



C++语言程序设计

C++强化训练

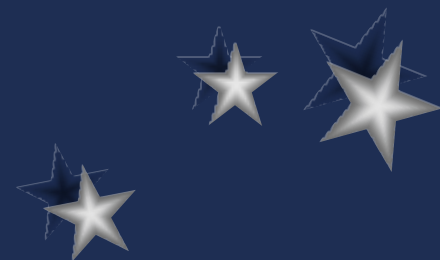
第二部分 构造化程序设计

东南大学 郑雪清



强化训练安排:

- 计算机基础总结
- 模板
- 构造化程序设计语法及算法
[包括指针与链表]
- 面向对象程序设计语法及算法
- 江苏省二级上机分析
- 国家二级理论试卷分析
- 江苏省二级理论试卷分析
- 总结、复习



强化训练目的:

- 加强程序设计基础知识掌握与C++基本编程能力的培养。涉及基本语法和基本算法。
- 掌握指针及指针的应用。
- 掌握面向对象程序设计几种要素：封装与隐藏、继承和派生、多态、静态组员和友员函数。掌握运算符重载真正意义及实现。
- 掌握面向对象程序设计的基本措施。
- 为国家和省二级做准备。



第二部分 构造化程序设计

一、体现式及优化和有关的副作用

1. 字符何时是字符何时是数字？

```
cout<<'a'+10<<char('a'+10); 107 k
```

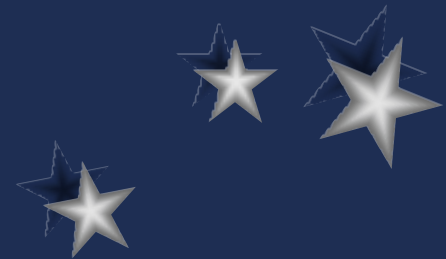
```
char a='A';
```

```
cout<<a+10<<char(a+10); 75 K
```

```
char a[]="I am a student.";
```

```
cout<<a<<*a<<a+5<<*(a+5);
```

```
I am a student | a student a
```



第二部分 构造化程序设计

2. 字符常量使用转义序列时。

'\84' (错!) 和 '\400' (是'\0')

3. 数据类型转换

类型转换：自动类型转换和强制类型转换两种。

- 自动类型转换：原则是精度低的往精度高的转换

(1) `int a=5,b=4,c=3,d;d=a*b / c-1.5+'a';`

成果是：20/3-1.5+97，d的值是101。

(2) `int j,j;float x,y;`

`x=(i=4.8)+(j=5.9);`



第二部分 构造化程序设计

● 强制类型转换

强制类型转换有两种格式：

(<类型名><体现式> 或 <类型名> (<体现式>)

例(1): (float*)5000 强制把整数5000转为地址。

例(2): float(100)转为单精度。

注意： a.第二种格式又称类型构造。

b.强制类型转换是圆括号运算符的一种。

c.圆括号运算符有三种：强制类型转换、类型构造、函数调用。

d.当数据类型（涉及基本数据类型或导出数据类型）在圆括号中或后跟圆括号，注意数据转换。

第二部分 构造化程序设计

(1)设有阐明“`int x=10,y=4,f;float m;`”,执行体现式“`f=m=x/y;`”后, f,m的值为D。

A.2,2.5 B.3,2.5 C.2.5,2.5 D.2,2.0

(2)数学式 $(3xy)/(5ab)$, 其中x和y是整数, a和b是实数, 在C++中相应的正确体现式是D。

A. $3/5*x*y/a/b$

B. $3*x*y/5/a/b$

C. $3*x*y/5*a*b$

D. $3/a/b/5*x*y$



第二部分 构造化程序设计

4.逻辑体现式优化及副作用

在C++中，在求逻辑体现式的过程中，一旦能拟定逻辑体现式的值时，就不必再逐渐求值了。这就是逻辑体现式的优化。

例(1): `int a=0,b=4,c=5,d,e;`

`d=a&&b++&&--c; e=a||b++||--c;`

例(2): `int a=2,b=4,c=5,d,e;`

`d=a&&b++&&--c; e=a||b++||--c;`

第二部分 构造化程序设计

(1) 设有阐明“int a=3,b=5,m;”，则执行体现式“m=a<=3&&a+b<8;”后，m的值为0。

(2) 设有阐明“int x=5,y=7,z=8;”，则执行体现式“z+=x++||y++||++z;”后，x,y,z为A。

A.6,7,9 B.6,8,10 C.6,8,8 D.6,8,1

(3) 设有阐明“int a=15,b=17,c;”，执行体现式“c=a||(b+=b);”后，a,b,c的值为A。

A.15,17,1 B.1,34,35
C.15,34,1 D.15,17,15

第二部分 构造化程序设计

二、三种基本构造[主要是分支和循环]

1. 条件语句

if条件语句嵌套时，else语句总是与近来的、没有与else配正确if语句配对。

例： `if(...) if(...) else (...)`

2. 开关语句(switch)

要注意语句格式、语法和break语句的作用。

注意： =和==混同。



第二部分 构造化程序设计

3. 循环构造语句

C++中循环构造语句有:`while()`、`do...while()`、和`for()`三种循环语句。

循环语句掌握的内容是:

- a. 语句格式、语法，`for`语句中三个体现式的作用和执行时的顺序。
- b. 进入循环前给有些变量怎样提供初始化；
- c. 输出成果的时机：在循环体内还是循环结束后输出。
- d. 循环语句和`break`，`continue`语句。

第二部分 构造化程序设计

(1).循环语句“for(int x=0,y=0;y!=100||x<10;)x++;”,执行的循环次数是 A 。

A. 无限次 B. 10 C. 11 D. 100

(2)有下列程序段: int k=0;while(k=1)k++;执行的循环次数是 C 。

A.有语法错,不能执行 B.执行1次
C.无限次 D.一次也不执行

(3)下列程序段的输出成果是 yes 。

```
int a=1,b=3,c=5;
```

```
if(c=a+b)cout<<"yes" else cout<<"no";
```



第二部分 构造化程序设计

(4) 下列程序段的输出成果是 20 0。

```
int x=10,y=20,t=0;
if(x==y)t=x;x=y;y=t;
cout<<x<<'\t'<<y<<'\n';
```

(5) 执行下列程序段时，输出 A。

```
int s1=0,s2=0,s3=0,s4=0;
for(int t=1;t<=4;t++)
switch(t)
{ case t>=4:s1++;break;
  case t>=3:s2++;
  default: s4++;}
cout<<s1<<','<<s2<<','<<s3<<','<<s4<<'\n';
```

A. 语法错，编译不经过

B. 1,1,1,2

C. 1,2,3,2

D. 1,1,2,2

第二部分 构造化程序设计

(6) 下列程序运营后的输出成果是3, 3。

```
#include<iostream.h>
void main()
{  int k=4,n=0;
   for(;n<k;){
       n++;
       if(n%3!=0)continue;
       k--;}
   cout<<k<<`,`<<n;
}
```



第二部分 构造化程序设计


三、函数

1. 函数的基本知识

(1). 函数定义

函数定义时，有返回值要注意函数的返回值类型与return语句后的体现式类型相一致，函数没有返回值时（void），则用 return ； 语句。

函数能够嵌套调用，但不能够嵌套定义。在一种函数的函数体中定义另一种函数是非法的。



第二部分 构造化程序设计

(2).函数调用

调用函数时，实参和形参要求一一相应。

函数调用有两种：作为体现式的函数调用和函数调用语句。对于没有返回值的函数调用只能经过函数调用语句实现。

(3).函数调用参数传递

参数传递有三种：值传递、地址传递和引用传递。

第二部分 构造化程序设计

(4).作用域（难点）与存储类

在C++中，作用域共分为五类：块作用域；文件作用域；函数原型作用域；函数作用域；类作用域。

局部变量：在一种函数内部定义的变量或在一种块中定义的变量称为局部变量。

全局变量：在函数外定义的变量或用extern阐明的变量称为全局变量。全局变量的作用域称为文件作用域，即在整个文件中都能够访问。

第二部分 构造化程序设计

(1) 设有函数定义调用语句
“f((e1,e2),(e3,e4,e5));”，则实参个数是____A____。

(A) 2 (B) 3 (C) 4 (D) 5

(2) 下列函数中对调用它的函数没有起到任何作用的是
C。

(A) `void f1(double &x){--x;}`

(B) `double f2(double x){return x-1.5;}`

(C) `void f3(double x){--x;}`

(D) `double f4(double *x){--*x;return *x;}`

第二部分 构造化程序设计

(3) 下列程序输出的第一行是 0 1 2 3 4 第二行是 15。

```
#include<iostream.h>      (作用域)
```

```
int k;
```

```
void main(void)
```

```
{ k=10;
```

```
for(int i=0;i<5;i++){
```

```
    int k;k+=i;
```

```
//因为重新定义变量k，所以就有不同的作用域
```

```
    cout<<k<<'\n';k++;::k++;}
```

```
cout<<'\n'<<k<<'\n';
```

```
}
```

第二部分 构造化程序设计

2. 递归函数

在利用递归措施求值时， 必须注意三点：

- 1.递归的公式；
- 2.递归的结束条件；
- 3.递归的约束条件。

关键是找到递归公式和递归的结束条件。

递归函数在调用时分两部分： 递推和回归。



第二部分 构造化程序设计

(1). 下列程序的输出成果是（vrg）。

```
#include<iostream.h>
void func2(int i);
char s[]="verygood!"
void func1(int i)
{ cout<<st[i];if(i<3){i=+2;func2(i);}}
void func2(int i)
{ cout<<st[i];if(i<3){i=+2;func1(i);}}
void main()
{ int i=0;func1(i);}
```



第二部分 构造化程序设计

(2). 下列程序的输出成果是 (15)。

```
#include<iostream.h>
long fib(int n)
{  if(n>2) return(fib(n-1)+fib(n-2));
   else return 5*n;
}
void main()
{  cout<<fib(3);}
```



第二部分 构造化程序设计

(3) 下列程序输出的第一行是 1 1 第二行是 4 1 最终一行是 x=10。

```
#include<iostream.h>
void fun(int n,int *s)
{ int f1,f2;
  if(n==1||n==2)*s=1;
  else { fun(n-1,&f1);fun(n-2,&f2);
        *s=2*f1+f2+1;
        cout<<f1<<'\t'<<f2<<'\n';}}
void main(void)
{int x;fun(4,&x);cout<<"x="<<x<<'\n';}
```

第二部分 构造化程序设计

(4) 下列程序输出的成果是 dcba 。

```
#include<iostream.h>
void show(char *s)
{if(*s){show(s+1);cout<<*s;}}

void main(void)
{ show("abcd\0efg\0hij");
  cout<<'\n';}
```



第二部分 构造化程序设计

(5) 下列程序输出的第一行是 123 第二行是 321
第三行是 123。

```
#include<iostream.h>
void p1(char s[],int i)
{ if(s[i]!=0)p1(s,i+1); cout<<s[i];}
void p2(char s[],int i)
{ if(s[i]!=0)p2(s,i+1); cout<<s[i];}
void main(void)
{ char str[]="123";cout<<str<<'\n';
  p1(str,0);cout<<'\n';
  p2(str,0);cout<<'\n';}
```



第二部分 构造化程序设计

3.重载函数

定义的重载函数必须具有：相同的函数名但具有不同的参数个数或不同的参数类型。

调用重载该函数时，将根据实参的数据类型和重载函数的形参匹配，拟定调用其中的一种重载函数。



第二部分 构造化程序设计

4. 静态变量

静态类型变量有拟定的初值。静态类型变量对全局变量和局部变量有不同的含义。

静态类型变量只能初试化一次。当局部变量使用静态类型变量时，其作用将保存函数的运营成果，以便下次调用函数时，能继续使用上次计算的成果。但不在变量的作用域时，不能直接使用，确需使用时，可经过变量的地址取值（即指针）。

当全局变量使用静态类型变量时，表达所阐明的变量仅限于这个源程序文件内使用。

第二部分 构造化程序设计

(1) 下列程序输出的成果是 i=105 i=110

。

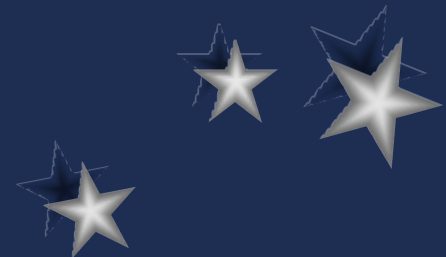
```
#include<iostream.h>
```

```
int t()
```

```
{ static int i=100;  
  i+=5; return i;}
```

```
void main(void)
```

```
{ cout<<"i="<<t()<<"\n";  
  cout<<"i="<<t()<<"\n"; }
```



第二部分 构造化程序设计

(2) 下列程序输出的成果是 i=105 105
i=105 110。

```
#include<iostream.h>
int t(int **p)
{ static int i=100;*p=&i;    i+=5;    return i;}
void main(void)
{ int i,*p;i=t(&p);
  cout<<"i="<<i<<"\t" <<*p<<"\n";
  t(&p);    cout<<"i="<<i<<"\t" <<*p<<"\n"; }
```



第二部分 构造化程序设计

(3) 下列程序输出的第一行是 20，第二行是 400

。

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

```
int f(int x)
```

```
{ static int u=1;
```

```
  x+=x;return u*=x;}
```

```
void main(void)
```

```
{ int x=10;
```

```
cout<<f(x)<<'\n';cout<<f(x)<<'\n';}
```




第二部分 构造化程序设计

(4) 下列程序输出的第一行是 21，第二行是 53。

```
#include<iostream.h>
int f(void)
{ static int a,b=10,c=1;
  a+=b;b+=c;return c=a+b;}

void main(void)
{ cout<<f()<<'\n';cout<<f()<<'\n';}
```



第二部分 构造化程序设计

四、编译预处理指令

编译预处理指令涉及三方面的内容：文件涉及、宏指令和条件编译，其要点是宏指令——带参数的宏定义。

注意：在宏调用时，将不作任何语法检验，也不做任何计算，只作简朴替代。



以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/798075077130006132>