C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

教材介绍

C++程序设计

（2008年版）

编著：刘振安

机械工业出版社

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×20题 = 20分

程序改错题　4分×5题 = 20分

完成程序题　4分×5题 = 20分

程序分析题　5分×2题 = 10分

程序设计题 10分×1题 = 10分

第三章　函数和函数模板

本章主要内容

- 函数的参数及其传递方式
- 深入讨论函数返回值
- 内联函数
- 函数重载和默认参数
- 函数模板

函数复习

一、函数的参数：函数的参数分为形参和实参两种。形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。形参和实参的功能是做数据传送，发生函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送。

函数复习

二、函数的形参和实参具有以下特点：

1、形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效，函数调用结束返回主调函数后，则不能再使用该形参变量。

2、实参可是是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此，应预先用赋值、输入等办法使实参获得确定值。

3、实参和形参在数量上、类型上、顺序上应严格一致，否则会发生“类型不匹配”的错误。

4、函数调用中发生的数据传送是单向的，即只能把实参的值传送给形参，而不能把形参的值反向的传给实参，因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

§3.1 函数的参数及其传递方式

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。

对象作为函数参数

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。

例：传对象不改变原来对象数据成员值的例子。

```
#include <iostream>
#include <string>
using namespace std;

void swap(string,string);   //定义函数swap，使用string类的对象作为函数形参
void main( ){
    string str1("现在"),str2("过去"); //定义对象str1和str2
    swap(str1,str2);  //使用传值方式传递函数实参str1和str2的数据成员值
    cout<<"返回后：  str1 = "<<str1<<"str2="<<str2<<endl;
}
void swap(string s1,string s2)  //定义string类的对象s1和s2作为函数形参
{
    string temp=s1;s1=s2;s2=temp;
    cout<<"交换为：  str1 = "<<s1<<"str2="<<s2<<endl;
}
```

交换为： str1 = 过去str2=现在
返回后： str1 = 现在str2=过去

对象指针作为函数参数

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。

例：使用对象指针作为函数参数的例子。

```
#include <iostream>
#include <string>
using namespace std;
void swap(string *,string *); //定义函数swap，使用string类的指针作为函数形参
void main( ){
    string str1("现在"),str2("过去");  //定义对象str1和str2
    swap(&str1,&str2);  // 因函数原型中参数的类型是指针，所以string *s1=&str1是完全正确的，
//即在主调函数中可将对象str1和str2的首地址值&str1和&str2作为实参，并传递给形参
    cout<<"返回后：  str1 = "<<str1<<"str2="<<str2<<endl;
}

void swap(string *s1,string *s2) //string类的对象指针s1和s2作为函数形参
{
    string temp=*s1;*s1=*s2;*s2=temp;
    cout<<"交换为：  str1 = "<<*s1<<"str2="<<*s2<<endl;
}
```

交换为： str1 = 过去str2=现在
返回后： str1 = 过去str2=现在

数组作为函数参数

在以数组作为函数参数时，因数组名即是数组指针名，指向数组首地址（例如a为数组名，则a同时也是数组指针名，a+1则指向数组a的第一个元素），所以也能采用传地址值的方式，当交换后，因形参的改变，实参也会同时改变。

例：传递数组名实例。

```
#include <iostream>
using namespace std;

void swap(int[ ]);
void main( ){
int a[ ]={3,8};
swap(a);
cout<<"返回后：a="<<a[0]<<" b="<<a[1]<<endl;
}

void swap(int a[ ])
{
    int temp= a[0];a[0]=a[1];a[1]=temp;
    cout<<"交换为：a="<< a[0]<<" b="<<a[1]<<endl;
}
```

程序运行结果为：交换为：a = 8 b = 3
返回后：a = 8 b = 3

引用作为函数参数

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。

例：使用引用作为函数参数的例子。

```
#include <iostream>
#include <string>
using namespace std;
void swap(string &,string &);//函数swap，使用string类的引用对象作为函数形参
void main( ) {
    string str1("现在"),str2("过去");  //定义对象str1和str2
    swap(str1,str2); // 传递对象的名字str1和str2
    cout<<"返回后：  str1 = "<<str1<<" str2="<<str2<<endl;
}

void swap(string &s1,string &s2) //string类的引用对象s1和s2作为函数形参
{
    string temp=s1;s1=s2;s2=temp;
    cout<<"交换为：  str1 = "<<s1<<" str2="<<s2<<endl;
}
```

程序输出结果：

交换为： str1 = 过去 str2 = 现在

返回后： str1 = 过去 str2 = 现在

可以通过间接引用的方法使用数组

五、可以通过间接引用的方法使用数组

程序输出结果：

平均成绩为63.6分，不及格人数为4人。

```
#include <iostream>
using namespace std;
typedef double array[12];      //自定义数组标识符array
void avecount(array& b,int n)  //定义函数avecount，其形参一个使用引用，一个使用对象
{
    double ave(0);
    int count(0);  //累加器初始化0
    for(int j=0;j<n-2;j++){
        ave=ave+b[j];
        if(b[j]<60) count++; }
    b[n-2]=ave/(n-2);  //平均成绩
    b[n-1]=count;      //不及格人数
}
void main( ){
    array b={12,34,56,78,90,98,76,85,64,43};
    array &a=b;
    avecount(a,12);
    cout<< "平均成绩为"<<a[10]<<"分，不及格人数为"<<int(a[11])<<"人。"<<endl;
}
```

六、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。例如：

int SaveName(char *first,char *second=”“,char *third=”“,char *fourth=”“);

表明在实际调用函数SaveName时，如不给出参数second、third和fourth，则取默认值“”。

另外，如果某个默认参数需要指明一个特定值，则在此之前的所有参数都必须赋值，如上例中，如果需要给出参数third的值，则必须同时也给first和second赋值。

例：设计一个根据参数数量输出信息的函数

```
#include <iostream>
#include <string>
using namespace std;
void display(string s1,string s2=" ",string s3=" ");
void main( ){
    string str1("现在"),str2("过去"),str3("未来");
    display(str1);
    display(str1,str2,str3);
    display(str3,str1);
    display(str2,str3);
}
void display(string s1,string s2,string s3)
{
   if(s2==" "&s3==" ")
       cout<<s1<<endl;
   else if(s3==" "&s2!=" ")
              cout<<s1<<"、 "<<s2<<endl;
   else
              cout<<s1<<"、 "<<s2<<"、 "<<s3<<endl;
}
```

现在
现在、 过去、 未来
未来、 现在
过去、 未来

使用const保护数据

七、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。

例：不允许改变作为参数传递的字符串内容的实例

```
#include <iostream>
#include <string>
using namespace std;

void change(const string&);
void main( ){
    string str("can you change it?");
    change(str);
    cout<<str<<endl;
}

void change(const string&s)
{
    string s2=s+"no!";
    cout<<s2<<endl;
}
```

```
can you change it?no!
can you change it?
```

§3.2 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。

一、返回引用的函数

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）；

例：返回引用的函数实例

```
#include <iostream>
using namespace std;

int a[ ]={2,4,6,8,10,12};    //定义全局数组a
int & index(int i);              //返回引用的函数index的原型声明

void main( ){
index(3)=16;              //将a[3]的值改为16
cout<<index(3)<<endl;    //输出a[3]的值16
}

int & index(int i)          //定义返回引用的函数index
{return a[i];}
```

二、返回指针的函数

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）；

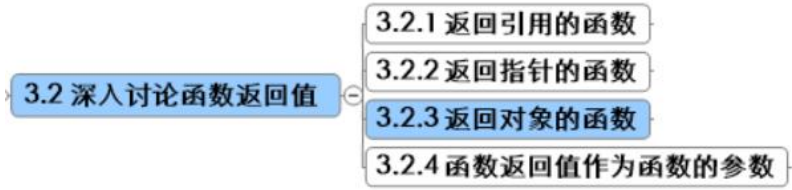
例：使用函数input输入一组数并返回一个指针，然后由main显示这组数

```
#include <iostream>
using namespace std;
float * input(int&);      //声明返回指针的函数input

void main( )
{
    int num;
    float *data;          //声明与input类型一致的指针变量data
    data=input(num);      //调用函数，返回指针赋给data
    if(data){             //data不空，返回所指内容
        for(int i=0;i<num;i++)     //使用指针的下标形式
        cout<<data[i]<<" ";    //循环输出data内容
        delete data;               //释放指针占用的内存
    }
}
```

```
float *input(int& n)                //语句⑤，定义返回指针的函数input
{
    cout<<"Input number:";          //询问输入数据的个数
    cin>>n;
    if(n<=0) return NULL;           //输入的个数不合理则退出
    float * buf=new float[n];       //根据输入数据的个数申请所需空间
    if(buf==0) return NULL;         //申请不到空间则退出
    for(int i=0;i<n;i++)
        cin>>buf[i];
    return buf;                     //返回指针
}
```

```
Input number:3
12.25 45.32 78.94
12.25 45.32 78.94
```

三、返回对象的函数



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）

例：函数返回对象的实例

```
#include <iostream>
#include <string>
using namespace std;

string input(const int);     //声明返回string类的对象的函数

void main( ){
    int n;
    cout<<"Input n=";
    cin>>n;                  //输入要处理的字符串个数n
    string str=input(n);      //将函数返回的对象赋给对象str
    cout<<str<<endl;
}
```

```
string input(const int n)
{
    string s1,s2;                //建立两个string类的对象（均为空串）
    for(int i=0;i<n;i++)         //逐次接收n个字符串
        {cin>>s1;s2=s2+s1+" ";}   //逐次将接收的字符串相连
    return s2;                   //返回字符串相连的最后结果
}
```

Input n=3
zhang san feng
zhang san feng

四、函数返回值作为函数的参数

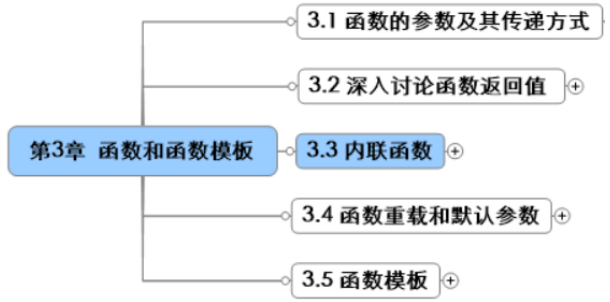
如果函数返回值作为另一个函数的参数，那么这个返回值必须与另一个函数的参数的类型一致。

例：函数返回值作为函数的参数的实例

```
#include <iostream>
using namespace std;

int max(int,int);    //声明含有两个整型参数的函数原型

void main( )
{   cout <<max(55,max(25,39))<<endl; }

int max(int m1,int m2)    //定义含有两个整型参数的函数原型
{return (m1>m2)?m1:m2;}
```

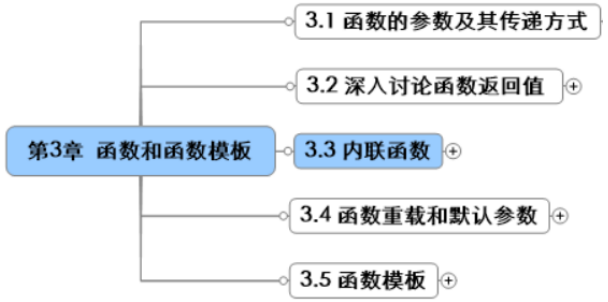
55

§3.3 内联函数



- 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。

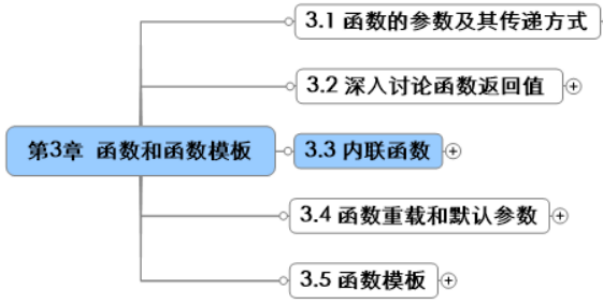
例：使用函数isnumber判定一个输入字符是否是数字。



1、不使用内联函数形式时的代码：

```
#include <iostream>
using namespace std;

int isnumber(char c)
{return (c>='0'&&c<='9')?1:0;}

void main( )
{
   char c;
   cin>>c;
   if(isnumber( c ))
        cout<<"你输入了一个数字";
   else
        cout<<"你输入的不是一个数字";
}
```

```
9
你输入了一个数字
```

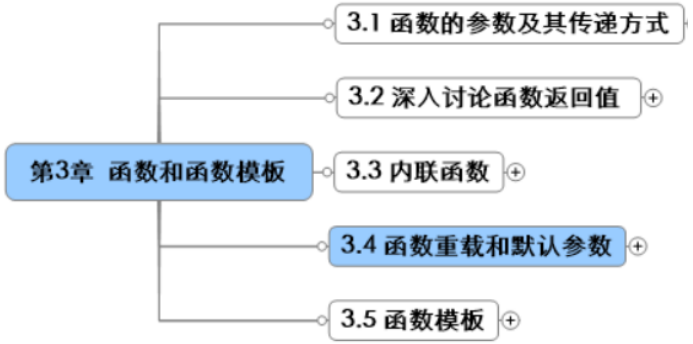


2、使用内联函数形式时的代码：

```
#include <iostream>
using namespace std;

inline int isnumber(char c)
{return (c>='0'&&c<='9')?1:0;}

void main( )
{
   char c;
   cin>>c;
   if(isnumber( c ))
       cout<<"你输入了一个数字";
   else
       cout<<"你输入的不是一个数字";
}
```

```
9
你输入了一个数字
```

将函数main中对函数isnumber的调用替换成表达式，即将mian函数中语句“if(isnumber(c)) ”转换为“if((c>=’0’&&c<=’9’)?1:0);”，这种替换手工来作很麻烦，可以让C＋＋编译程序来作，方法就是将函数isnumber的定义前面加上关键字inline，变为内联函数的形式即可，这样，C＋＋编译器在遇到对函数isnumber调用的地方都会用这个函数体替换该调用表达式。



§3.4 函数重载和默认参数

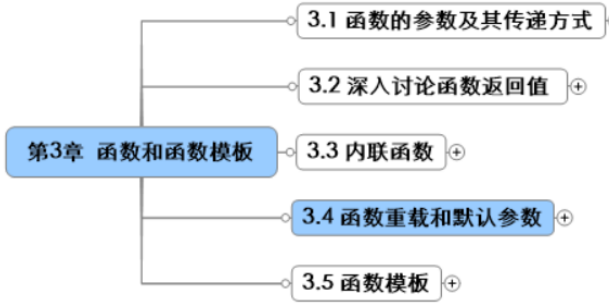
一、函数重载

- 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。 “一个名字，多个函数”
- 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。

例：函数重载产生多态性的例子

```
#include <iostream>
using namespace std;
double max(double,double);
int max(int,int);
char max(char,char);
int max(int,int,int);
void main( )
{
   cout<<max(2.5,17.54)<<" "<<max(56,8)<<" "<<max('w','p')<<endl;
   cout<<"max(5,9,4)="<<max(5,9,4)<<" max(5,4,9)="<<max(5,4,9)<<endl;
}
double max(double m1,double m2){return (m1>m2)?m1:m2;}
int  max(int m1,int m2){return (m1>m2)?m1:m2;}
char max(char m1,char m2){return (m1>m2)?m1:m2;}
int  max(int m1,int m2,int m3)
{   int t=max(m1,m2);
    return max(t,m3);}
```

17.54 56 w
max(5,9,4)=9 max(5,4,9)=9



函数重载与默认参数的结合

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。

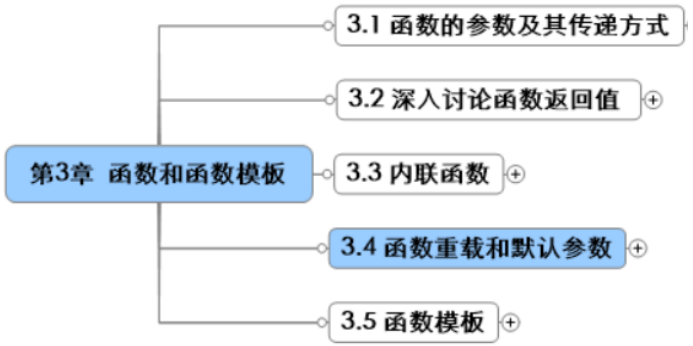
例：设计一个求4（不多于4）个整数的和的程序

1、不使用函数重载，也不使用默认参数时的代码

```
#include <iostream>
using namespace std;

int add1(int,int);                  //声明两个整数相加的函数add1
int add2(int,int,int);              //声明三个整数相加的函数add2
int add3(int,int,int,int);          //声明四个整数相加的函数add3

void main( )
{cout<<add1(1,3)<<","<<add2(1,3,5)<<","<<add3(1,3,5,7)<<endl;}
int add1(int m1,int m2){return m1+m2;}
int add2(int m1,int m2,int m3){return m1+m2+m3;}
int add3(int m1,int m2,int m3,int m4){return m1+m2+m3+m4;}
```

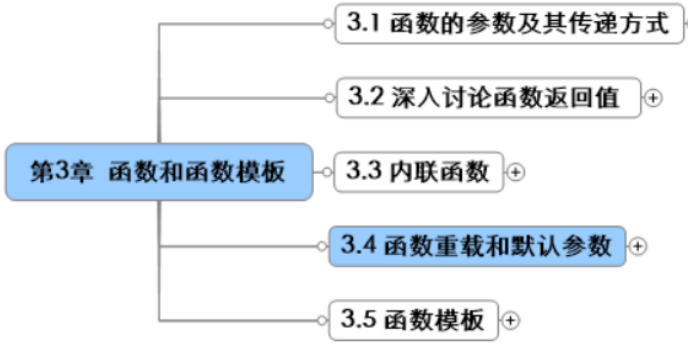
4,9,16

2、使用函数重载，但不使用默认参数时的代码




```
#include <iostream>
using namespace std;
int add(int,int);
int add(int,int,int);
int add(int,int,int,int);

void main( )
{cout<<add(1,3)<<","<<add(1,3,5)<<","<<add(1,3,5,7)<<endl;}
int add(int m1,int m2){return m1+m2;}
int add(int m1,int m2,int m3){return m1+m2+m3;}
int add(int m1,int m2,int m3,int m4){return m1+m2+m3+m4;}
```

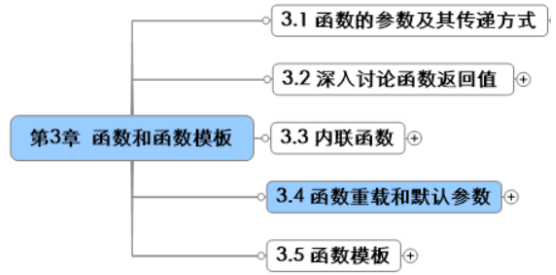
4,9,16



3、即使用函数重载，又使用默认参数时的代码

```
#include <iostream>
using namespace std;
int add(int m1=0,int m2=0,int m3=0,int m4=0)
{return m1 + m2 + m3 + m4;}

void main( )
{cout<<add(1,3)<<","<<add(1,3,5)<<","<<add(1,3,5,7)<<endl;}
```

4,9,16



使用默认参数之后的函数重载

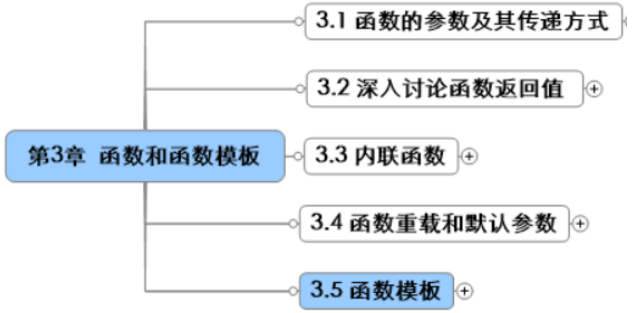
如果使用默认参数，就不能对参数个数少于默认参数个数的函数形态进行重载，只能对于多于默认参数个数的函数形态进行重载。

例如如下代码，默认参数个数为4个，则重载形态1的add函数时会出错，因为编译器决定不了是使用形态1、还是形态2的add函数。

```
#include <iostream>
using namespace std;
int add(int,int,int);                          //形态1
int add(int m1=0,int m2=0,int m3=0,int m4=0)    //形态2
{return m1 + m2 + m3 + m4;}

void main( )
{cout<<add(1,3)<<","<<add(1,3,5)<<","<<add(1,3,5,7)<<endl;}
int add(int m1,int m2,int m3){return m1+m2+m3;}
```

这段代码在执行到主函数main中语句“cout<<add(1,3,5)”时，因编译器不知道应该重载add函数的哪个形态，将会出错。

error C2668: 'add' : ambiguous call to overloaded function



§3.5 函数模板

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。

1、编制求两个数据最大值的函数模板程序

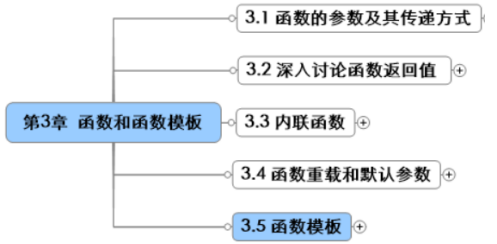
```
#include <iostream>
using namespace std;
template <class T>
T max(T m1,T m2) {return (m1>m2)?m1:m2;}
void main( )
{
    cout<<max(2,5)<<"\t"<<max(2.0,5.)<<"\t"
    <<max('w','a')<<"\t"<<max("ABC","ABD")<<endl;
}
```

5 5 w ABD

```
#include <iostream>
#include <complex>
#include <string>
using namespace std;
void printer(complex <int> );
void printer(complex <double> );
void main( ){
    int i(0);
    complex <int> num1(2,3);
    complex <double> num2(3.5,4.5);//用构造函数complex初始化num2并赋值
    printer(num1);
    printer(num2);
}
void printer(complex <int> a)
{
    string str1("real is "),str2="imag is ";
    cout<<str1<<a.real( )<<','<<str2<<a.imag( )<<endl;
}
void printer(complex <double> a)
{
    string str1("real is "),str2="imag is ";
    cout<<str1<<a.real( )<<','<<str2<<a.imag( )<<endl;
}
```

```
template <class T>
void printer(complex <T> a)
{
    string str1("real is "),str2="imag is ";
    cout<<str1<<a.real( )<<','<<str2<<
            a.imag( )<<endl;
}
```

```
real is 2,imag is 3
real is 3.5,imag is 4.5
```



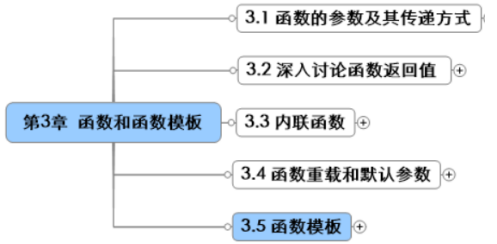
函数模板的参数及显式规则

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的话，会使人感到很厌烦，一般可使用下面的默认方式：函数模板名（参数列表），但这样的前提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。

如语句“cout<<max（6.5，8)<<endl;”，在没有定义max(double,int)形式的情况下，语句是无法执行的，这时就要显式的给出比较准则“cout<<max<double>(6.5,8)<<endl;”，或者对参数表中的参数进行强制转换，如“cout<<max（6.5，（double）8)<<endl;”，以便正确的由参数列表推断出模板参数。



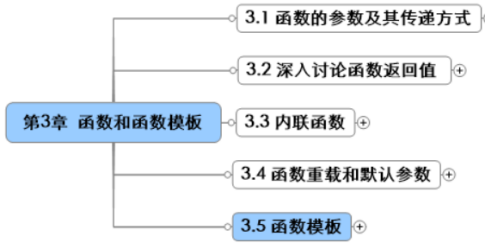
关键字typename和使用显式规则

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。

例：使用关键字typename和显式规则的实例

```
#include <iostream>
using namespace std;
template <typename T>                          //使用typename代替class
T max(T m1,T m2) {return (m1>m2)?m1:m2;}      //求二者最大值
template <typename T>                          //必须重写
T min(T m1,T m2) {return (m1<m2)?m1:m2;}      //求二者最小值
void main( )
{
    cout<<max("ABC","ABD")<<","<<min("ABC","ABD")<<","
    <<min ('W','T')<<","<<min(2.0,5.); //输出ABD，ABC，T，2
    cout<<"\t"<<min<double>(8.5,6)<<","<<min(8.5,(double)6)
    <<","<<max((int)8.5,6); //输出6，6，8
    cout<<"\t"<<min<int>(2.3,5.8)<<","<<max<int>('a','y')
    <<","<<max<char>(95,121)<<endl; //输出2，y，121
}
```

ABD,ABC,T,2 6,6,8 2,121,y



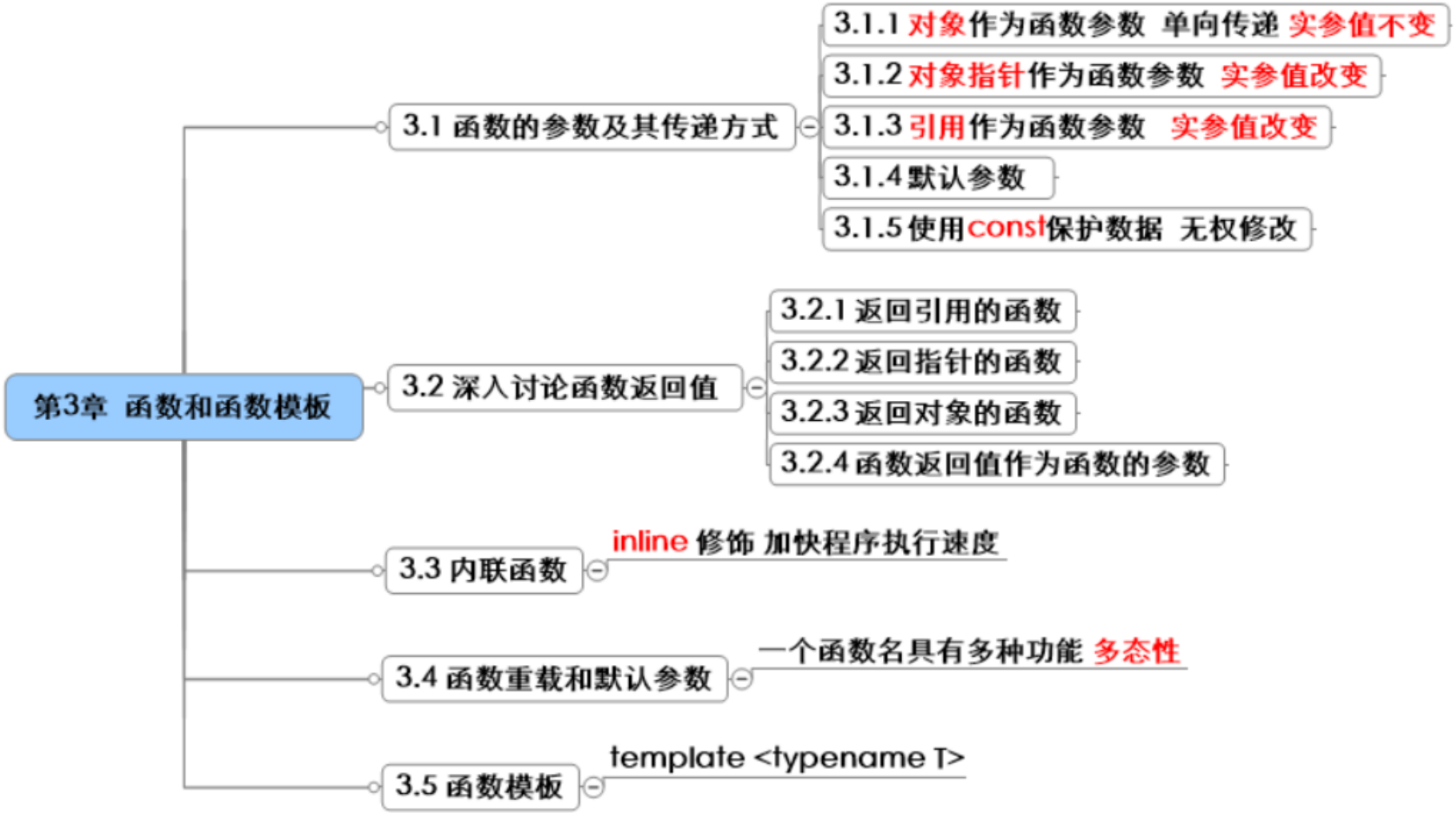
关键字typename和使用显式规则

此程序需要注意的地方主要是：

第一，定义函数模板时，函数模板一定要包含语句“template <typename T>”，不能因为前面有过该语句，在定义后面的函数模板时就将该句省略；

第二，如果参数列表中的参数类型不同，或参数类型相同，但需要转化参数类型时，一定要显式的给出比较准则，如“min<double>(8.5,6)”、“min<int>(2.3,5.8)”、“max<int>(‘a’,’y’)”、“max<char>(95,121)”，或者对参数表中的参数进行强制转换，如“cout<<max（8.5,（double）6)”、“max((int)8.5,6)”，以便程序能够正确的由参数列表推断出模板参数。

本章总结


第3章 函数和函数模板
3.1 函数的参数及其传递方式
3.1.1 对象作为函数参数 单向传递 实参值不变
3.1.2 对象指针作为函数参数 实参值改变
3.1.3 引用作为函数参数 实参值改变
3.1.4 默认参数
3.1.5 使用const保护数据 无权修改
3.2 深入讨论函数返回值
3.2.1 返回引用的函数
3.2.2 返回指针的函数
3.2.3 返回对象的函数
3.2.4 函数返回值作为函数的参数
3.3 内联函数
inline 修饰 加快程序执行速度
3.4 函数重载和默认参数
一个函数名具有多种功能 多态性
3.5 函数模板
template <typename T>

2017年10月 部分真题讲解

1. 按照标识符的要求,不能组成标识符的符号是

A. 连接符　　B. 下划线　　C. 大小写字母　　D. 数字字符

2. 下列输出语句中,正确的是

A. cout <<("%c\n","student")　　B. cout <<("%s\n","hello")

C. cout <<("%c\n","c")　　D. cout <<("%s\n",&a)

3. 已知:print( )函数是一个类的常成员函数,无返回值,下列表示中正确的是

A. void print( ) const　　B. void print(const)

C. void const print( )　　D. const void print( )

4. if 与 else 在使用过程中为避免嵌套出现二义性,C++ 中规定与 else 子句配对的是

A. 其之前最近的 if 语句　　B. 其之前最近且尚未配对的 if 语句

C. 缩排位置相同的 if 语句　　D. 其之后最近的 if 语句

5. 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. release

6. 逻辑运算符两侧运算对象的数据

A. 是逻辑型数据　　B. 只能是整型数据

C. 只能是整型或字符型数据　　D. 可以是任何类型的数据

8. 所谓数据封装就是将一组数据和与这组数据有关操作组装在一起，形成一个实体，这实体也就是

A. 类　　B. 对象　　C. 函数体　　D. 数据块

10. 在 C++ 中，函数原型不能标识

A. 函数的返回类型　　B. 函数参数的个数

C. 函数参数类型　　D. 函数的功能

11. 若二维数组 y 有 m 列，则位于 y[i][j]之前的元素数量是

A. j * m + i　　B. i * m + j

C. i * m + j - 1　　D. i * m + j + 1

12. 下列关于类的权限的描述错误的是

A. 类本身的成员函数只能访问自身的私有成员

B. 类的对象只能访问该类的公有成员

C. 普通函数不能直接访问类的公有成员，必须通过对象访问

D. 一个类可以将另一个类的对象作为成员

13. 下面不能够判断字符串 S 是空串的是

A. if (S[0] ==0)　　B. if(strlen(S) ==0)

C. if(strcmp(S,"") ==0)　　D. if(S == '\0')

14. 下列输出字符'd'的方法中,错误的是

A. cout << put('d')　　B. cout << 'd'

C. cout. put('d')　　D. char a = 'd'; cout << a;

15. 关于引用,下列的说法中错误的是

A. 引用是给被引用的变量取一个别名

B. 引用主要是用来作函数的形参和函数的返回值

C. 在声明引用时,要给它另开辟内存单元

D. 在声明引用时,必须同时使它初始化

16. 下面关于 C ++ 字符数组的叙述中，错误的是

A. 字符数组可以放字符串

B. 字符数组的字符可以整体输入、输出

C. 可以在赋值语句中通过赋值运算符“ = ”对字符数组整体赋值

D. 可以用关系运算符对字符数组比较大小

18. 非数组指针或引用型变量做实参时，它和对应虚参之间的数据传递方式是

A. 地址传递　　B. 单向值传递　　C. 双向值传递　　D. 由用户指定传递方式

25. cin 是输入流 istream 的一个对象，处理标准输入；_____是输出流 ostream 的一个对象，处理标准输出。

27. 设在程序中使用如下语句申请了一个对象数组：Point * ptr = new Point[2]；当要释放 ptr 指向的动态数组对象时，所使用的语句是 _____ 。

28. 书写程序语句时，适当增加空行和程序注释以增加程序的 _____ 。

29. C++ 语言中如果调用函数时，需要改变实参或者返回多个值，应该采取 _____ 方式。

35. 面向对象的四个基本特性是多态性、继承性、封装性、_____。

37. 若 int a = 8; int b = (++ a) ++; 则 b = _____。

39. 将指向对象的引用作为函数的形参，形参是对象的引用，实参是 _____。

三、改错题:本大题共5小题,

```
#include" stdafx. h"
#include  < iostream >
using namespace std;
class Test
{ private:
int x,y = 20;
public:
Test( int i,int j) { x = i,y = j;}
int getx( ) { return x;}
int gety( ) { return y;}
};
void main( )
{ Test mt(10,20);
cout << mt. getx( ) << endl;
cout << mt. gety( ) << endl;
}
```

```
#include <iostream>
using namespace std;
int main( ) {
const int num = 20;
int scores[ num ];
for( int i = 1 ;i < = num ;i ++ )
scores[ i ] = i;
return 0; }
```

```
#include <iostream.h>
class Aton
{int X,Y;
protected:
int zx,zy;
public:
void init(int i,int j){zx=i;zy=j;}
Aton(int i,int j,int n=0,int m=0)
{X=i,Y=j,zx=m,zy=n;
}
};
void main()
{Aton A(25,20,3,5);
A.init(5,9);
cout<<A.X<<endl;
cout<<A.Y<<endl;
}
```

```
#include <iostream>
using namespace std;
int main( )
{ int a = 10, b = 18, c = 77;
  const int *p = c;
  cout << *p << endl;
  return 0;
}
```

祝大家顺利通过考试！