模块二　从结构演变一个简单的类

知识点一　用关键字 class 代替 struct，就是一个标准的类。

实例：

```
#include <iostream>
using namespace std;
class Point{          //定义类 Point
private:
   double x,y;   //类 Point 的数据成员
public:
   Point( ){ };   //类 Point 的无参数构造函数
   Point(double a,double b) {x=a;y=b;}     //具有两个参数的构造函数
```

```
    void Setxy(double a,double b) {x=a;y=b;} //成员函数，用于重新设置数据成员
    void Display( ){cout<<x<<"\t"<<y<<endl;} //成员函数，按指定格式输出数据成员
    };
void main(){
    Point a;   //定义类 Point 的对象 a
    Point b(18.5,10.6);   //定义类 Point 的对象 b 并初始化
    a.Setxy(10.6,18.5);   //为对象 a 的数据成员赋值
    a.Display();   //显示对象 a 的数据成员
    b.Display();   //显示对象 b 的数据成员
}
```

程序运行结果：

10.6　18.5

18.5　10.6

知识点二　类的示意图

上例中的 Point 类可以看作直角坐标系中的点类，其结构示意图如右：

第一个方框中是类名；第二个方框中是坐标点的数据，称为属性（或称数据成员）；第三个方框中表示类所提供的具体操作方法，实际上是如何使用数据 x 和 y，以实现预定功能的函数，这里称为成员函数。

模块三　面向过程与面向对象

知识点一　面向过程的方法

所谓面向过程，就是不必了解计算机的内部逻辑，而把精力集中在对如何求解问题的算法逻辑和过程的描述上，通过编写程序把解决问题的步骤告诉计算机。

C 语言就是面向过程的结构化程序设计语言，其程序设计特点就是通过函数设计，实现程序功能的模块化、结构化。但实际工作中，尽管结构化程序设计中的分而治之的想法非常好，但在结构化程序设计语言和结构化程序设计方法下却难以贯彻到底，特别是在软件规模在三四万行以上时，开发和维护就十分困难。

知识点二　面向对象的方法

为了解决面向过程的方法存在的问题，人们提出了面向对象的方法。所谓对象，就是现实世界中客观存在的事务。相对于过程，对象是稳定的，复杂的对象可由简单的对象组成，对象各自完成特定的功能。

在面向对象程序设计中，可以将一组密切相关的函数统一封装在一个对象中，从而合理而又有效的避免全局变量的使用，更彻底的实现了结构化程序设计的思想。

结构化程序设计使用的是功能抽象，而面向对象程序设计不仅能进行功能抽象，还能进行数据抽象，“对象”实际上是功能抽象和数据抽象的统一。

面向对象的程序设计方法不是以函数过程和数据结构为中心，而是以对象代表来求解问题的中心环节，追求的是现实问题空间与软件系统解空间的近似和直接模拟，从而使人们对复杂系统的认识过程与系统的程序设计实现过程尽可能的一致。

知识点三　软件开发过程及发展趋势

软件开发者将被开发的整个业务范围称为问题域，软件开发是对给定问题求解的过程，分为两项主要内容：认识和描述。“认识”就是在所要处理的问题域范围内，通过人的思维，对该问题域客观存在的事务以及对所要解决的问题产生正确的认识和理解。“描述”就是用一种语言把人们对问题域中事务的认识、对问题及解决方法的认识描述出来，最终的描述必须使用一种能够被机器读懂的语言，即编程语言。

模块四　C＋＋面向对象程序设计的特点

知识点一　对象

面向对象的程序设计具有抽象、封装、继承和多态性等关键要素。

C＋＋中的对象是系统中用来描述客观事物的一个实体，是构成系统的一个基本单位，C＋＋中使用对象名、属性和操作三要素来描述对象，如右所示：对象名用来标识一个具体对象；用数据来表示对象的属性，一个属性就是描述对象静态特征的一个数据项；操作是描述对象动态特征（行为）的一个函数序列（使用函数实现操作），也称方法或服务。数据称为数据成员，函数称为成员函数，一个对象由一组属性和对这组属性进行操作的成员函数构成。

知识点二　抽象和类

抽象是一种从一般的观点看待事物的方法，即集中于事物的本质特征，而不是具体细节或具体实现。面向对象的方法把程序看作是由一组抽象的对象组成的，如果把一组对象的共同特征进一步抽象出来，就形成了“类”的概念。

对于一个具体的类，它有许多具体的个体，这些个体叫做“对象”，同一类的不同对象具有相同的行为方式。一个对象是由一些属性和操作构成的，对象的属性和操作描述了对象的内部细节，类是具有相同的属性和操作的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和操作两个主要部分。

类的作用是定义对象，类和对象的关系如同一个模具和其铸造出来的铸造件的关系，对象之间就像是同一模具铸出的零件，模样相同，铸造材料可能不同。类给出了属于该类的全部对象的抽象定义，而对象则是符合这种定义的实体。所谓一个类的所有对象具有相同属性和操作，是指它们的定义形式（即属性的个数、名称、数据类型）相同，而不是说每个对象的属性值都相同。

知识点三　封装

按照面向对象的封装原则，一个对象的属性和操作是紧密结合的，对象的属性只能由这个对象的操作来存取。

对象的操作分为内部操作和外部操作，前者只供对象内部的其他操作使用，不对外提供；后者对外提供一个信息接口，通过这个接口接受对象外部的消息并为之提供操作（服务）。对象内部数据结构的这种不可访问性称为信息（数据）隐藏。

数据封装给数据提供了与外界联讯的标准接口，只有通过这些接口，使用规范的方式，才能访问这些数据，同时程序员也只需要和接口打交道，而不必了解数据的具体细节。

知识点四　继承

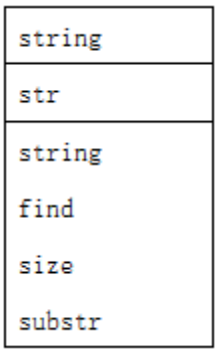
继承是一个类可以获得另一个类的特性的机制，支持层次概念，通过继承，低层的类只需定义特定于它的特征，而共享高层的类中的特征。继承具有重要的实际意义，它简化了人们对事物的认识和描述。

知识点五　多态性

不同的对象可以调用相同名称的函数，但可导致完全不同的行为的现象称为多态性。利用多态性，程序中只需进行一般形式的函数调用，函数的实现细节留给接受函数调用的对象，这大大提高了解决人们复杂问题的能力。

模块五　使用类和对象

知识点一　使用 string 对象



实际上 string 类很复杂，如右的 string 类的简化图中只给出了下例中涉及的属性和部分操作。

由图，类 string 的属性是一个字符串 str，同名函数 string 是构造函数，用来初始化字符串，另外三个成员函数用来对属性 str 进行操作，其中 find 成员函数用来在 str 字符串中检索需要的子串；size 成员函数计算并输出 str 存储的单词长度；substr 成员函数用来返回 str 字符串中的子串。

在程序中可以使用 string 类定义存储字符串的对象，这些对象属于 string 类，因此还要使用#include <string>来包含这个类的头文件。

知识点二　使用 string 类的典型成员函数

string 对象是通过调用成员函数实现操作，从而提供对象的行为或消息传递的，对象调用成员函数的语法为：

对象名称.成员函数（参数（可供选择的消息内容））

（1）成员函数 substr 用来返回给定字符串的子串，格式为：

对象名称.substr（要截取子串的起始位置，截取的长度）；

（2）成员函数 find 用来在主串中检索所需字符串，格式为：

对象名称.find（要查找的字符串，开始查找的位置）；

函数返回查找到的字符串在主串中的位置；

（3）string 类还提供一个辅助功能，以便使用 getline 从流 cin 中读出输入的一行给 string 类的对象

知识点三　使用 complex 对象

C++标准库提供 complex 类定义复数对象，在程序中包含这个类的头文件为：#include <complex>

复数（complex number）类需要两个初始值：实部和虚部，complex 是一个模板类，利用构造函数初始化的格式为：

comlex <数据类型> 对象名（实部值，虚部值）；

如：complex <int> num1(2,3)；　//定义复数 2+3i

complex <float> num2(2.5,3.6)；　//定义复数 2.5+3.6i

complex 的 real 和 imag 成员函数用来输出对象的实部和虚部的值，如：

```
cout<<num2.real( )<<”,” <<num2.imag( )<<endl;
```

知识点四　使用对象小结

标准库提供的类都是经过抽象，代表了一类对象，例如 string 类描述的是字符串特性，具有一个用来描述对象静态性质的字符串，字符串的值可以区分不同的对象；通过一系列的操作方法对这个字符串进行操作，用函数实现这些方法，又称这些函数为成员函数；数据成员（属性）和成员函数（方法）代表了字符串一类事物的特征。

String 类的使用方法一般如下：

```
#include <string>
string str1;  //定义这个类的一个对象 str1
str1 = “this is a string” ;   //给 str1 赋值
string str2 ( “this is a str2” ) ; //定义 str2 并赋值
cout<<str1<<str2.size( );
```

可以先定义对象，然后给它赋值（如上式 str1），也可以在定义对象的同时初始化对象（如上式 str2），如果要使用对象的成员函数，则用“.”运算符（如最后一句）。

同理，复数类可以抽象为具有实部和虚部的数据成员，以及能对复数进行基本操作的成员函数，与 string 不同的是，定义复数类时与数据成员的类型无关，当定义复数类的对象时才指定实部和虚部的数据类型。

注意，类是抽象出一类物质的共同特征，模板则是归纳出不同类型事物的共同操作。

模块六　string 对象数组与泛型算法

知识点一　注意点

第 1.2.6 节介绍的泛型算法同样适合 string 类，但要注意不要将该节介绍的 find 函数与 string 本身的 find 函数混淆。

另外，string 类还有一个 swap 成员函数，用来交换两个对象的属性。假设有两个 string 类的对象 str1 和 str2，下面两种调用方式是等效的：

```
str1.swap(str2);
str2.swap(str1);
```

知识点二　string 类的成员函数 begin 和 end

string 类有一对用来指示其元素位置的基本成员函数：指示第一个元素的 begin 和指示最后一个元素之后的字串结束标记 end。它们标识存储元素的空间。如果 begin 不等于 end，算法首先作用于 begin 所指元素，并将 begin 前进一个位置，然后作用于当前的 begin 所指元素，如此继续前进，直到 begin 等于 end 为止，所以它们是元素存在的半开区间。

真题演练

1、（201110 单选题）面向对象不仅进行功能抽象，还要进行（ ）
A、动态抽象　　B、消息抽象　　C、数据抽象　　D、算法抽象

答案：C
解析：面向对象程序设计不仅能进行功能抽象，而且能进行数据抽象。“对象”实际上是功能抽象和数据抽象的统一。

2、（201504 单选题）结构化程序设计所规定的三种基本控制结构是（ ）。

A、输入、处理、输出
B、树形、网形、环形
C、顺序、选择、循环
D、主程序、子程序、函数

答案：C
解析：结构化程序设计所规定的三种基本控制结构是顺序、选择、循环。
1. 顺序结构
顺序结构表示程序中的各操作是按照它们出现的先后顺序执行的，这种结构的特点是：程序从入口点 a 开始，按顺序执行所有操作，直到出口点 b 处，所以称为顺序结构。
2.选择结构
选择结构表示程序的处理步骤出现了分支，它需要根据某一特定的条件选择其中的一个分支执行。选择结构有单选择、双选择和多选择三种形式。
3.循环结构
循环结构表示程序反复执行某个或某些操作，直到某条件为假（或为真）时才可终止循环。

3、（201604 单选题）在 C++中，类与类之间的继承关系具有（ ）。

A、自反性　　B、对称性　　C、传递性　　D、反对称性

答案：C
解析：通过继承，低层的类只须定义特定于它的特征，而共享高层的类中的特征，即 C 选项的传递性。

4、（201404 单选题）设存在数组 a，其长度为 Len，则下列哪项泛型算法用于在 a 中寻找值 Value 的位置（ ）。
A、reverse(a，a+Len，Value)；
B、sort(a，a+Len，Value)；
C、find(a，a+Len，Value)；
D、copy(a，a+Len，Value)；

答案：C
解析：成员函数 find 用来在主串中检索所需字符串。

5、（200910 单选题）使用 string.h 库操纵字符串时，将两个字符串连接成一个字符串的函数是( )

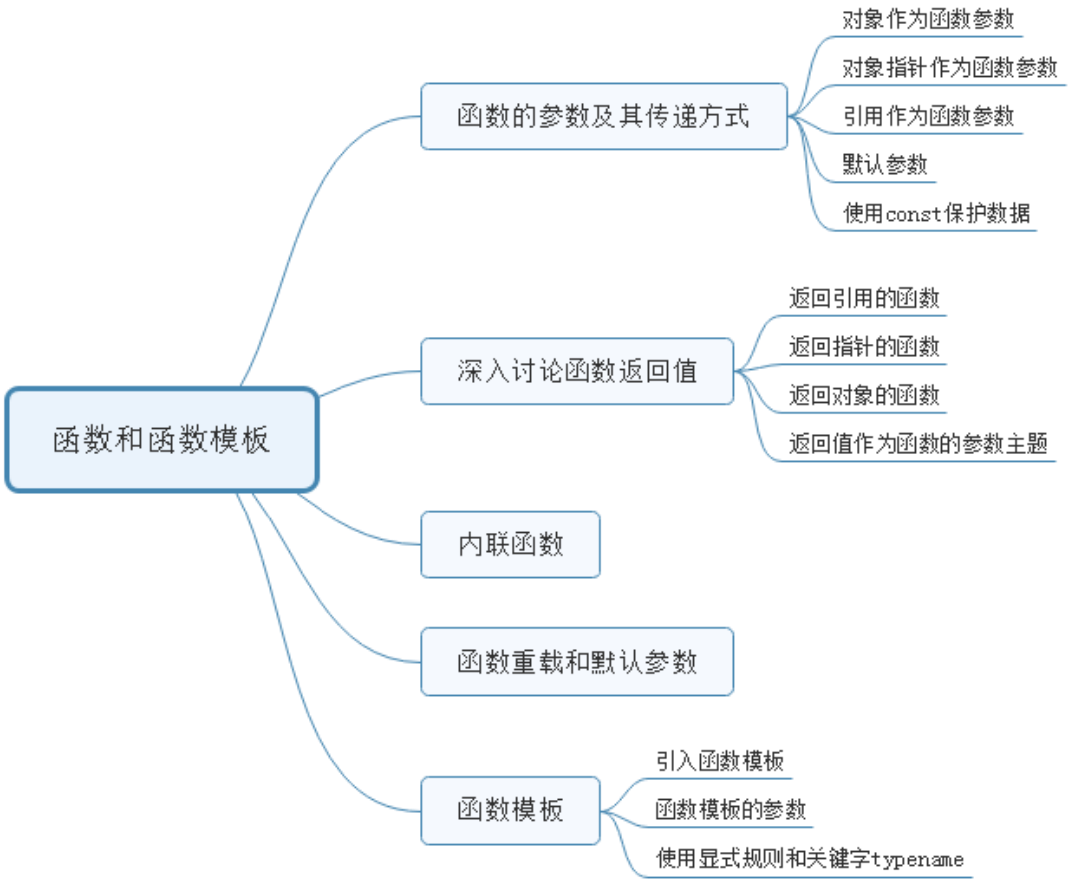
A、strlen()　　B、strcap()　　C、strcat()　　D、strcmp()

答案：C
解析：strcat()：用于连接字符串

本章的目的是通过实例说明结构如何向类变化，从而引入对象和类的知识。要求通过使用 C++标准程序库提供的两个典型的类，进一步熟悉类和对象，建立面向对象的基本概念。重点：引入面向对象程序设计的基本概念，熟悉 string 类的使用方法。难点：使用对象及泛型算法。

第三章　函数和函数模板

思维导图



模块一　函数的参数及其传递方式

知识点一　C + + 的函数参数传递方式

C 语言函数参数的传递方式只有传值一种，又分为传变量值和传变量地址值两种情况，而 C + + 的函数参数传递方式有两种：第一种和 C 语言一样，是传值；第二种是传引用，即传对象的地址，所以也称传地址方式。

注意传地址值传递的是值，是以对象指针作为参数；而传地址传递的是地址，是以对象引用作为参数。

所以在设计函数参数时，可以使用“对象”、“对象指针”和“对象引用”作为参数。

知识点二　对象作为函数参数

使用对象作为函数参数，是将实参对象的值传递给形参对象，传递是单向的，形参具有实参的备份，当在函数中改变形参的值时，改变的是这个备份中的值，不影响原来实参的值。

知识点三　对象指针作为函数参数

使用指向对象的指针作为函数参数，形参是对象指针（指针可以指向对象的地址），实参可以是对象的地址值，虽然参数传递方式仍然是传值方式，但因为形参传递的就是实参本身，所以当在函数中改变形参的值时，改变的就是原来实参的值。

传对象地址值要用到对象的指针，而对于数组，因数组名就是数组的指针名，所以数组也能用传数组地址值的方式。

知识点四　引用作为函数参数

使用引用作为函数参数，在函数调用时，实参对象名传给形参对象名，形参对象名就成为实参对象名的别名。实参对象和形参对象代表同一个对象，所以改变形参对象的值就是改变实参对象的值。

知识点五　默认参数

默认参数就是不要求程序员设定该参数，　而由编译器在需要时给该参数赋默认值。当程序员需要传递特殊值时，必须显式的指明。默认参数必须在函数原

型中说明，默认参数可以多于 1 个，但必须放在参数序列的后部。

知识点六　使用 const 保护数据

可以用 const 修饰要传递的参数，意思是通知函数，它只能使用参数而无权修改参数，以提高系统的自身安全，C + + 中普遍使用这种方法。

模块二　深入讨论函数返回值

c++函数的返回值类型可以是除数组和函数以外的任何类型，非 void 类型的函数必须向调用者返回一个值，数组只能返回地址。当返回值是指针或引用对象时，需要特别注意：函数返回所指的对象必须继续存在，因此不能将函数内部的局部对象作为函数的返回值。

知识点一　返回引用的函数

函数可以返回一个引用，这样的目的是为了将该函数用在赋值运算符的左边，因为其他情况下，一个函数是不能直接用在赋值运算符左边的。

返回引用的函数原型的声明方式为：

数据类型 & 函数名（参数列表）；

知识点二　返回指针的函数

指针函数：返回值是存储某种类型数据的内存地址的函数。

返回指针的函数原型的声明方式为：

数据类型 *函数名（参数列表）；

知识点三　返回对象的函数

返回对象的函数原型的声明方式为：

数据类型 函数名（参数列表）

知识点四　函数返回值作为函数的参数

如果函数返回值作为另一个函数的参数，那么这个返回值必须与另一个函数的参数的类型一致。

模块三　内联函数

知识点一　内联函数的概念

使用关键字 inline 说明的函数称为内联函数，内联函数必须在程序中第一次调用此函数的语句出现之前定义，在 C++中，除具有循环语句、switch 语句的函数不能说明为内联函数外，其他函数都可以说明为内联函数。使用内联函数可以提高程序执行速度，但如果函数体语句多，则会增加程序代码的大小。

模块四　函数重载和默认参数

知识点一　函数重载

函数重载可使一个函数名具有多种功能，即具有“多种形态”，称这种特性为多态性。从函数原型可见，它们的区别一是参数类型不同，二是参数个数不同，所以仅凭返回值不同或仅凭参数个数不同，均不能区分重载函数。在函数重载时，源代码只指明函数调用，而不说明调用具体调用哪个函数，直到程序运行时才确定调用哪个函数，编译器的这种连接方式称为动态联编或迟后联编。

知识点二　函数重载与默认参数

当函数重载与默认参数相结合时，能够有效减少函数个数及形态，缩减代码规模。如果使用默认参数，就不能对参数个数少于默认参数个数的函数形态进行重载，只能对于多于默认参数个数的函数形态进行重载。

模块五　函数模板

知识点一 函数模板

在程序设计时没有使用的实际类型，而是使用虚拟的类型参数，故其灵活性得到加强。当用实际的类型来实例化这种函数时，就好像按照模板来制造新的函数一样，所以称为函数模板。

C++中规定模板以关键字 template 和一个形参表开头，形参表中以 class 表示“用户定义的或固有的类型”，一般选用 T 作为标识符来标识类型参数。

知识点二 函数模板的参数

例如语句“cout<<max<int> (2,5)<<endl;”，属于显式的给出了模板参数的比较准则，函数模板参数的显式规则主要用于特殊场合。

显示比较准则形式为：函数模板名<模板参数> （参数列表）

每次调用都显式的给出比较准则，会使人厌烦，一般可使用下面的默认方式：

函数模板名（参数列表）

但这样的条件是这个调用的函数参数列表能够唯一的标识出模板参数所属的集合，否则，仍要显式的给出比较准则。

知识点二 使用显式规则和关键字 typename

C++中，关键字 typename 仅用于模板中，其用途之一就是代替类模板 template 参数列表中的关键字 class。

真题演练

1、（200810 单选题）使用值传递方式将实参传给形参，下列说法正确的是（ ）

A:形参是实参的备份

B:实参是形参的备份

C:形参和实参是同一对象

D:形参和实参无联系

答案：A

解析：将指向对象的指针作为函数参数，形参是对象指针，实参是对象的地址值。虽然参数传递方式仍然是传值方式，但因为形参传递的就是实参本身，所以当在函数中改变形参的值时，改变的就是原来实参的值。

2、（201404 单选题）使用值传递方式将实参传给形参，下列说法正确的是（ ）

A、形参是实参的备份

B、实参是形参的备份

C、形参和实参是同一对象

D、形参和实参无联系

答案：A

解析：将指向对象的指针作为函数参数，形参是对象指针，实参是对象的地址值。虽然参数传递方式仍然是传值方式，但因为形参传递的就是实参本身，所以当在函数中改变形参的值时，改变的就是原来实参的值。

3、（201404 单选题）一个函数功能不太复杂，但要求频繁使用，则该函数适合作为（ ）。

A、内联函数
B、重载函数
C、递归函数
D、:嵌套函数

答案：A
解析：使用内联函数能加快程序执行速度，但如果函数体语句多，则会增加程序代码的大小当一个函数功能不太复杂，但要求频繁使用时，该函数适合作为内联函数。

4、（201101 单选题）int Func(int，int)；不可与下列哪个函数构成重载( )

A、int Func(int，int，int)；
B、double Func(int，int)；
C、double Func(double，double)；
D、double Func(int，double)；

答案：B
解析：函数重载区别为参数类型不同，参数个数不同。

5、实现两个相同类型数加法的函数模板的声明是（ ）

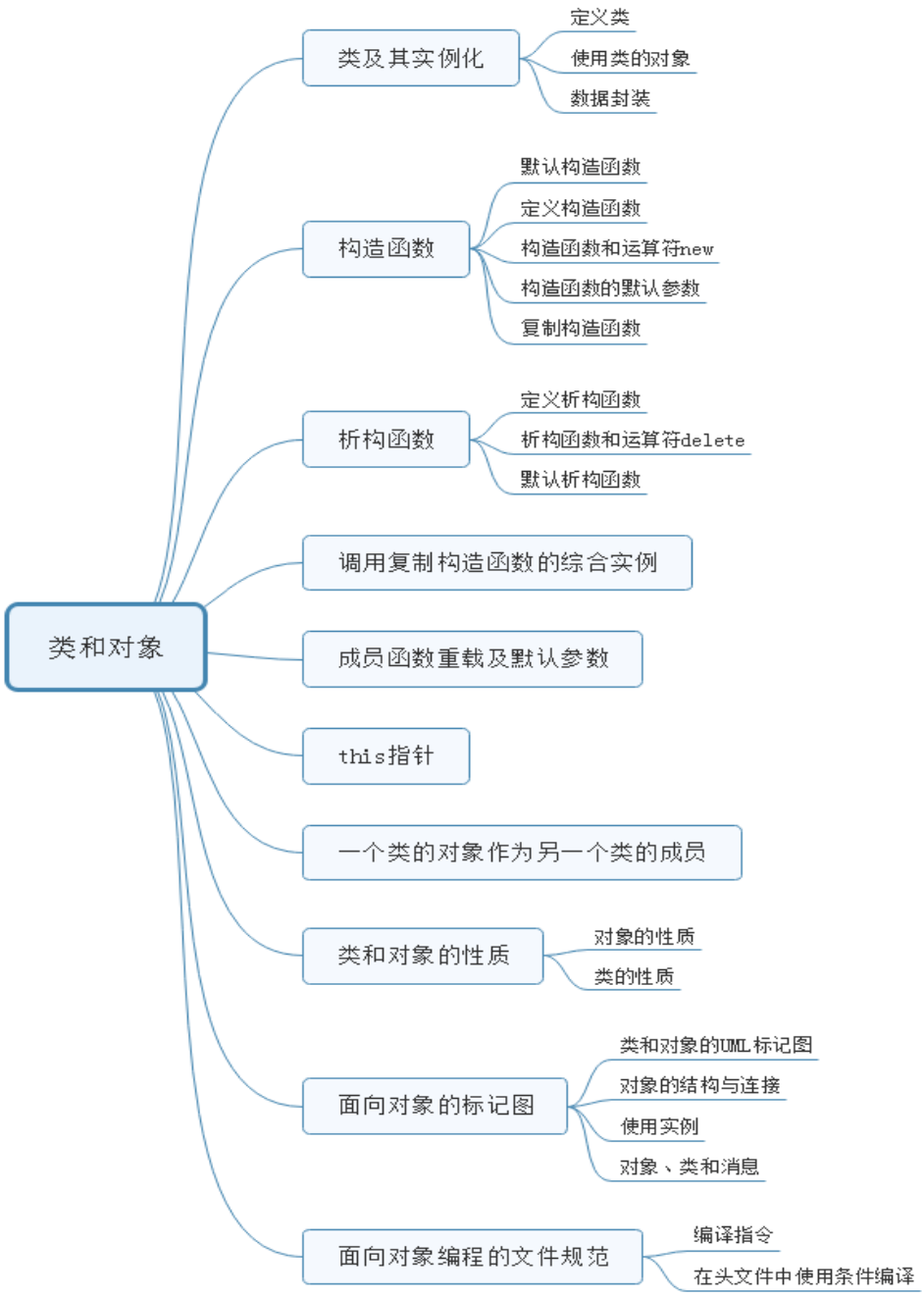
A:add(T x,T y)
B:T add(x,y)
C:T add(T x,y)
D:T add(T x,T y)

答案：D
解析：因为字符 T 能使人联想到它是单词 Type 的第一个字母，所以一般都选 T 作为标识符来标识类型参数，即参数名是可以任意挑选的。变量同样需要修饰，因此选择 D 选项。

本章除了要求掌握函数参数的传递方式和返回值等问题之外，还要求掌握内联函数、函数重载和函数模板。重点：函数参数的传递方式及其返回值、函数重载、默认参数和函数模板。难点：函数重载和默认参数。

第四章　类和对象

思维导图



模块一　类及其实例化

知识点一　类及其实例化

对象就是一类物体的实例，将一组对象的共同特征抽象出来，就形成了类的概念，如从直角坐标系的点抽象出点的概念，这就是 point 类，point 类只是表示类名是 point，它的点位置有两个坐标属性，只有当产生一个具体的坐标点 A，A 的两个属性具有了具体的值，才能代表实际的点，即实例化了。

知识点二　类的定义

类是对一组性质相同的对象的程序描述，也属于一种用户自己构造的数据类型，也要遵循 C + + 的规定，如先声明后使用、是具有唯一标识符的实体、在类中声明的任何成员不能使用 extern、auto 和 register 关键字修饰、类中声明的变量属于该类等。与其他数据类型不同的是，类除了数据，还可以有对数据进行操作的函数，分别称为类的数据成员和成员函数，而且不能在类的声明中对数据成员使用表达式进行初始化。

（1）声明类

C + + 中声明类的一般形式为：

```
class 类名{
  private：
      私有数据和函数
  public：
      公有数据和函数
  protected：
      保护数据和函数
};
```

类的声明以关键字 class 开始，其后跟类名，类所声明的内容用花括号括起来，称为类体，右花括号后的分号作为类的声明语句的结束标志。

类中定义的数据和函数称为这个类的成员，无论是数据成员还是成员函数，均具有一个访问权限，分别是私有、公有和保护，通过前加关键字 private、public 和 protected 来定义，如无关键字，则所有成员默认声明为 private 权限。

（2）定义成员函数

成员函数在声明后还必须在程序中通过定义来实现对数据成员的操作。

定义成员函数的一般形式为：

返回类型 类名：：成员函数名（形参列表）

```
{
 成员函数的函数体          //内部实现
}
```

其中“：：”是作用域运算符，类名是指成员函数所属类的名字，用于表示其后的成员函数属于这个特定的类。其中返回类型是指这个成员函数返回值的类型。

（3）数据成员的赋值和初始化

不能在类体内或类体外面给数据成员赋值，当然在类体外面就更不允许了。数据成员的具体值是用来描述对象的属性的。只有产生了一个具体的对象。这些数据值才有意义。如果在产生对象时就使对象的数据成员具有指定值，则称为对象的初始化。赋值和初始化时两个不同的概念。赋初值是在有了对象之后，对象调用自己的成员函数实现赋值操作；初始化是使用同名的构造函数实现的。

知识点三　使用类的对象

（1）有关类的一些规律如下：类的成员函数可以直接使用自己类的私有成员（数据成员和成员函数）；类外面的函数不能直接访问类的私有成员，而只能通过类的对象使用该类的公有成员函数；对象 A 和 B 的成员函数的代码一样，两个对象的区别是属性的取值。

（2）定义类对象指针的语法：

类名* 对象指针名；对象指针名=对象的地址；

也可以直接进行初始化。即：类名 * 对象指针名=对象的地址；

类对象的指针可以通过"->"运算符访问对象的成员，即：对象指针名->对象成员名

知识点四　数据封装

（1）面向对象的程序设计是通过为数据和代码建立分块的内存区域，以便提供对程序进行模块化的一种程序设计方法，这些模块可以被用作样板，在需要时再建立副本。由此，对象是计算机内存中的一块区域，通过将内存分块，每个模块在功能上保持相对独立。

（2）面向对象是消息处理机制，对象之间只能通过成员函数调用实现相互通信。当对象的一个函数被调用时，对象执行其内部的代码来响应这个调用，使对象呈现出一定的行为，对象被视为能作出动作的实体，对象使用这些动作完成相互之间的作用。

（3）C++通过类实现数据封装，即通过指定各成员的访问权限来实现。一般情况下将数据说明为私有的，以便隐藏数据；而将部分成员函数说明为公有的，用于提供外界和这个类的对象相互作用的接口（界面），从而使其他函数也可以访问和处理该类的对象。

（4）公用的成员函数是外界所能观察到（访问到）的对象界面，它们所表达的功能构成对象的功能，使同一个对象的功能能够在不同的软件系统中保持不变，这样当数据结构发生变化时，只需要修改少量的类的成员函数的实现代码，就可以保证对象的功能不变。只要对象的功能保持不变，则公有的成员函数所形成的接口就不会发生变化，这样，对象内部实现所作的修改就不会影响使用该对象的软件系统，这就是数据封装的益处。

模块二　构造函数

知识点一　默认构造函数

当没有为一个类定义任何构造函数的情况下，C + + 编译器会自动建立一个不带参数的构造函数，以 Point 类为例，默认构造函数的形式为：Point：：Point（）{} 即它的函数体是空的。一旦程序定义了自己的构造函数，系统就不再提供默认构造函数，这时如果没有再定义一个无参数的构造函数，但又声明了一个没有初始化的对象（如 Point A；），则因系统已不再提供默认构造函数而造成编译错误。同理，如果程序中定义了有参数的构造函数，又存在需要先建立对象数组后进行赋值操作的情况，则必须为它定义一个无参数的构造函数。

知识点二　定义构造函数

（1）构造函数的声明与定义

假设数据成员为 x1、x2、...xn，则有以下两种形式：

类名：：类名（形参 1，形参 2,...形参 n）：x1（形参 1），x2（形参 2）,...xn（形参 n）{}

```
类名：：类名（形参 1，形参 2，...形参 n）
{
   x1 = 形参 1；
   x2 = 形参 2；
   ……
   xn=形参 n；
}
```

构造函数的参数在排列时无顺序要求，只要保证相互对应即可，可以使用默认参数或重载。在程序中说明一个对象时，程序自动调用构造函数来初始化该对象。

（2）自动调用构造函数

程序员不能在程序中显式地调用构造函数，构造函数是自动调用的。可以设

计多个构造函数，编译系统根据对象产生的方法自动调用相应的构造函数，构造函数将在产生对象的同时初始化对象。

知识点三　构造函数和运算符 new

运算符 new 用于建立生存期可控的动态对象，new 返回这个对象的指针，使用运算符 new 建立的动态对象只能用运算符 delete 删除，以便释放所占空间。

知识点四　构造函数的默认参数

如果程序定义了有参数构造函数，又想使用无参数形式的构造函数，解决的方法是将相应的构造函数全部使用默认参数设计。

知识点五　复制构造函数

复制构造函数的作用，就是通过拷贝方式使用一个类已有的对象来建立该类的一个新对象，所以又直译为拷贝构造函数。通常情况下，由编译器建立一个默认复制构造函数，其原形为：类名：：类名（const 类名&），或不加 const 修饰如构造函数一样，复制构造函数也可自定义，如果自定义了，则编译器只调用自定义的复制构造函数，而不再调用默认的复制构造函数。

模块三　析构函数

知识点一　定义析构函数

为了与构造函数区分，在析构函数前加一个“~”号，并且在定义析构函数时也不能指定返回类型，即使是 void 类型也不可以，同时析构函数也不能指定参数，但可以显式的说明参数为 void，如 A：：~A（void），从函数重载角度分析，一个类只能定义一个析构函数，且不能指明参数，以便编译系统自动调用。析构函数在对象的生存期结束时被自动调用，然后对象占用的内存被回收。全局对象和静态对象的析构函数在程序运行结束之前调用。类的对象数组的每个元素调用一次析构函数，全局对象数组的析构函数在程序结束之前被调用。

知识点二　析构函数和运算符 delete

运算符 delete 与析构函数一起工作，当使用运算符 delete 删除一个动态对象时，它首先为这个对象调用析构函数，然后再释放这个动态对象占用的内存，

这和使用运算符 new 建立动态对象的过程刚好相反。

知识点三　默认析构函数

如果在定义类时没有定义析构函数，则编译器将自动为类产生一个函数体为空的默认析构函数，以 Point 类为例，默认析构函数形式如下：
Point：：~Point（ ）{ }

模块四　调用复制构造函数的综合实例

知识点一　调用复制构造函数的三种情况的小结

1、当用一个类的对象去初始化另一个对象时，需要调用复制构造函数。
2、如果函数的形参是类的对象，调用函数时，进行形参与实参的结合时，需要调用复制构造函数
3、如果函数的返回值是对象，当函数调用完成返回时，需要调用复制构造函数，产生临时对象，并在执行完返回值赋值语句后，析构临时对象和对象。
4、程序输出结果证实了调用构造函数和析构函数是按相反的顺序执行的，如果调用的函数程序里也产生了对象，则在函数程序里也遵循这一规律。

模块五　成员函数重载及默认参数

知识点一　成员函数重载及默认参数

成员函数可重载或使用默认参数。为了提高可读性，一般不在声明类时声明对象。

模块六　this 指针

知识点一　this 指针的概念和作用

C + + 规定，当一个成员函数被调用时，系统将自动向它传递一个隐含的参数，该参数是一个指向调用该函数的对象的指针，名为 this 指针，从而使成员函数知道该对哪个对象进行操作。this 指针是 C + + 实现封装的一种机制，它

将对象和该对象调用的成员函数连接在一起，从而在外部看来，每个对象都拥有自己的成员函数。

知识点二　this 指针的实际形式

例如当执行 A.Setxy（25，55）时，成员函数 Setxy（int a，int b）实际上是如下形式：

```
void Point：：Setxy（int a，int b，（Point *　）this）
{　　this->x = a;
　　this->y = b;
}　　　　//此时成员函数的 this 指针指向对象 A。
```

但是，一般情况下，即使在定义 Setxy 函数时使用指针，也不要给出隐含参（Point *）this，写成下面的形式即可：

```
void Point：：Setxy（int a，int b）
{　　this->x = a;
　　this->y = b;
}
```

除非特殊需要，一般情况下都省略掉符号“this->”，而让系统进行默认设置

模块七　一个类的对象作为另一个类的成员

知识点一　一个类的对象作为另一个类的成员

因为类本身就是一种新的数据类型，所以一个类的对象可以作为另一个类的成员。例如有 A、B 两个类，可以通过在 B 类里定义 A 的对象作为 B 的数据成员，或者定义一个返回类型为 A 的函数作为 B 的成员函数。

模块八　类和对象的性质

知识点一　对象的性质

（1）同一类的对象之间可以相互赋值。
（2）可以使用对象数组。
（3）可以使用指向对象的指针，使用取地址运算符&将一个对象的地址置于该指针中。

（4）对象可以用作函数参数。C++推荐使用对象的引用作为参数传递。为了避免被调用函数修改原来对象的数据成员，可以使用 const 修饰符。
（5）对象作为函数参数时，可以使用对象、对象引用和对象指针三种方式。
（6）一个对象可以作为另一个类的成员。

知识点二　类的性质

（1）使用类的权限
1、类本身的成员函数可以使用类的所有成员（私有和公有成员）。
2、类的对象只能访问公有成员函数。
3、其他函数不能使用类的私有成员，也不能使用公有成员函数，它们只能通过定义类的对象为自己的数据成员，然后通过类的对象使用类的公有成员函数。
4、虽然一个类可以包含另外一个类的对象，但这个类也只能通过被包含的类的对象使用那个类的成员函数，通过成员函数使用数据成员。
（2）不完全的类的声明
类不是内存中的物理实体，只有当使用类产生对象时，才进行内存分配，这种对象建立的过程称为实例化。不完全声明的类不能实例化，否则会编译出错；不完全声明仅用于类和结构，企图存取没有完全声明的类成员，也会引起编译错误。
（3）空类
尽管类的目的是封装代码和数据，它也可以不包括任何声明，如：class Empty{ }; 这种空类没有任何行为，但可以产生空类对象。
（4）类作用域
声明类时所使用的一对花括号形成类作用域，在类作用域中声明的标识符只在类中可见。如果某成员函数的实现是在类定义之外给出的，则类作用域也包含该成员函数的作用域，因此，当在该成员函数内使用一个标识符时，编译器会先在类定义域中寻找。

模块九　面向对象的标记图

知识点一　类和对象的 UML 标记图

（1）类和对象的标记符号类似，它们都只能表示静态特征。
（2）在 UML 语言中，类使用短式和长式两种方式表示。短式仅用 1 个含有类名的长方框表示，长式使用 3 个方框表示，最上面是类的名称，中间框填入属性（C++中称为数据成员），最下面填入成员函数（操作），属性和操作可以

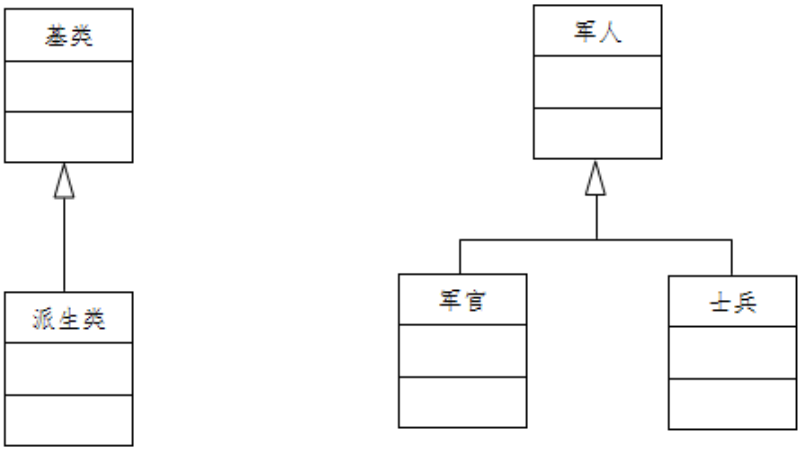
根据需要进行细化。

（3）对象的表示有 3 种方式，最简单的是只填写对象名称；完整的方式是在对象名和类名 ,并用下划线将它们标注出来 ;当还没有决定这个对象的名称时，可以不给出对象名，但不能省去冒号。

知识点二　对象的结构与连接

（1）分类关系及其表示

C＋＋中的分类结构是指继承（基类/派生类）结构，UML 使用一个空三角形表示继承关系，三角形指向基类。



（2）对象组成关系及其表示

组成关系说明的是整体与部分的关系，C＋＋中最简单的是包含关系，如线段由两个点组成。C＋＋中的聚合有两种实现方式，一种是独立的定义，可以属于多个整体对象，并有不同的生存期，例如一个法律顾问可以属于几个单位，这种所属关系是可以动态变化的，称为聚集，用空心菱形表示，另一种方式是用一个类的对象作为一种广义的数据类型来定义整体对象的一个属性 ,构成一个嵌套对象，这种情况下，这个类的对象只能隶属于唯一的整体对象并与它同生同灭，称这种情况为组合，使用实心菱形表示。

（3）实例连接及其表示

实例连接反映对象之间的静态关系，例如车和驾驶员的关系，实例连接有一对一、一对多和多对多 3 种连接方式，用一条实线表示连接关系。简单的实例连接是对象实例之间的一种二元关系。

（4）消息连接及其表示

消息连接描述对象之间的动态关系，即若一个对象在执行自己的操作时，需要通过消息请求另一个对象为它完成某种服务，则说第 1 个对象与第 2 个对象之间存在着消息连接。消息连接是有方向的，使用一条带箭头的实线表示，从消息的发送者指向消息的接收者。

知识点三　对象、类和消息

（1）对象的属性是指描述对象的数据成员。对象的行为是定义在对象属性上的一组操作的集合，操作（成员函数）是响应消息而完成的算法，表示对象内部实现的细节，对象的操作集合体现了对象的行为能力。对象的属性和行为是对象定义的组成要素，分别代表了对象的静态和动态特征，无论对象是简单的或是复杂的，对象一般具有以下特征：有一个状态，由与其相关联的属性值集合所表征；有唯一标识名，可以区别于其他对象；有一组操作方法，每个操作决定对象的一种行为；对象的状态只能被自己的行为所改变；对象的操作包括自身操作（施加于自身）和施加于其他对象的操作；对象之间以消息传递的方式进行通信；一个对象的成员仍可以是一个对象。其中，前三条是对象的基本特征，后四条属于特征的进一步定义。

（2）消息是向对象发出的服务请求，它是面向对象系统中实现对象间的通信和请求任务的操作。消息传递是系统构成的基本元素，是程序运行的基本处理活动。一个对象所能接受的消息及其所带的参数，构成该对象的外部接口。对象接收它能识别的消息，并按照自己的方式来解释和执行。一个对象可以同时向多个对象发送消息，也可以接收多个对象发来的消息。消息值反映发送者的请求，由于消息的识别和解释取决于接收者，因而同样的消息在不同对象中可解释成不同的行为。

（3）对象传送的消息一般由 3 部分组成：接收对象名、调用操作名和必要的参数。每个消息在类描述中用一个相应的方法给出，即使用成员函数定义操作。向对象发送一个消息，就是引用一个方法的过程，即实施对象的各种操作就是访问一个或多个在类中定义的方法。

（4）消息协议是一个对象对外提供服务的规定格式说明，外界对象能够并且只能向该对象发送协议中所提供的消息，请求该对象服务。具体实现是将消息分为私有和公有消息，前者只供内部使用，后者是对外的接口，协议则是一个对象所能接受的所有公有消息的集合。

模块十　面向对象编程的文件规范

知识点一 编译指令

C++的源程序可包含各种编译指令，以指示编译器对源代码进行编译之前先对其进行预处理。所以的编译指令都以#开始，每条指令单独占用一行，同一行不能有其他编译指令和 C++语句（注释例外）。编译指令不是 C++的一部分，但扩展了 C++编程环境的使用范围，从而改善程序的组织和管理。

（1）嵌入指令

嵌入指令#include 指示编译器将一个源文件嵌入到该指令所在的位置。尖括号或双引号中的文件名可含有路径信息。如：#include <\user\prog.h> 注意：由于编译指令不是 C++的一部分，因此在这里表示反斜杠时只需使用一个反斜

杠。如果在 C + + 程序中表示上述文件名 ,则必须使用双反斜杠 ,如 :char fname [ ]= “\\user\\prog.h” ;

（2）宏定义

＃define 指令定义一个标识符及串，在源程序中每次遇到该标识符时，编译器将自动用后面的串代替它。该标识符称为宏名，替换过程称为宏替换。宏定义的一般形式为：＃define 宏名 替换正文。注意：宏定义由新行结束，而不以分号结束，分号视为替换正文的一部分。

（3）条件编译指令

条件编译指令是指＃if、＃else、＃elif 和＃endif，它们构成了类似 C + + 的 if 选择结构，其中＃endif 表示一条指令结束。

（4）defined 操作符

关键字 defined 不是指令，而是一个预处理操作符，用于判定一个标识符是否已经被＃defined 定义 ,如果标识符 identifier 已被＃defined 定义 ,则 defined（identifier）为真，否则为假。

知识点二 在头文件中使用条件编译

在多文件设计中，由于文件包含指令可以嵌套使用，可能会出现不同文件包含了同一个头文件 ,这样会引起变量及类的重复定义。为了避免重复编译类的同一个头文件，可对这类头文件使用条件编译。

真题演练

1、（201001 单选题）C++允许在结构中定义函数，这些函数称为（ ）

A、静态函数
B、构造函数
C、析构函数
D、成员函数

答案：D
解析：类中定义的数据和函数称为这个类的成员(数据成员和成员函数)。

2、（201110 单选题）构造函数用于（ ）。

A、定义对象
B、初始化对象
C、:清除对象
D、普通计算

答案：B
解析：C++有称为构造函数的特殊成员函数，它可自动进行对象的初始化。

3、（201610 单选题）用运算符 delete 删除一个动态对象时（ ）。

A、首先为该动态对象调用构造函数，再释放其占用的内存
B、首先释放该动态对象占用的内存，再为其调用构造函数
C、首先为该动态对象调用析构函数，再释放其占用的内存
D、首先释放该动态对象占用的内存，再为其调用析构函数

答案：C
解析：运算符 delete 与析构函数一起工作。当使用运算符 delete 删除一个动态对象时，它首先为这个动态对象调用析构函数，然后再释放这个动态对象占用的内存，这和使用 new 建立动态对象的过程正好相反。因此选择 C 选项。

4、（201101 单选题）this 指针存在的目的是( )

A、保证基类公有成员在子类中可以被访问
B、保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码
C、保证基类保护成员在子类中可以被访问
D、保证基类私有成员在子类中可以被访问

答案：B
解析：使用 this 指针，保证了每个对象可以拥有自己的数据成员，但处理这些数据成员的代码可以被所有的对象共享。

5、在编译指令中，宏定义使用哪个命令（ ）。

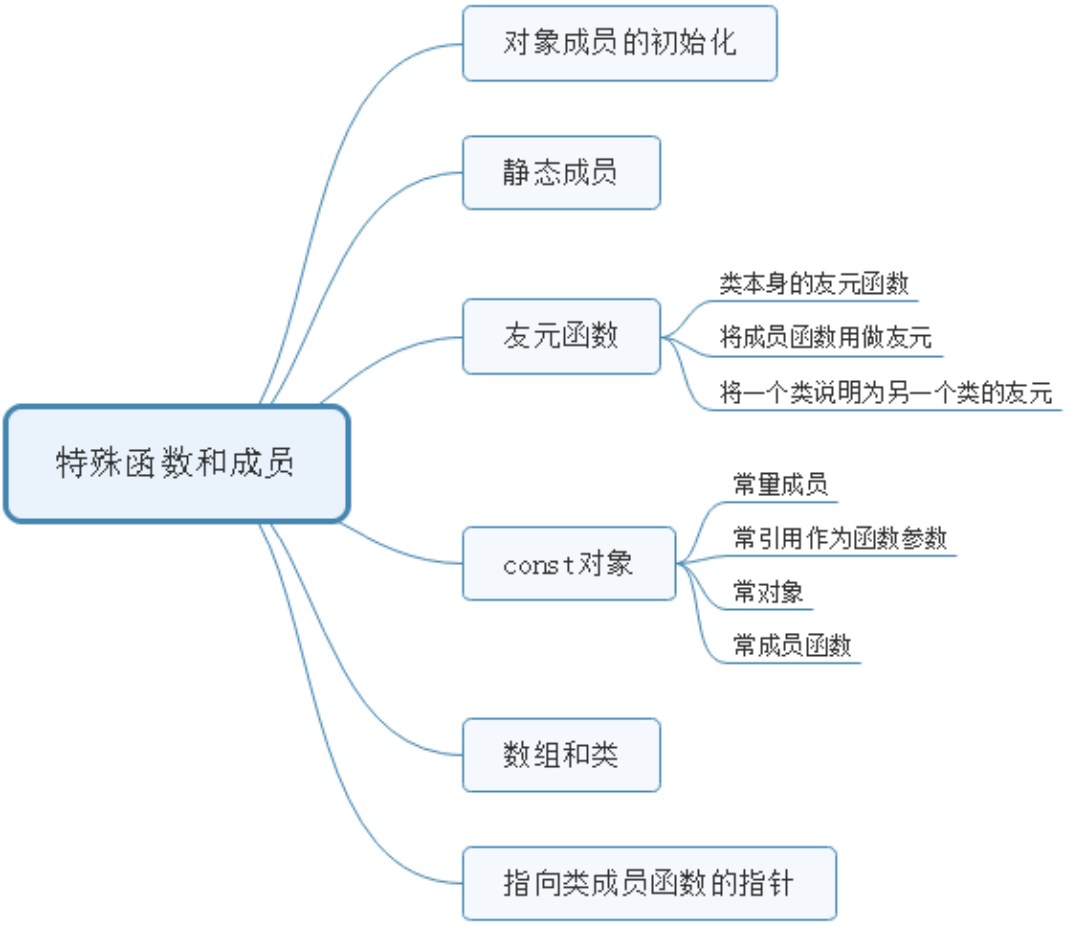
A、#if
B、#include
C、#define
D、#error

答案：C
解析：#define 指令用以进行宏定义，其一般形式如下：#define 宏名 替换正文。因此选择 C 选项。

本章的目的是建立类和对象的概念，要求重点掌握在 C++中定义类、建立和使用对象的基本方法。重点：类的定义方法、构造函数、复制构造函数、析构函数、成员函数重载及默认参数。难点：一个类作为另一个类的成员、this 指针、UML 图和面向对象编程的文件规范。

第五章 特殊函数和成员

思维导图



模块一　对象成员的初始化

知识点一　对象成员的概念

可以在一个类中说明具有某个类的类型的数据成员，这些成员称为对象成员。在类 A 中说明对象成员的一般形式为：

```
class A{
    类名 1  成员名 1；
    类名 2  成员名 2；
    ……
    类名 n  成员名 n；
}；
```

说明对象成员是在其所属类名之后给出对象成员的名字。

知识点二　含对象成员的类的构造函数格式

为初始化这些对象成员，A 类的构造函数需要分别调用这些对象成员所属类的构造函数。含有对象成员的类 A 的构造函数的定义形式为：

A：：A（参数表 0）：成员 1（参数表 1），成员 2（参数表 2），…，成员 n（参数表 n）｛…… //其他操作｝

冒号“：”后由逗号隔开的项组成成员初始化列表，其中的参数表给出了为调用相应成员所在类的构造函数时应提供的参数，参数列表中的参数都来自于“参数表 0”，可以使用任意复杂的表达式，其中可以有函数调用。

对象成员构造函数的调用顺序取决于这些对象成员在类 A 中说明的顺序，与它们在成员初始化列表中的顺序无关。当建立 A 类的对象时，先调用对象成员的构造函数，初始化对象成员，然后才执行 A 类的构造函数，初始化 A 类中的其他成员。析构函数的调用顺序与构造函数正好相反。

模块二　静态成员

知识点一　静态成员

（1）简单成员函数是指声明中不含 const、volatile、static 关键字。如果类的数据成员或成员函数使用关键字 static 修饰，这样的成员称为静态数据成员或静态成员函数，统称为静态成员。

（2）静态数据成员只能说明一次，如果在类中仅对其进行声明，则必须在文

件作用域的某个地方进行定义，在进行初始化时，必须进行成员名限定，如："int Test::x=25;"，也可直接在构造函数中使用类成员限定符对其进行初始化，如："Test(int a,int b) {Test::x=a;n=b;}"

（3）注意：由于 static 不是函数类型中的一部分，所以在类声明之外定义静态成员函数时，不使用 static，在类中定义的静态成员函数是内联的。

（4）静态成员函数与一般成员函数的不同

- 可以不指向某个具体的对象，只与类名连用；
- 在没有建立对象之前，静态成员就已存在；
- 静态成员是类的成员，不是对象的成员；
- 静态成员为该类的所有对象共享，它们被存储于一个公用内存中；
- 没有 this 指针，只能通过对象名或指向对象的指针访问类的数据成员；
- 静态成员函数不能被说明为虚函数；
- 静态成员函数不能直接访问非静态函数；

模块三　友元函数

知识点一　类外的独立函数作为类的友元函数

虽然在类中说明友元函数，但它并不是类的成员函数，所以可以在类外面像普通函数那样定义这个函数、这个独立函数其实就是一个普通的函数，仅有的不同点是：它在类中被说明为类的友元函数，可以访问该类所有对象的私有成员。

知识点二　将成员函数用做友元

一个类的成员函数（包括构造函数和析构函数）可以通过使用 friend 说明为另一个类的友元函数。例如将类 One 的成员函数 func 说明为类 Two 的友元，可在类 Two 中用语句："friend void One::func(Two&);" 实现，其中因 func 属于类 One，所以要用限定符说明出处。这样 One 的对象就可以通过友元函数 func(Two&)访问类 Two 的所有成员，因为是访问类 Two，所以应使用类 Two 对象的引用作为传递参数。

知识点三　将一个类说明为另一个类的友元

可以将一个类 One 说明为另一个类 Two 的友元，这时，整个类 One 的成员函数均是类 Two 的友元函数，声明语句简化为："friend class 类名；"。

例：类 One 作为类 Two 的友元的实例

```
#include <iostream>
using namespace std;
class Two{
    int y;
```

```
public:
    friend class One;   //声明类 One 为类 Two 的友元
};
class One{                          //类 One 的成员函数均是类 Two 的友元函数
    int x;
public:
    One(int  a,Two&r,int  b){x=a;r.y=b;}   //语句①，利用类 One 的构造函数给本类及类 Two 的对象赋值
    Void  Display(Two&);   //声明类 Ont 的成员函数，它能访问类 Two 的成员
};
void One::Display(Two&r){cout<<x<<"  "<<r.y<<endl;}   //语句②
void main( ){
Two Obj2;   //语句③
One Obj1(23,Obj2,55);   //语句④
Obj1.Display(Obj2);      //语句⑤
}
```

上面程序的主要执行步骤为：

第一步：执行语句③，调用类 Two 的默认构造函数 Two::Two( ){ };生成类 Two 的对象 Obj2，其值未定；

第二步：执行语句④，调用语句①，将实参 23、Obj2、55 赋给形参 a、r、b，其中 r 为类 Two 的对象 Obj2 的引用，执行函数体内语句，将 a、b 的值赋给 x 和 y，x、y 的值变为 23、55。其中因构造函数本身属于类 One 的成员，所以可以直接存取本类对象 Ojb1 的私有数据成员 x，而作为类 Two 的友元，其对类 Two 的对象 Obj2 的私有数据 y 的修改则只能通过对象名实现，相应语句为“r.y=b;”。

第三步：执行语句⑤，调用语句②，将实参 Obj2 的名字赋给形参 r（实际上传递的是地址），r 作为 Obj2 的引用，不产生临时对象，不需调用构造函数，执行函数体内语句，输出 Obj1 的私有数据成员 x 的值和 Obj2 的私有数据成员 y 的值，输出信息“23 55”。

第四步：先后执行类 One 和类 Two 的默认析构函数，析构对象 Obj1 和 Obj2。

模块四　const 对象

知识点一 常量成员

常量成员包括静态常数据成员、常数据成员和常引用，其中前者仍保留静态成员特征，需要在类外初始化，后两者则只能通过初始化列表来获得初值。

知识点二 常引用作为函数参数

使用引用作为函数参数，传送的是地址。但有时仅希望将参数的值提供给函数使用，并不允许函数改变对象的值，这时可以使用常引用作为参数。

知识点三 常对象

在对象名前使用 const 声明的对象就是常对象 ,常对象必须在声明的同时进行初始化，而且不能被更新，形式为：
类名 const 对象名（参数表）； //必须在声明的同时进行初始化

知识点四 常成员函数

声明常成员函数的格式：类型标识符 函数名（参数列表） const；
定义格式：类型标识符 类名::函数名（参数列表） const {……//函数体}
在类中用内联函数定义 const 函数 :类型标识符 函数名(参数列表) const {……//函数体}

模块五 数组和类

知识点一 数组和类

类的引入产生了除 C 语言可以声明的数组类型外的一系列新的数组类型，下面简单介绍类对象数组和类对象指针数组。
ANSI C 可以声明的数字类型都适用于 C++，但类的引入产生了一系列新的数组类型。

模块六 指向类的成员函数的指针

知识点一 指向类的成员函数的指针

（1）C++既包含指向类数据成员的指针，又包含指向成员函数的指针。它提供一种特殊的指针类型，指针指向类的成员，而不是指向该类的一个对象中该成员的一个实例，这种指针称为指向类成员的指针。不过，指向类数据成员的指针要求数据成员是公有的，用途不大。

（2）指向类的成员函数的指针

如类 A 的成员函数的参数类型列表为 list，返回类型为 type，则指向该函数的指针 pointer 的声明形式为“type (A::*pointer) (list);” ，此声明可读做：pointer 是一个指针，指向类 A 的成员函数，此成员函数参数类型列表为 list，返回值类型为 type。

如果类 A 的成员函数 fun 的原型和 pointer 指向的函数的原型一样，则语

句“pointer = A::fun;”将函数 fun 的地址（不是真实地址，而是在类 A 中所有对象的偏移）置给指针 pointer，即指针 pointer 指向函数 fun（好比数组的情况，数组名即是数组的首地址）。
在使用指向类成员函数的指针访问类的某个成员函数时，必须指定一个对象，通过对象名、引用或指向对象的指针调用该成员函数，其中前两者使用运算符“.*”，后者使用运算符“->”。

真题演练

1、（201410 单选题）在类中使用 static 关键字修饰的成员函数称为（ ）。

A、全局成员函数
B、公有成员函数
C、静态成员函数
D、非静态成员函数

答案：C
解析：如果类的数据成员或成员函数使用关键字 static 进行修饰，这样的成员称为静态数据成员或静态成员函数。

2、（201704 单选题）友元函数的主要作用是（ ）

A、提高程序的效率
B、加强类的封装性
C、实现数据的隐蔽性
D、增加成员函数的种类

答案：A
解析：友元说明可以出现于类的私有或公有部分。因为友元说明也必须出现于类中，所以应将友元看做类接口的一部分。使用它的主要目的是提高程序效率。因此 A 正确。

3、（201604 单选题）已知：func( )函数是一个类的常成员函数，它无返回值，下列表示中，是正确的是（ ）。

A、void func( ) const;
B、const void func( );
C、void const func( );

D、void func(const);

答案：A
解析：声明常成员函数的格式为：类型标识符 函数名（参数列表）const；因此选择 A 选项。

4、（201504 单选题）对于友元描述正确的是（ ）。

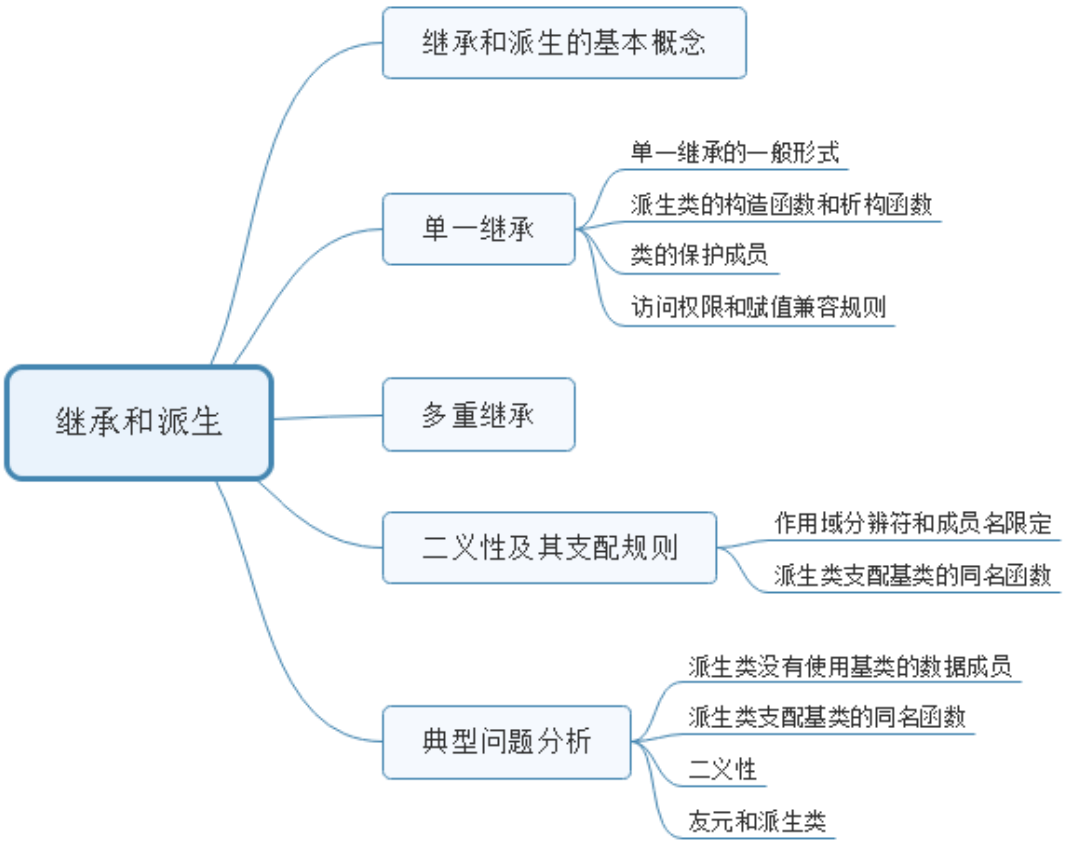
A、友元是本类的成员函数
B、友元不是本类的成员函数
C、友元不是函数
D、友元不能访问本类私有成员

答案：B
解析：一个类的成员函数包括构造函数和析构函数。不包括友元函数。

本章的目的是介绍类的一些特殊的函数。要求掌握它们的性质，并通过学习一个多文件编程范例，掌握使用面向对象思想解题的全过程。重点：静态成员、友元、const 对象、数组和类。难点：友元、转换函数、指向类成员函数的指针。

第六章 继承和派生

思维导图



模块一　继承和派生的概念

知识点一 继承和派生

（1）从一个或多个以前定义的类(基类)产生新类的过程称为派生，这个新类称为派生类。派生的新类同时也可以增加或者重新定义数据和操作，这就产生了类的层次性。

（2）类的继承是指派生类继承基类的数据成员和成员函数。继承常用来表示类属关系，不能将继承理解为构成关系。

（3）当从现有类中派生出新类时，派生类可以有如下几种变化：

- 增加新的成员（数据成员或成员函数）。
- 重新定义已有的成员函数。
- 改变基类成员的访问权限。

模块二　单一继承

知识点一 单一继承的一般形式

```
class 派生类名:访问控制 基类名{
    private:
        成员声明列表
    protected:
        成员声明列表
    public:
        成员声明列表
};
```

其中，访问控制是指如何控制基类成员在派生类中的访问属性，它是关键字 private、protected 和 public 中的一个，声明中的其余部分和类的声明类似。

知识点二 派生类的构造函数和析构函数

（1）定义派生类构造函数的一般定义形式：

派生类名::派生类名（参数表 0）:基类名（参数表）{……　//函数体}

冒号后“基类名（参数表）”称为成员初始化列表，参数表给出所调用的基类构造函数所需要的实参，实参的值可来自“参数表 0”，也可由表达式给出。

（2）派生类析构函数的一般定义形式：派生类名::~派生类名（）{……//函数体}构造函数和析构函数也都可在类体内直接定义为内联函数，这时的定义形式需把上述定义式中的“派生类名::”去掉。

知识点三 类的保护成员

类的保护成员是指在类声明中以关键字 protceted 声明的成员，保护成员具有私有成员和公有成员的双重角色：对派生类的成员函数而言，它是公有成员，可直接访问；而对其他函数而言，它是私有成员，不能直接访问，只能通过基类的对象访问。这样就解决了使用公有方式产生的派生类的成员函数虽然可直接访问基类中定义的或从另一个基类继承来的公有成员，但不能访问基类的私有成员的问题。

知识点四 访问权限和赋值兼容规则

（1）公有派生和赋值兼容规则

在公有派生情况下，可以通过定义派生类自己的成员函数来访问派生类对象继承来的公有和保护成员，但不能访问继承来的私有成员和不可访问成员（基类的基类的私有成员）。所以当希望类的某些成员能够被派生类访问，而又不能被其他的外界函数访问的时候，就应当把它们定义为保护的，而千万不能把它们定义为私有的，否则在派生类中它们就会是不可访问的。

所谓赋值兼容规则，就是在公有派生情况下，通过将基类的成员都定义为公有或保护的，从而使每一个派生类的对象都可作为基类的对象来使用的情况。比如，如果将基类的某个成员定义为私有的话，则派生类中继承的该成员就不能被派生类的成员函数访问，这样派生类的对象就不能作为基类的对象使用，也就不符合赋值兼容规则。

（2）“isa”和“has-a”的区别

（1）所谓“isa”，就是公有继承“就是一个”的含义，是指在公有继承的赋值兼容规则下，如果类 B 公有继承于类 A，在可以使用类 A 的对象的任何地方，则类 B 的对象同样也能使用，即每一个类 B 的对象“就是一个”类 A 的对象，但反之则不然，即如果需要一个类 B 的对象，则类 A 的对象就不行。

（2）所谓“has-a”，就是分层时“有一个”的含义，分层也称包含、嵌入或聚合，它是一种处理过程，通过让分层的类里包含被分层的类的对象作为其数据成员，以便把一个类建立在另一些类之上。例如一个 person 对象有一个名字、一个地址和两个电话号码，那么 person 类表示的就是一种“has-a”的关系。

（3）公有继承存取权限表

派生类一般都使用公有继承，对派生类而言，基类中的保护类型的成员介于私有和公有之间，派生类可以访问它，而类的对象、外部函数以及不属于本系类之外的类则不可访问它。

基类成员	派生类成员函数	基类和派生类对象	外部函数
private 成员	不可访问	不可访问	不可访问
protected 成员	protected	不可访问	不可访问
public 成员	public	可访问	可访问

（4） 私有派生

通过私有派生，基类的私有和不可访问成员在派生类中是不可访问的，而公有和保护成员这时就成了派生类的私有成员，派生类的对象不能访问继承的基类成员，必须定义公有的成员函数作为接口。

更重要的是，虽然派生类的成员函数可通过自定义的函数访问基类的成员，但将该派生类作为基类再继续派生时，这时即使使用公有派生，原基类公有成员在新的派生类中也将是不可访问的。

（5）保护派生

派生也可使用 protected 定义，这种派生使原来的权限都降一级使用，即 private 变为不可访问；protected 变为 private；public 变为 protected。因为限制了数据成员和成员函数的访问权限，所以用的也很少。

它与私有派生的区别主要在于下一级的派生中，如果降上例中 Rectangle 类改为如下的保护继承方式：

```
class Rectangle:protected Point {   //类体};
```

则 Test 类中就可以使用基类的 Show 函数，即下面的函数定义是正确的：

```
class Test:public Rectangle{      //公有继承，仍能访问基类的公有成员
public:
    Test(int a,int b,int h,int w):Rectangle(a,b,h,w){ }
    void Show( ){Point::Show( );Rectangle::Show( );}
};
```

模块三　多重继承

知识点一 多重继承

多重继承可视为单一继承的扩展，一个类从多个基类派生的一般形式为：

class 类名 1：访问控制 类名 2，访问控制 类名 3，…，访问控制 类名 n{　//类体}

其中，类名 1 继承了类名 2 到类名 n 的所有数据成员和成员函数，访问控制用于限制其派生类中的成员对基类的访问控制，规则和单一继承情况一样。

模块四　二义性及其支配规则

知识点一　作用域分辨和成员名限定

对基类成员的访问必须是无二义性的，如使用一个表达式的含义能解释为可以访问多个基类中的成员，则这种对基类成员的访问就是不确定的，称这种访问具有二义性。

作用域分辨操作的一般形式如下：类名：：标识符，“类名”可以是任一基类或派生类名，“类标识符”是该类中声明的任一成员名。

知识点二　派生类支配基类的同名函数

基类的成员和派生类新增的成员都具有作用域，基类在外层，派生类在内层。如果此时派生类定义了一个和基类成员函数同名的新成员函数（因参数不同属于重载，所以这里是指具有相同参数表的成员函数），派生类的新成员函数就覆盖了外层的同名成员函数，在这种情况下，直接使用成员名就只能访问派生类的成员函数，只有使用作用域分辨，才能访问基类的同名成员函数。

模块五　典型问题分析

知识点一　派生类没有使用基类的数据成员

使用派生时，爱犯的错误是忘记派生类已经继承了基类的数据成员，而又重新定义新的数据成员，例如从 Point 类派生 Rectangle 类，忘记后者继承了前者的一个点（x，y），而又重新定义了四个数据成员（x，y，H，W），其实，后者只需再定义两个数据成员 H 和 W 即可。

知识点二　派生类支配基类的同名函数

当基类和派生类具有同名成员函数时，往往不知对象调用哪个成员函数，其实，只需记住：对象一定先调用自己的同名成员函数，只有使用了作用域分辨运算符，才会调用直接基类的同名成员函数。

知识点三　二义性

程序设计过程中，一定要注意避免定义的二义性，为此可通过使用作用域分辨运算符“：：”和成员名限定来消除二义性。

知识点四　友元和派生类

友元声明与访问控制无关。友元声明在私有区域进行或在公有区域进行是没有太大区别的。虽然友元函数的声明可以置于任何部分，但通常置于能强调其功能的地方以使其直观。对友元函数声明的惟一限制是该函数必须出现在类声明内的某一部分。

友元关系是无等级的。友元可以访问任何一个类成员函数可以访问的对象，这比一个派生类可以访问的对象还多。友元关系不能继承。一个派生类的友元只能访问该派生类的直接基类的公有和保护成员，不能访问私有成员。当友元访问直接基类的静态保护成员时，只能使用对象名而不能使用成员名限定。

真题演练

1、（201604 单选题）关于函数模板，描述错误的是（ ）。

A、函数模板必须由程序员实例化为可执行的函数模板
B、函数模板的实例化由编译器实现
C、一个类定义中，只要有一个函数模板，则这个类是类模板
D、类模板的成员函数都是函数模板，类模板实例化后，成员函数也随之实例化

答案：C
解析：类模板声明的一般方法为：template<类模板参数>class 类名{//类体}；，因此类中有函数模板不能说明是类模板，因此 C 选项错误。

2、（201610 单选题）假设声明了以下的类模板，错误的调用语句是（ ）

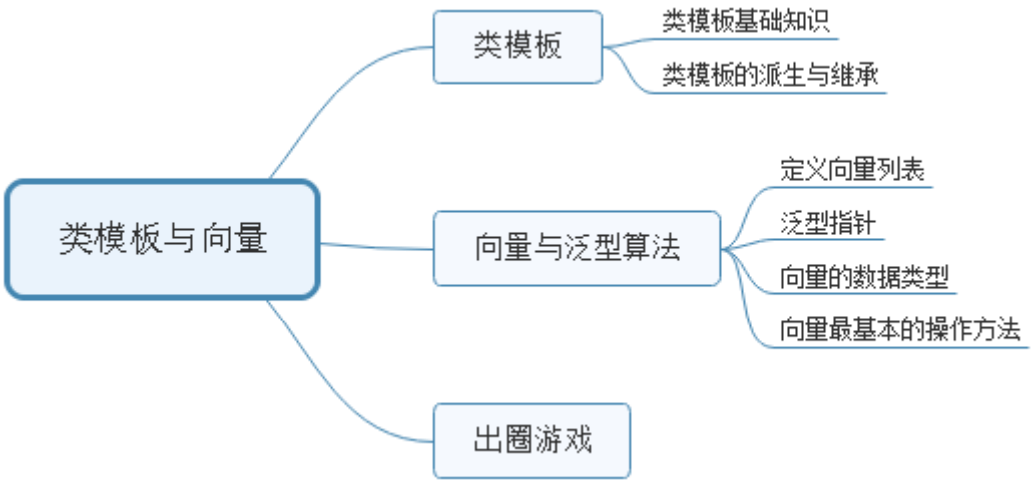
template <class T>
T max(T x，T y){return（x>y）? x :y;}
并定义了 int i；char c；
A、max（i，i）
B、max（c，c）
C、:max（（int）c，i）
D、:max（i，c）

答案：D
解析：类模板 max 声明了类型都为 T 的两个私有数据成员 x，y，并且 x，y 类型相同。因此 D 选项错误。

本章的目的是介绍 C++语言继承方面的语法特征，要求通过仔细研读实例，掌握派生和包含的设计方法。重点：派生类的构造函数和析构函数、访问权限和赋值兼容规则。难点：二义性及其支配规则。

第七章 继承和派生

思维导图



模块一　类模板

知识点一　类模板基础知识

（1）类模板的成分及语法

类模板声明的一般形式：template <类模板参数> class 类名 { //类体};

类模板参数中的 class 意为“任意内部类型或用户定义类型”，T 是结构或类。由此可见，类模板与函数模板的声明方法及参数格式是相同的，不同的是在类模板参数表之后还有类声明。在类中可以像使用其他参数类型(如 int、double 等) 那样使用模板参数。例如，可以把模板参数用作数据成员的类型、成员函数的类型或成员函数的参数类型等。

（2）类模板的对象

类模板也称参数化类，初始化类模板时，只要传给它指定的数据类型，编译器就会用指定类型替代模板参数产生相应的模板类。

类模板定义对象的一般格式：

类名 <模板实例化参数类型> 对象名; //默认或无参数构造函数

类名 <模板实例化参数类型> 对象名（构造函数实参列表）; //构造函数

（3）定义模板类的成员函数

在类体外面定义成员函数时，必须用 template 重写类模板说明，一般格式为：

template <模板参数>

返回类型 类名 <模板类型参数>::成员函数名（函数参数列表）{ //函数体}

式中<模板类型参数>是指 template 后的“ < >”内使用 class（或 typename）声明的类型参数，构造函数和析构函数没有返回类型。

模板实例化参数类型包括数据类型和值，因为编译器不能从构造函数参数列表推断出模板实例化参数类型，所以必须显式的给出对象的参数类型。

知识点二　类模板的派生和继承

类模板也可以继承，继承的方法和普通的类一样。声明模板继承之前，必须重新声明类模板。模板类的基类和派生类都可以是模板类或非模板类。

例 4：类模板 Line 继承非模板类 Point

```
#include <iostream>
using namespace std;
class Point{
        int x,y;
public:
        Point(int a,int b){x=a;y=b;}   //类 Point 的构造函数
```

```
        void display(){cout<<x<<","<<y<<endl;}   //类Point的公有成员函数
    };
    template <typename T>   //声明继承之前，需重新声明类模板
    class Line:public Point{   //模板类 Line 公有继承非模板类 Point
        T x2,y2;
    public:
        Line(int a,int b,T c,T d):Point(a,b){x2=c;y2=d;}    //类 Line 的构造函数
        void display(){Point::display();cout<<x2<<","<<y2<<endl;}
    };
    void main(){
        Point a(3,8);
        a.display();   //输出 3，8
        Line<int> ab(4,5,6,7);    //线段 ab 两个点的坐标均是整数
        ab.display();   //输出 4，5    6，7
        Line<double>  ad(4,5,6.5,7.8);    //线段 ad 一个点的坐标是整数，另一个是实数
        ad.display();   //输出 4，5    6.5，7.8
    }
    例 5：模板类 Point 公有派生模板类 Line
    #include <iostream>
    using namespace std;
    template <typename T>
    class Point{
        T x,y;
    public:
        Point(T a,T b){x=a;y=b;}   //模板类 Point 的构造函数
        void display(){cout<<x<<","<<y<<endl;}   //模板类Point的公有成员函数
    };
    template <typename T>   //声明继承之前，需重新声明类模板
    class Line:public Point<T>{   //模板类 Line 公有继承模板类 Point
        T x2,y2;
    public:
        Line(T a,T b,T c,T d):Point<T>(a,b){x2=c;y2=d;}   //模板类 Line 的构造函数
        void display(){Point<T>::display();cout<<x2<<","<<y2<<endl;}
```

```
};
void main(){
    Point <double> a(3.5,8.8);
    a.display();   //输出 3.5，8.8
    Line<int> ab(4,5,6,7);    //全部使用整数
    ab.display();   //输出 4，5    6，7
    Line<double> ad(4.5,5.5,6.5,7.5);    //全部使用实数
    ad.display();   //输出 4.5，5.5    6.5，7.5
}
```

在本例中，因为派生类是模板类，基类 Point 也是模板类，所以属于基类的坐标点和属于派生类的坐标点的数据类型相同。

模块二　向量与泛型算法

知识点一　向量与泛型算法

向量是一维数组的类版本，与数组类似，其中的元素项是连续存储的，不同的是：第一，在数组生存期内，数组的大小是不变的，而向量中存储元素的多少可以在运行中根据需要动态的增长或缩小;第二，向量是类模板，具有成员函数，例如可以使用向量名.size（ ）的方法动态地获得 vector 对象当前存储元素地数量。

知识点二　定义向量列表

（1）向量的构造函数

向量（vector）类模板定义在头文件 vector 中，它提供 4 种构造函数，用来定义由各元素组成的列表。

如果用 length 表示长度，用 type 表示数据类型，用 name 表示对象名，则有：

vector <type> name; //定义元素类型为 type 的向量空表

vector <type> name（length）; //定义具有 length 个类型为 type 的元素的向量，元素均初始化为 0

vector <type> name（length,a）; //定义具有 length 个类型为 type 的元素的向量，元素均初始化为 a

vector <type> name1（name）; //用已定义的向量 name1 构造向量 name

空表没有元素，它使用成员函数获取元素。

（2）向量赋值的典型方法：

vector <char> A; //定义空的 char 向量 A

vector <int> B(20); //定义含 20 个值均为 0 的 int 型元素的向量 B

vector <int> C(20,1); //定义含 20 个值均为 1 的 int 型元素的向量 C
vector <int> D(C); //用向量 C 初始化 D，使 D 与 C 一样
vector <char> E(20,’t’); //定义含 20 个值均为 t 的 char 型元素的向量 E
vector <int> F(2*2,5); //定义含 4 个值均为 5 的 int 型元素的向量 F
vector <int> G(F); //用向量 F 初始化 G，使 G 与 F 一样
D=F; //整体赋值，使 D 与 F 相同
G=C; //整体赋值，使 G 与 C 相同

式中通过赋值运算符“=”可使同类型的向量列表相互赋值，而不管它们的长度如何，向量可以改变赋值目标的大小，使它的元素数目与赋值源的元素数目相同。

对于上面的定义，通过使用语句“cout<<B.size()<<”,” <<C.size()<<”,” <<D.size()<<”,” <<E.size()<<”,” <<F.size()<<”,” <<G.size()<<endl;”，可得到关于每个向量长度的信息“20,20,4,20,4,20”.

知识点三　泛型指针

（1）与操作对象的数据类型相互独立的算法称为泛型算法，泛型算法通过一对泛型指针 begin 和 end 实现，元素存在范围是半开区间[begin,end)。

（2）iterator 声明正向泛型指针

在向量中，泛型指针是在底层指针的行为之上提供一层抽象化机制，是使用类实现的，取代程序原来的“指针直接操作方式”。

对向量的访问可以是双向的，即可正向，也可逆向。iterator 用于声明向量的正向泛型指针，它在 STL 里面是一种通用指针，如用 T 表示向量的参数化数据类型，则 iterator 在向量中的作用相当于 T *，其声明形式为：vector <type>::iterator 泛型指针名；例如“vector <char>::iterator p;”表示向量的数据类型为 char，指针名为 p，注意不能使用*p，这是因为 iterator 相当于 char *，如果用*p，则表示指针指向的元素值，这样来定义指针就是错误的。

（3）reverse_iterator 声明逆向泛型指针

逆向泛型指针的开始和结束标志是 rbegin 和 rend，加操作时向 rend 方向移动，减操作时向 rbegin 方向移动。逆向泛型指针的声明使用 reverse_iterator，声明形式为：vector <type>::reverse_iterator 泛型指针名；

（4）使用 typedef 声明或定义泛型指针标识符

使用 typedef 声明或定义泛型指针标识符后，可直接用该标识符定义泛型指针。例如语句“typedef vector <double>::iterator iterator;”定义标识符 iterator，在程序中就可直接使用 iterator 声明或定义泛型指针，如语句“iterator p;”声明了泛型指针 p。

知识点四　向量的数据类型

向量除了可使用基本数据类型，还可使用构造类型，只要符合构成法则即可。

知识点五　向量最基本的操作方法

（1）访问向量容量信息的方法

- size（）：返回当前向量中已经存放的对象个数；
- max_size（）：返回向量最多可以容纳的对象个数，此参数取决于硬件结构，不由用户指定；
- capacity（）：返回无需再次分配内存就能容纳的对象个数，其初始值为程序员最初申请的元素个数，即已申请的空间。实际操作时，当存放空间已满，又增加一个元素时，其值在原来的基础上自动翻倍，扩充空间以便存放更多的元素。这前三者关系：max_size()>=capacity()>=size()
- empty（）：当前向量为空时，返回 true 值，内容不空时，返回 0 值。

（2）访问向量中对象的方法

- front（）：返回向量中的第一个对象；
- back（）：返回向量中的最后一个对象；
- operator[]（size_type，n）：返回向量中下标为 n（第 n+1 个）的元素。

（3）在向量中插入对象的方法

- push_back（const T&）：向向量尾部插入一个对象。
- insert(iterator it,const T&):向 it 所指的向量位置前插入一个对象。
- insert(iterator it,size_type n,const T&X):向 it 所指向量位置前插入 n 个值为 x 的对象。

（4）在向量中删除对象的方法

- pop_back（const T&）:删除向量中最后一个对象。
- erase(iterator it):删除 it 所指向的容器对象
- clear():删除向量中的所有对象，empty()返回 true 值。

模块三　出圈游戏

知识点一　出圈游戏

```
#include <iostream>
#include <vector>
using namespace std;
class SeqList{  //定义类 SeqList
    char name[10];  //定义含 10 个元素的字符型数组 name 为 SeqList 类的私有数据成员
```

```
public:
    void DispName( ){cout<<name;}
    void SetName(char b[]){strcpy(name,b);}
    void Joseph(vector <SeqList>&);
};
//Joseph 函数开始
void SeqList::Joseph(vector <SeqList>&c)
{  //以 SeqList 类的引用作为向量 c 的数据类型，向量 c 作为函数 Joseph 的参数
    int m,star,i,j,k;  //定义间隔数 m、开始位置 star、i、j、k 为整型数
    cout<<"输入间隔数 m(m<=20)";
    cin>>m;  //间隔数
    while(m>20)  //间隔数大于 20 时，重新输入
    {
        cout<<"间隔太大，重新输入：";
        cin>>m;
    }
    cout<<"报数开始位置(不能大于"<<c.size()<<"):";
    cin>>star;
    while(star>c.size())
    {
        cout<<"开始位置大于人数，重新输入：";
        cin>>star;
    }
    cout<<"准备输入名字"<<endl;
    getchar();  //消除回车干扰
    //输入参加游戏人的名字
    char s[10];
    for(i=0;i<c.size();i++)
    {
```

确定游戏的间隔数

确定游戏开始开始位

输入参加游戏者的名字

```
        cout<<"第"<<i+1<<"个人的名字: ";
        gets(s);
        c[i].SetName(s);
    }
    i=star-2;  //为方便编程，从规定开始报数处再减 1 作为计数依据
    vector<SeqList>::iterator p;
    p=c.begin();
    int length=c.size();
```

要将出圈者删除，可使用 vector 类的成员函数 erase，但 erase 的参数是泛型指针，所以需要定义一个泛型指针 p 实现删除。

```
for(k=1;k<=length;k++)  //语句 1
    {
        j=0;  //报数
        while(j<m)
        {
            i++;
            if(i= =c.size())
                i=0;
            j++;
        }
        if (k==length) break;
        c[i].DispName();  //语句 2
        cout<<",";
        c.erase(p+i);  //语句 3
        --i;  //调整计数位置初始值
    }
    //break 语句跳转至此处，输出最后出列的编号
    c[i].DispName();
    cout<<endl;
}  //Joseph 函数结束
void main(){
```

因为有多少人参加游戏，便要进行多少圈，直至所有人都出圈；同时，因向量的 size 是变化的，每删除一个出圈者，size 的值也减少一个，所以不能以 size 作为总的循环次数，而是要以 c 的长度 length 即总人数作为总的循环次数。总的循环从 k＝1 开始，到 k＝length 结束，实现所有人均出圈（语句 1）。

以第一圈（k＝1）为例，j＝0 开始报数，为体现公平，不从第 1 个人的位置报数，而是通过随机输入的方法决定开始位置 star 参数，因为 j 是从 0 开始计数，到 j＝m-1 结束，所以第一次的报数位置是 star－1，为方便 while 中的循环设计，采取先减 1 的方法，即选择 star－2 作为循环变量 i 的初值。执行 i＋＋，i 的位置到达 star－1 处，当 i 的值等于当前圈循环人数 c.size 的时候，其值从 0 重新计数，否则，执行 j＋＋继续计数，直到 j 等于 m，此时 i 的值就是出圈人的位置；执行语句 2，输出出圈人信息；执行语句 3，删除出圈人的记录；执行－－i，调整计数位置初始值，第一圈循环结束。如此循环，直至所有人均出圈。

```
    int length=0;
    cout<<"请输入人数：";
    cin>>length;
    vector<SeqList>c(length);  //定义长 length、内容均为 0 的 SeqList 类的向量 c
    SeqList game;  //生成 SeqList 类的对象 game
    game.Joseph(c);  //对象 game 调用本类的成员函数 Joseph，参数为长 length、内容均为 0 的 SeqList 类的向量 c
}
```

真题演练

1、（201510 单选题）有关多态性说法不正确的是（ ）。
A、C++语言的多态性分为编译时的多态性和运行时的多态性
B、:编译时的多态性可通过函数重载实现
C、运行时的多态性可通过模板和虚函数实现
D、实现运行时多态性的机制称为动态多态性

答案：C
解析：静态联编所支持的多态性称为编译时的多态性，当调用重载函数时，编译器可以根据调用时所使用的实参在编译时就确定下来应调用哪个函数，动态联编所支持的多态性称为运行时的多态性，这由虚函数来支持。因此 C 选项不正确。

2、（201310 单选题）C++中要实现动态联编，调用虚函数时必须使用（ ）调用虚函数。

A、基类指针
B、类名
C、派生类指针
D、对象名

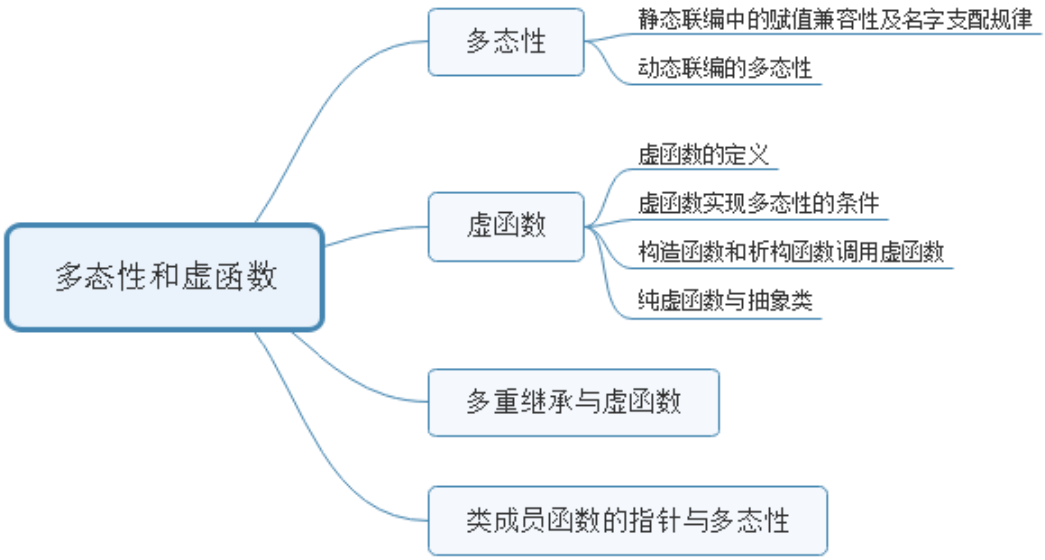
答案：A
解析：联编是指一个计算机程序自身彼此关联的过程。按照联编所进行的阶段不同，可分为两种不同的联编方法：静态联编和动态联编。编译程序在编译阶段并不能确切知道将要调用的函数，只有在程序执行时才能确定将要调用的函数，为此要确切知道该调用的函数，要求联编工作要在程序运行时进行，这种在程序运行时进行联编工作被称为动态联编，或称动态束定，又叫晚期联编。C++规定

动态联编是在虚函数的支持下实现的。C++中要实现动态联编，调用虚函数时必须使用基类指针。

本章的目的是介绍设计 C++类模板及使用向量容器的基本方法，要求掌握类模板的定义及其简单的派生方法、熟悉向量容器的定义和基本操作方法，了解泛型算法和 STL 库的概念。重点：类模板、定义向量列表和向量容器的基本操作。难点：泛型指针、在向量容器指定位置插入和删除对象的方法。

第八章 多态性和虚函数

思维导图



模块一　多态性

知识点一　多态性

静态联编所支持的多态性称为编译时的多态性，当调用重载函数时，编译器可以根据调用时所使用的实参在编译时就确定下来应调用哪个函数；动态联编所支持的多态性称为运行时的多态性，由虚函数实现。虚函数类似于重载函数，但与重载函数的实现策略不同，即对虚函数的调用使用动态联编。

知识点二　静态联编中的赋值兼容性及名字支配规律

在赋值兼容性规则下，声明的基类指针和引用只能指向基类，声明的派生类指针和引用也只能指向派生类，它们的原始类型决定它们只能调用各自的同名函数，除非派生类没有基类的同名函数，派生类的指针和引用才能根据继承原则调用基类的函数。通过内存分配原理，深入分析赋值兼容性规则可知，对象的内存地址空间中只包含数据成员，并不存储有关成员函数的信息，这些成员函数的地址翻译过程与其对象的内存地址无关，只与数据成员的类型有关，编译器只根据数据类型翻译成员函数的地址并判断调用的合法性，而类型则是事先决定的，这就是静态联编。

知识点三　动态联编的多态性

虚函数的地址翻译取决于对象的内存地址，虚函数的调用规则是：根据当前对象，优先调用对象本身的虚成员函数。派生类能继承基类的虚函数表，而且只要是和基类同名（参数相同）的成员函数，无论是否使用 virtual 声明，它们都自动成为虚函数。如果派生类没有改写继承基类的虚函数，则函数指针调用基类的虚函数；如果派生类改写了基类的虚函数，编译器将重新为派生类的虚函数建立地址，函数指针会调用改写过的虚函数。

模块二　虚函数

知识点一　多态性

静态联编所支持的多态性称为编译时的多态性，当调用重载函数时，编译器可以根据调用时所使用的实参在编译时就确定下来应调用哪个函数；动态联编所支持的多态性称为运行时的多态性，由虚函数实现。虚函数类似于重载函数，但与重载函数的实现策略不同，即对虚函数的调用使用动态联编。

知识点二　虚函数的定义

虚函数只能是类中的一个成员函数，但不能是静态成员，关键字 virtual 用于类中该函数的声明中。当在派生类中定义了一个同名的成员函数时，只要该成员函数的参数个数和相应类型及其返回类型与基类中同名的虚函数完全一样，则无论是否为该成员函数使用 virtual，它都将成为一个虚函数。

知识点三　虚函数实现多态性的条件

（1）关键字 virtual 指示 C++编译器对调用虚函数进行动态联编，这种多态性是程序运行到需要的语句处才动态确定的，所以称为运行时的多态性。不过，使用虚函数并不一定产生多态性，也不一定使用动态联编，例如在调用中对虚函数使用成员名限定，可以强制 C++对该函数的调用使用静态联编。

（2）产生运行时的多态性有三个前提：第一，类之间的继承关系满足赋值兼容性规则；第二，改写了同名虚函数；第三，根据赋值兼容性规则使用指针或引用。满足前两条并不一定产生动态联编，必须同时满足第三条才能保证实现动态联编。第三条又分两种情况：第一种是已经演示过的按赋值兼容性规则使用基类指针或引用访问虚函数；第二种是把指针或引用作为函数参数，即这个函数不一定是类的成员函数，可以是普通函数，而且可以重载。

知识点四　构造函数和析构函数调用虚函数

（1）在构造函数和析构函数中调用虚函数采用静态联编，即它们调用的虚函数是自己的类或基类中定义的函数，而不是任何在派生类中重新定义的虚函数。

（2）目前的 C++标准不支持虚构造函数。由于析构函数不允许有参数，所以一个类只能有一个虚析构函数。虚析构函数使用 virtual 说明，且只要基类的析构函数被说明为虚函数，则派生类的析构函数，无论是否使用 virtual 说明，都自动成为虚函数。

（3）运算符 new 和构造函数一起工作，运算符 delete 和析构函数一起工作，当使用 delete 删除一个对象时，delete 隐含着对析构函数的一次调用，如果析构函数为虚函数，则这个调用采用动态联编。

知识点五　纯虚函数与抽象类

（1）说明纯虚函数的一般形式为：virtual 函数类型 函数名（参数列表）=0；

（2）如果通过同一个基类派生一系列的类，则将这些类总称为类族。抽象类的这一特点保证了进入类族的每个类都具有提供纯虚函数所要求的行为，进而保证了围绕这个类族所建立起来的软件能正常运行。

（3）抽象类至少含有一个虚函数，而且至少有一个虚函数是纯虚函数，以便将它与空的虚函数区分开来，如下：

virtual void area( )=0; //纯虚函数

virtual void area( ) { }; //空的虚函数

（4）在成员函数内可以调用纯虚函数，因为没有为纯虚函数定义代码，所以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。

模块三　多重继承与虚函数

知识点一　多重继承与虚函数

多重继承可被视为多个单一继承的组合，因此，分析多重继承情况下的虚函数调用与分析单一继承有相似之处。

例、多重继承使用虚函数

```
#include <iostream>
using namespace std;
class A {
public:
    virtual void f() {cout<<"Call A"<<endl;}   //必须使用 virtual 声明
};
class B {
public:
    virtual void f() {cout<<"Call B"<<endl;}   //必须使用 virtual 声明
};
class C:public A,public B {
public:
    void f() {cout<<"Call C"<<endl;}   //可省略关键字 virtual
};
void main() {
    A *pa;
    B *pb;
    C *pc,c;
    pa=&c; pb=&c; pc=&c;
    pa->f();   //输出 Call c
    pb->f();   //输出 Call c
    pc->f();   //输出 Call c
}
```

上面的所有调用将实际调用 C 中的虚函数 f，即均输出“Call C”。

模块四　类成员函数的指针与多态性

知识点一　类成员函数的指针与多态性

在派生类中，当一个指向基类成员函数的指针指向一个虚函数，并且通过指向对象的基类指针访问这个虚函数时，仍发生多态性。

例、使用基类成员函数的指针产生多态性

```
#include <iostream>
using namespace std;
class base {
public:
    virtual void print() {cout<<"In base"<<endl;}   //虚函数
};
class derived:public base {
public:
    void print() {cout<<"In Derived"<<endl;}   //虚函数
};
void display(base *pb,void(base::*pf)()) { (pb->*pf)();}      //定义函数
display，使用 base 类的指针 pb 与指向 base 类成员函数的指针 pf 为参数，函数体内执行的语句意思为指针 pb 调用指针 pf 指向的成员函数
void main() {
    base b;
derived d;
    base *pb=&d;      //定义基类指针 pb，指针指向派生类对象 d
    void (base::*pf)();   //声明指向类 base 的成员函数的指针 pf
    pf=base::print;   //指针 pf 指向 base 类的具体成员函数 print
    display(pb,pf);   //输出“In Derived”，因 pb 指向派生类对象，优先调用本类的成员函数
     display(&b,base::print);   //输出“In Base”
    display(&d,base::print);   //输出“In Derived”
}
```

例中 display 函数有两个参数，第一个参数是基类指针，第二个参数是指向类成员函数的指针（见第 5.6 节），函数定义是使用基类指针调用指向成员函数的指针所指向的成员函数，至于是调用基类还是派生类的成员函数，取决于基类指针所指向的对象，基类指针指向的对象是基类对象，则调用基类的成员函数，基类指针指向的对象是派生类对象，则调用派生类的成员函数。也可直接将对象的地

址作为参数，而不是基类指针。

真题演练

1、（201310 单选题）C++中要实现动态联编，调用虚函数时必须使用（ ）调用虚函数。

A、基类指针
B、类名
C、派生类指针
D、对象名

答案：A
解析：联编是指一个计算机程序自身彼此关联的过程。按照联编所进行的阶段不同，可分为两种不同的联编方法：静态联编和动态联编。编译程序在编译阶段并不能确切知道将要调用的函数，只有在程序执行时才能确定将要调用的函数，为此要确切知道该调用的函数，要求联编工作要在程序运行时进行，这种在程序运行时进行联编工作被称为动态联编，或称动态束定，又叫晚期联编。C++规定动态联编是在虚函数的支持下实现的。C++中要实现动态联编，调用虚函数时必须使用基类指针。

2、（201310 单选题）下面声明纯虚函数语句正确的是（ ）。

A、void fun( ) =0;
B、virtual void fun( )=0;
C、virtual void fun( );
D、virtual void fun( ){ };

答案：B
解析：抽象类至少含有一个虚函数，而且至少有一个虚函数是纯虚函数，以便将它与空的虚函数区分开来。纯虚函数：virtual void area( )=0;

3、（201604 单选题）以下基类中的成员函数表示纯虚函数的是（ ）。

A、virtual void vf(int)
B、void vf(int)=0
C、virtual void vf( )=0
D、virtual void yf(int){ }

答案：C
解析：抽象类至少含有一个虚函数，而且至少有一个虚函数时纯虚函数，以便将它与空的虚函数区分开来。纯虚函数表示方法之一为:virtual void area( )=0，因此选择 C 选项。

4、（201504 单选题）下面函数原型声明中，声明了 fun 为虚函数的是（ ）。

A、void fun()=0
B、virtual void fun()=O
C、virtual void fun()
D、virtual void fun(){}

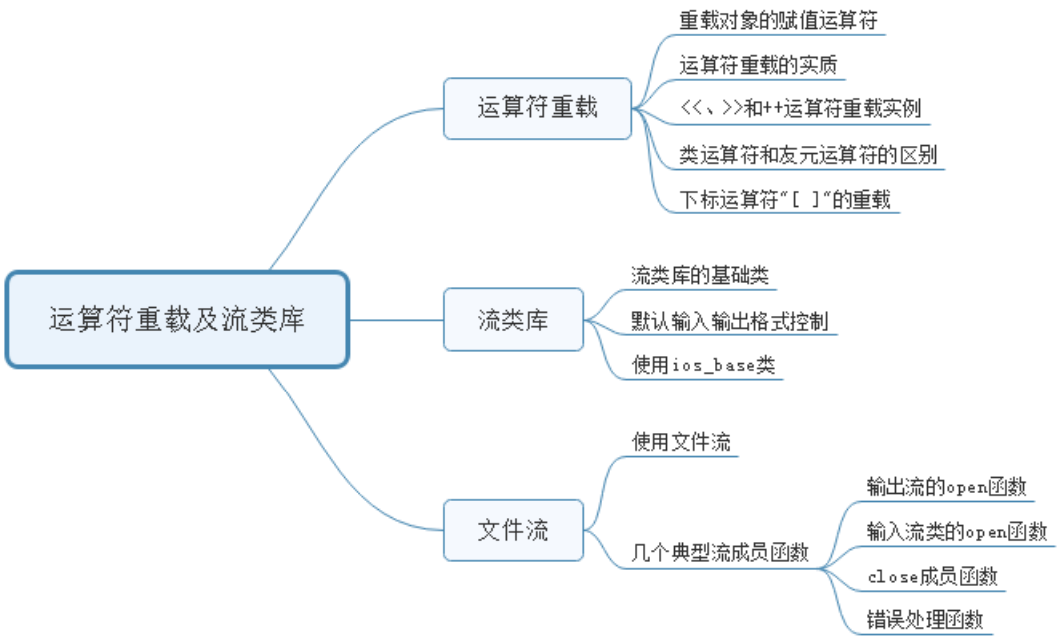
答案：C
解析：虚函数只能是类中的一个成员函数，但不能是静态成员，关键字 virtual 用于类中该函数的声明中。例如：

```
class A{
        public:
                virtual void fun( );//声明虚函数
} ;
void A::fun( ){...//}   //定义虚函数
```

本章的目的是引入 C++语言的多态性和虚函数的用途，要求理解编译时的多态性和运行时的多态性，掌握多态性和虚函数的有关知识。重点：虚函数、纯虚函数、抽象类及产生多态性的条件。难点：虚函数实现多态性的条件、多重继承与虚函数、类成员函数指针与多态性。

第九章 运算符重载及流类库

思维导图



模块一　运算符重载

知识点一　重载对象的赋值运算符

编译器在默认情况下为每个类生成一个默认的赋值操作，用于同类的两个对象之间相互赋值，默认的含义是逐个为成员赋值，即将一个对象的成员的值赋给另一个对象相应的成员，但这种赋值方式对于某些类可能是不正确的，如类 str 的数据成员是“char *st”，则语句“str s1(“hello”),s2(“world”); s2=s1;”经赋值后，s2.st 和 s1.st 是同一块存储地址，当 s2 和 s1 的生存期结束时，存储”hello”的变量将被删除两次，这是个严重的错误。另外，对于”s1=s1;”的情况，也应不执行赋值操作。

C++中关键字 operator 和运算符连用就表示一个运算符函数，如”operator=”表示重载赋值运算符“=”，又如”operator+”表示重载“+”运算符，所以应将“operator=”和“operator+”从整体上视为一个（运算符）函数名。运算符函数虽然是函数，但完全可以不写成函数调用的形式，而是采用原来的书写习惯，系统将会自动按照其真正的含义执行，运算符重载就是函数重载，这就是运算符重载的实质。

知识点二　运算符重载的实质

一般地，为用户定义的类型重载运算符都要求能够访问这个类型的私有成员，为此需通过两种方法实现，即将运算符重载为这个类型的成员函数或友元，前者称类运算符，后者称友元运算符。C++的运算符大部分都可以重载，除了. :: * 和? :五种运算符，前三者是因为有特殊含义，后两者是因为不值得重载；另外，sizeof 和#不是运算符，故不能重载；而= （ ） [ ] ->这四个运算符只能用类运算符来重载。

知识点三　<<、>>和++运算符重载实例

（1）插入符函数的一般形式如下：

```
ostream &operator<<(ostream & output,类名 & 对象名)
{ ..... //函数代码
return output;
}
```

（2）提取符函数的一般形式如下：

```
istream &operator>>(istream & input,类名 & 对象名)
{ ..... //函数代码
```

```
return input;
}
```

（3）有的 C++运算符编译器不区分前缀或后缀运算符，这是只能通过对运算符函数进行重载来反映其为前缀或后缀运算符。注意不能自己定义新的运算符，只能是把 C++原有的运算符用到自己设计的类上面去，同时，经过重载，运算符并不改变原有的优先级，也不改变它所需的操作数数目。当不涉及到定义的对象时，它仍然执行系统预定义的运算，只有用到自己定义的对象上，才执行新定义的操作。

知识点四　类运算符和友元运算符的区别

如果运算符所需的操作数（尤其是第一个操作数）希望进行隐式类型转换，则运算符应通过友元来重载；另一方面，如果一个运算符的操作需要修改对象的状态，则应当使用类运算符，这样更符合数据封装的要求。但参数是使用引用还是对象，则要根据运算符在使用中可能出现的情况而定。C++同时具有类运算符和友元运算符，参数也可以使用对象或引用，这就给编程带来了极大的方便，但也要注意各自适用的特定场合。

知识点五　下标运算符"[ ]"的重载

C++中，运算符[]只能用类运算符来重载。

模块二　流类库

知识点一　流类库的基础类

（1）在 C++中，输入输出是通过流完成的，把接收输出数据的地方称为目标，输入数据的出处称为源，而输入输出操作可以看成字符序列在源、目标以及对象之间的流动。C++将与输入和输出有关的操作定义为一个类体系，放在一个系统库里，以备用户调用，这个类体系称为流类，提供这个流类实现的系统库称为流类库。

（2）从 ios 类公有派生的 istream 和 ostream 两个类分别提供对流进行提取操作和插入操作的成员函数，而 iostream 类通过组合 istream 类和 ostream 类来支持对一个流进行双向操作，它并没有提供新的成员函数。

（3）C++的流类库预定义了 4 个流：cin、cout、cerr 和 clog，事实上，可以将 cin 视为 istream 的一个对象，将 cout 视为 ostream 的一个对象。流是一个抽象概念，在实际进行 I/O 操作时，必须将流和一种具体的物理设备联接起来，如 cin 联接键盘，cout、cerr 和 clog 联接显示终端。

知识点二　默认输入输出格式控制

（1）数值数据，默认方式是自动识别浮点数并用最短的方式输出。

（2）字符数据

- 单字符：默认方式是舍去空格，直到读到字符为止。
- 字符串：默认方式是从读到第一个字符开始，到空格符结束。
- 字符数组：默认方式是使用数组名来整体读入。
- 字符指针：尽管为其动态分配了地址，也只能采取逐个赋值的方法，不仅不以空格符结束，反而舍去空格，读到字符才计数。
- 因为字符串没有结束位，所以将其作为整体输出时，有效字符串的后面将出现乱码，这时可用手工增加字符串结束符“\0”来消除乱码。
- 当用键盘同时给一个单字符对象和一个字符串对象赋值时，不要先给字符串赋值，如果先给它赋值，应强行使用结束符。

（3）Bool（布尔型）数据，默认方式把输入的 0 识别为 false，其他的值均识别为 1。输出时，只有 0 和 1 两个值。

（4）如果默认输入输出格式不能满足自己的要求，就必须重载它们。

知识点三　使用 ios_base 类

（1） ios_base 类简介

ios_base 类派生 ios 类，ios 类又是 istream 类和 ostream 类的虚基类。

常用常量名	含义
left/right	输出数据按输出域左边/右边对其
dec/oct/hex	转换基数为十进制/八进制/十六进制形式
scientific/fixed	使用科学计数法/定点形式表示浮点数
skipws	跳过输入中的空白
internal	在指定任何引导标志或基之后填充字符
showbase	输出带有一个表示制式的字符
showpoint	浮点输出时必须带有一个小数点和尾部的 0
showpos	在正数前添加一个“+”号
uppercase	十六进制数值输出使用大写 A~F，科学计数显示使用大写字母 E
untibuf	每次插入之后，ostream::osfx 刷新该流的缓冲区，默认缓冲单元为 cerr
boolalpha	把逻辑值输出为 true 和 false（否则输出 1 和 0）
adjustfield	对齐方式域（与 left、right、internal 配合使用，如 ios_base::adjustfield）
basefield	数字方式域（与 dec、oct、hex 配合使用，如 ios_base::basefield）
floatfield	浮点方式域（与 fix、scientific，如 ios_base::floatfield）

成员函数	作用
long flags(long)	允许程序员设置标志字的值，并返回以前所设置的标志字
long flags()	仅返回当前的标志字
long setf(long,long)	用于设置标志字的某一位，第2个参数指定所要操作的位，第1个参数指定为该位所设置的值
long setf(long)	用于设置参数指定的标志位
long unsetf(long)	用于清除参数指定的标志位
int width(int)	返回以前设置的显示数据的域宽
int width()	只返回当前域宽（默认宽度为0）
char fill(char)	设置填充字符，设置的宽度小时，空余的位置用填充字符来填充，默认条件下是空格。该函数返回以前设置的填充字符。
char fill()	获得当前的填充字符
int precision(int)	返回以前设置的精度（小数点后的小数位数）
int precision()	返回当前的精度

（2）直接使用格式控制

常量名可直接用在系统提供的输入输出流中，而且有些是成对的，加no前缀表示取消原操作，如：

showbase	showpoint	showpos	skipws	uppercase	untibuf	boolalpha
noshowbase	noshowpoint	noshowpos	noskipws	nouppercase	nountibuf	noboolalpha

（3）使用成员函数

在使用成员函数进行格式控制时，setf 用来设置，unsetf 用来恢复默认设置，它们被流对象调用，使用常量设置，以实现科学计数法、对齐、带符号输出等功能。

模块二　文件流

知识点一　文件流

在C++中，文件操作是通过流来完成的，C++共有输入文件流、输出文件流和输入输出文件流3种，并已通过头文件<fstream>实现标准化。要打开一个输入文件流，需定义一个ifstream类型的对象；要打开一个输出文件流，需定

义一个 ofstream 类型的对象，如要打开输入输出文件流，则要定义一个 fstream 类型的对象。这三种类型都定义在头文件<fstream>中。

知识点二　使用文件流

（1）流类具有支持文件的能力，在建立和使用文件时，就像使用 cin 和 cout 一样方便，以写文件为例，如建立一个类似 cout 的输出文件流 myFile，只要它是 ofstream 类的对象，就可以像使用 cout 一样，使用 myFile 将指定内容写入文件；同理，如建立一个类似 cin 的输入文件流 getText，只要它是 ifstream 类的对象，就可像使用 cin 一样，用其读取文件内容。

（2）对文件进行操作的方法

- 打开一个相应的文件流，如“ofstream myStream;”打开一个输出文件流。
- 把这个流和相应的文件关联起来，如“myStream.open(“myText.txt”);”，就是通过 open 函数把两者关联起来，以后对文件流的操作就是对文件的操作。
- 操作文件流，想对文件进行什么操作，对相应的文件流进行相应操作即可。
- 及时关闭不再使用的文件，格式为：“文件流名.close();”

知识点三　几个典型流成员函数

（1）输出流的 open 函数

- 函数原型：void open(char const *,int filemode,int=filebuf::openprot);
- 参数：第一个参数是要打开的函数名；第二个是文件打开方式；第三个是文件的保护方式，一般都使用默认值。
- 参数具体设置：第二个参数 filemode 可取以下值：
  ios_base::in 打开文件进行读操作，这样可避免删除现存文件的内容
  ios_base::out 打开文件进行写操作，这是默认方式
  ios_base::ate 打开一个已有的输入或输出文件并查找到文件尾
  ios_base::app 打开文件以便在文件尾部添加数据
  ios_base::binary 指定文件以二进制方式打开，默认为文本方式
  ios_base::trunc 如文件存在，将其长度截断为零并清除原有内容
  上面的这几种方式也可通过“或”运算符“|”同时使用

（2）输入流类的 open 函数

- 使用默认构造函数建立对象，然后调用 open 成员函数打开文件，如：
  ifstream inFile; //用构造函数建立文件流对象 inFile
  inFile.open(“filename”,iosmode); //文件流对象打开文件 filename
- 可以使用指针方式，如：
  ifstream * pinFile; //动态建立文件流对象，获得对象指针

pinFile->open(“filename”,iosmode); //用对象指针调用 open 函数打开文件 filename

- 可以直接使用有参数构造函数指定文件名和打开模式，如：
  ifstream inFile(“filename”,iosmode);
- 输入流的打开模式 iosmode 有两种参数：
  ios_base::in //打开文件用于输入（默认方式）
  ios_base::binary //指定文件以二进制方式打开，默认为文本方式

（3）close 成员函数

close 成员函数用于关闭与一个文件流相关联的磁盘文件，如果没有关闭该文件，因文件流是对象，在其生存期结束时，也将自动关闭该文件，但要养成及时关闭不再使用的文件的习惯，以避免误操作或干扰而产生错误。

（4）错误处理函数

- 各函数功能：在对一个流对象进行 I\O 操作时，可能产生错误，这时可使用文件流的错误处理成员函数进行错误类型判别，各函数作用如下表：

函数	功能
bad()	如果进行非法操作，返回 true，否则返回 flase
clear()	设置内部错误状态，如果用缺省参量调用则清除所有错误位
eof()	如果提取操作已经到达文件尾，则返回 true，否则返回 flase
good()	如果没有错误条件和没有设置文件结束标志，返回 ture，否则返回 flase
fail()	与 good 相反，操作失败返回 flase，否则返回 true
is_open	判定流对象是否成功的与文件关联，若是，返回 ture，否则返回 flase

- 可以使用上面这些函数检查当前流的状态，如：if(cin.good())
  cin>>data;
- 函数 clear 更多的是用于在已知流发生错误的情况下清除流的错误状态，也可用于设置流的错误状态。
- “ !”运算符经过了重载，它与 fail 函数执行相同的功能，因此表达式 if(! cout)等价于 if（cout.fail()）,if(cout)等价于 if(! cout.fail())
- 如果需要测试文件是否存在，可以使用 fail 成员函数确定，如：
  ifstream inFile（ “filename”,ios_base::binary）;
  if(inFile.fail()) cout<<”无错误” <<endl;

真题演练

1、（201704 单选题）以下关于运算符重载的描述中，错误的是（ ）。

A、运算符重载其实就是函数重载
B、成员运算符比友元运算符少一个参数
C、需要使用关键字 operator
D、成员运算符比友元运算符多一个参数

答案：D

解析 :因为任何运算都是通过函数来实现的 ,所以运算符重载其实就是函数重载，要重载某个运算符，只要重载相应的函数就可以了。与以往稍有不同的是，需要使用新的关键字“operator”。成员运算符比友元运算符少一个参数，这是因为成员函数具有 this 指针。因此 D 选项错误。

2、（201604 单选题）如果表达式++a 中的“++”是作为成员函数重载的运算符，若采用运算符函数调用格式，则可表示为（ ）

A、a.operator++(1)
B、operator++(a)
C、operator++(a,1)
D、:a.operator++( )

答案：D
解析：使用类运算符”++”运算符，使用函数调用方式，例：若前缀：++n 为 n.operator++( )；若后缀：n++为 n.operator++(0);，题目中为前缀++，因此选择 D 选项。

3、（201610 单选题）进行文件操作时需要包含头文件（ ）。

A、iostream
B、fstream
C、stdio
D、stdlib

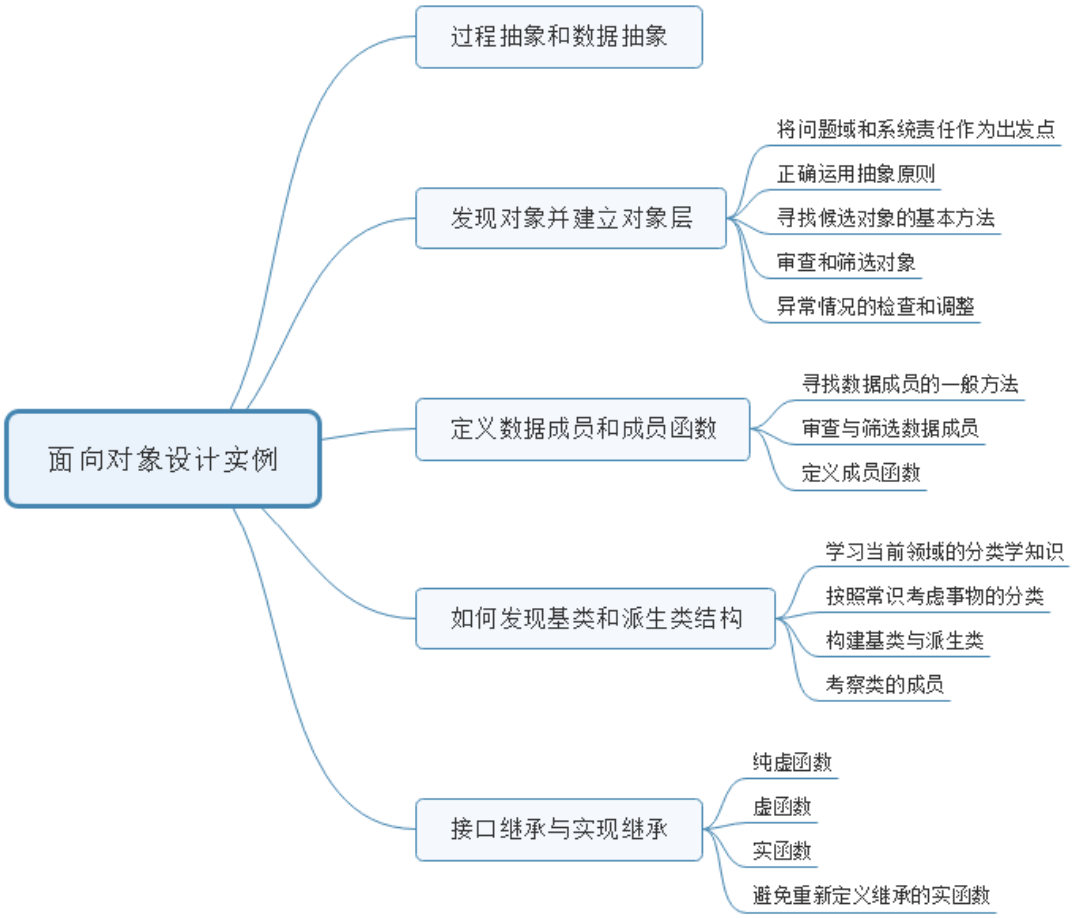
答案：B
解析：要打开一个输入文件流，需要定义一个 ifstream 类型的对象；要打开一

个输出文件流，需要定义一个 ofstream 类型的对象；如果要打开输入输出文件流 ,则要定义一个 fstream 类型的对象。这三种类型都定义在头文件<fstream>里。

本章的目的是学习运算符重载和文件概念 ,要求掌握运算符重载的基本知识、流类库的概念及使用流类库进行文件存取的概念。重点 :运算符重载、格式控制、文件存取。难点：运算符重载。

第十章 面向对象设计实例

思维导图



模块一 过程抽象和数据抽象

知识点一 抽象的概念

抽象是从许多事物中舍弃个别的、非本质性特征，抽取共同及本质性特征的过程。

知识点二 抽象的意义

（1）尽管问题域中事物很复杂，但分析员不需了解和描述其全部特征，只需要分析研究与系统目标有关的事物及其本质特征，其余的都可以舍弃；

（2）通过舍弃个体事物在细节上的差异，抽取其共同特征可以得到一些事物的抽象概念，如 OOA（面向对象设计）中的类。

知识点三 过程抽象和数据抽象

抽象是面向对象方法中使用最广泛的原则，早在面向对象方法出现之前就已经开始应用，主要是过程抽象和数据抽象。

（1）过程抽象：是指任何一个完成确定功能的操作序列，其使用者都可以把它看作一个单一的实体；

（2）数据抽象：是根据施加于数据之上的操作来定义数据类型，并限定数据的值只能由这些操作来修改和观察。。

模块二 发现对象并建立对象层

知识点一 步骤

（1）将问题域和系统责任作为出发点。前者侧重客观存在的事物与系统中对象的映射，后者侧重系统责任范围内的每一项职责都能落实到某些对象来完成，二者重合很多，但又不完全一致。

（2）正确运用抽象原则。OOA 使用对象映射问题域中的事物。通过舍弃与当前目标与系统责任无关的事物及其特征来完成抽象。

（3）寻找候选对象的基本方法。主要是从问题域、系统边界、系统责任三方面找出可能有的候选对象。

（4）审查和筛选对象。原则为：第一，舍弃无用对象，标准为看对象是否提供了有用的数据成员和成员函数；第二，对象精简，即将只有一个数据成员和成

员函数的对象合并到相关的对象中描述；第三，舍弃目前不需要考虑的对象。

（5）异常情况的检查和调整。包括以下的异常情况：第一，类的成员不适合该类的全部对象；第二，不同类的成员相同或相似；第三，对同一事物的重复描述。

模块三 定义数据成员和成员函数

知识点一 寻找数据成员的一般方法

（1）审查时去掉无用的数据。
（2）确定在当前的问题域中，对象应有的数据成员。
（3）根据系统责任的要求，考虑对象应有的数据成员。
（4）考虑建立这个对象是为了保存哪些信息。
（5）考虑为了在服务中实现其功能，需要增设哪些数据成员。
（6）考虑对象有哪些需要区别的状态，是否需要增加一个数据成员来区分它们。
（7）考虑用什么数据成员报市整体 - 部分结构合实例连接。

知识点二 审查与筛选数据成员

通过数据成员是否满足下面条件来进行审查和筛选：
（1）是否体现了以系统责任为目标的抽象。
（2）是否描述了这个对象本身的特征。
（3）是否符合人们日常的思维习惯。
（4）是否可通过继承得到，基类中定义了的数据成员，都不要在派生类中出现。
（5）是否可从其他数据成员直接导出。

知识点三 定义成员函数

使用中要注意区分成员函数、非成员函数和友元函数三者。设计一个类时，要注意接口功能不仅要完整紧凑，而且要使类的结构达到最小化。

模块四 如何发现基类和派生类结构

知识点一 学习当前领域的分类学知识

分析员应该学习一些与当前问题域有关的分类学知识，然后按照本领域已有的分类方法，可以找出一些与对象对应的基类与派生类。

知识点二 按照常识考虑事物的分类

如问题域没有可供参考的现行分类方法，可按照自己的常识，从各种不同的角度考虑事物的分类，分类要考虑是否符合分类学常识，基类与派生类结构中的各个类之间的关系应符合分类学的常识和人类的日常思维方式。

知识点三 构建基类与派生类

思路有两种，一种是把每一个类看作是一个对象集合，分析这些集合之间的包含关系，如果一个类是另一个类的子集，则它们就应组织到同一个基类与派生类结构中；另一种是看一个类是否具有另一个类的全部特征，又分两种情况：一是建立这些类时已经计划让某个类继承另一个类的全部成员，此时应该建立基类与派生类结构来落实这种继承；二是起初只是孤立的建立每一个类，现在发现一个类中定义的成员全部在另一个类中重新出现了，此时应考虑建立基类与派生类结构，把后者作为前者的派生类，以简化定义。

知识点四 考察类的成员

从下面两方面考察每个类的成员：一是看一个类的成员是否适合这个类的所有对象；二是检查是否有两个或更多类含有一些共同的成员，如果有，则考虑若把这些共同的成员提取出来后能否构成一个在概念上是包含原来那些类的基类，组成一个基类与派生类结构。

模块四 接口继承与实现继承

知识点一 成员函数分类

类通过成员函数提供与外界的接口，成员函数分为实、虚、纯虚三种。

（1）纯虚函数

纯虚函数最显著的两个特征是：它们必须由继承它的非抽象类重新说明；它们在抽象类中没有定义。

说明一个纯虚函数的目的是使派生类继承这个函数的接口，但这种抽象的基类从不给派生类提供实现。

（2）虚函数

虚函数的情况不同于纯虚函数，派生类继承虚函数的接口，虚函数一般只提供一个可供派生类选择的实现。之所以声明虚函数，目的在于使派生类既继承函数接口，也继承虚函数的实现。

声明虚函数的目的，是使派生类既继承函数接口，又能继承虚函数的实现。

（3）实函数

声明实函数的目的，是使派生类既继承这个函数的实现，也继承其函数接口。

（4）应注意尽量避免重新定义继承的实函数

虽然在派生类中可以重定义基类的同名实函数，但是从使用安全角度来看，为了提高程序质量，在实际应用中应避免这样做。

知识点二 函数接口的继承和函数实现的继承

假设有一个基类，其派生类通过公有继承方式，公有继承实际上分函数接口的继承及函数实现的继承两部分，可概括为：

（1）继承的总是成员函数的接口，对于基类是正确的任何事情，对于它的派生类必须也是正确的，反之则不然；

（2）声明纯虚函数的目的，是使派生类仅仅继承函数接口，而纯虚函数的实现则由派生类去完成；

（3）声明虚函数的目的，是使派生类既能继承基类对此虚函数的实现，又能继承虚函数提供的接口；

（4）声明实函数的目的，是使派生类既能继承基类对此实函数的实现，又能继承实函数提供的接口。

真题演练

1、 声明实函数的目的是使（ ）。

A、派生类既继承函数的实现，也继承函数接口

B、派生类只继承函数接口

C、派生类只继承函数的实现

D、派生类改写实函数

答案：A

解析 :派生类继承虚函数的接口，一般虚函数只提供一个可供派生类选择的实现。声明虚函数的目的是使派生类既继承函数接口，也继承虚函数的实现。

本章的目的是要求考生认识到面向对象语言和面向对象编程不能简单等同。重点：过程抽象和数据抽象、接口继承与实现继承。难点：接口继承与实现继承。