C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×15题 = 15分

程序填空题　4分×5题 = 20分

程序分析题　6分×5题 = 30分

程序设计题　　2题 = 15分

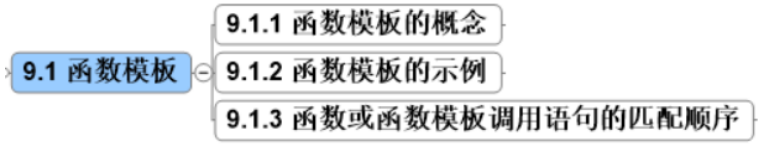
（第一题5分，第二题10分）

第九章　函数模板与类模板

本章主要内容

- 函数模板
- 类模板

9.1 函数模板



设计程序中的函数时，可能会遇到函数中参数的类型有差异，但需要实现的功能类似的情形。函数重载可以处理这种情形。重载函数的参数表中，可以写不同类型的参数，从而可以处理不同的情形。

9.1.1 函数模板的概念

为了提高效率，实现代码复用，C++提供了一种处理机制，即使用**函数模板**。函数在设计时并不使用实际的类型，而是使用虚拟的类型参数。这样可以不必为每种不同的类型都编写代码段。当用实际的类型来实例化这种函数时，将函数模板与某个具体数据类型连用。编译器将以函数模板为样板，生成一个函数，即产生了模板函数，这个过程称为函数模板实例化。函数模板实例化的过程由编译器完成。程序设计时并不给出相应数据的类型，编译时，由编译器根据实际的类型进行实例化。

课堂练习

C++中专门定义一个仅在模板中使用的参数类型的关键字是（　）。

A:const

B:inline

C:short

D:typename

课堂练习

C++中专门定义一个仅在模板中使用的参数类型的关键字是（　）。

A:const

B:inline

C:short

D:typename

答案：D

9.1.2 函数模板的示例

例9-1单项选择题

实现两个相同类型数加法的函数模板的声明可以是【 】

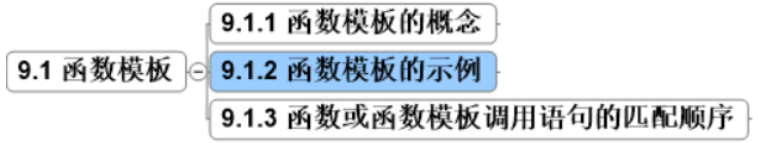
A.add(T x , T y)

B.T add(x, y)

C.T add(T x, y)

D.T add(T x,T y)

9.1.2 函数模板的示例



例9-1单项选择题

实现两个相同类型数加法的函数模板的声明可以是【 】

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

【分析】实现两个相同类型数的加法，返回值也应该和执行加法的操作数是同类型的。所以选项D是正确的。选项A中没有返回值类型，选项B中没有指定参数的类型，选项C中，参数y没有指定类型。

课堂练习

设有函数T Sum(T x，T y){return x+y；}，其中T为模板类型，则下列语句中对该函数错误的使用是（）

A:Sum(1，2)；

B:Sum(3.0，2.2)；

C:Sum(‘A’，‘C’)；

D:Sum("A“,"C")；

课堂练习

设有函数T Sum(T x，T y){return x+y；}，其中T为模板类型，则下列语句中对该函数错误的使用是（）

A:Sum(1，2)；

B:Sum(3.0，2.2)；

C:Sum(‘A’ ，‘C’)；

D:Sum("A “,"C")；

答案：D

"A"，"C"是两个字符串，两个字符串不能做加减运算

```
#include<iostream>
using namespace std;
template<typename T>
T abs(T x)
{
        return x<0?-x:x;
}
int main( )
{
        int n=-5;
        int m=10;
        double d=-.5;
        float f=3.2;
        cout<<n<<"的绝对值是:"<<abs(n)<<endl;
        cout<<m<<"的绝对值是:"<<abs(m)<<endl;
        cout<<d<<"的绝对值是:"<<abs(d)<<endl;
        cout<<f<<"的绝对值是:"<<abs(f)<<endl;
        return 0;
}
```

```
-5的绝对值是：5
10的绝对值是：10
-0.5的绝对值是：0.5
3.2的绝对值是：3.2
```

9.1.2 函数模板的示例

在主函数中，调用abs(n)时，编译器根据实参n的类型int，推导出模板中的类型参数T为int，然后实例化函数模板，生成函数模板abs的一个实例。

```
int abs (int x)
{
    return x < 0 ? -x : x;
}
```

这个实例即是模板函数。

当调用abs(d)时，根据实参d的类型double，又实例化一个新的函数。

```
double abs (double x)
{
    return x < 0 ? -x : x;
}
```

这是另一个模板函数。

实际上，函数模板不是一个具体的函数，编译器不能为其生成可执行代码。定义函数模板后只是一个对函数功能框架的描述，当它具体执行时，将根据传递的实际参数决定其功能。

9.1.2 函数模板的示例

虽然**函数模板**的使用形式与**函数**类似，但二者有本质的**区别**，主要表现在以下3个方面。

1)函数模板本身在编译时不会生成任何目标代码，只有当通过模板生成具体的函数实例时才会生成目标代码。

2)被多个源文件引用的函数模板，应当连同函数体一同放在头文件中，而不能像普通函数那样只将声明放在头文件中。

3)函数指针也只能指向模板的实例，而不能指向模板本身。

```
#include<iostream>
using namespace std;
template<class T>
void Swap(T &x,T &y)
{
    T tmp=x;
    x=y;
    y=tmp;
}
class myDate
{
public:
    myDate();
    myDate(int,int,int);
    void printDate()const;
private:
    int year,month,day;
};
myDate::myDate()
{
    year=1970;
    month=1;
    day=1;
}
myDate::myDate(int y,int m,int d)
{   year=y;
    month=m;
    day=d;
}
void myDate::printDate() const
{
    cout<<year<<"/"<<month<<"/"<<day<<endl;
    return;
}
int main()
{
    int n=1,m=2;
    Swap(n,m);          //编译器自动生成void Swap（int &,int &）函数
    cout<<n<<" "<<m<<endl;
    double f=1.2,g=2.3;
    Swap(f,g);          //编译器自动生成void Swap（double &,double &）函数
    cout<<f<<" "<<g<<endl;
    myDate d1,d2(2000,1,1);//创建两个对象
    Swap(d1,d2);      //编译器自动生成void Swap（myDate &,myDate &）函数
    d1.printDate();
    d2.printDate();
    return 0;
}
```

9.1.2 函数模板的示例

函数模板中还可以带多个类型参数。例如，下面这个函数模板的写法是合法的。

```
template <class T1, class T2>
void print(T1 arg1, T2 arg2)
{      cout<<arg1<<", "<<arg2<<endl;
}
```

9.1.3 函数或函数模板调用语句的匹配顺序

函数与函数模板也是允许重载的。在函数和函数模板名字相同的情况下，一条函数调用语句到底应该被匹配成对哪个函数或哪个模板的调用呢？C++编译器遵循以下先后顺序。

1)先找参数完全匹配的普通函数（不是由模板实例化得到的模板函数）。

2)再找参数完全匹配的模板函数。

3)然后找实参经过自动类型转换后能够匹配的普通函数。

4)如果上面的都找不到，则报错。

课堂练习

下列关于函数模板的描述中，正确的是（　）。

A:函数模板是一个实例函数

B:使用函数模板定义的函数没有返回类型

C:函数模板的类型参数与函数的参数相同

D:通过使用不同的类型参数，可以从函数模板得到不同的实例函数

课堂练习

下列关于函数模板的描述中，正确的是（　）。

A:函数模板是一个实例函数

B:使用函数模板定义的函数没有返回类型

C:函数模板的类型参数与函数的参数相同

D:通过使用不同的类型参数，可以从函数模板得到不同的实例函数

答案：D

课堂练习

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)

课堂练习

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

课堂练习

有关函数模版和模版函数说法错误的是（　）。

A:函数模版只是对函数的描述，编译器不为其产生任何执行代码，所以它不是一个实实在在的函数

B:模版函数是实实在在的函数，它由编译系统在遇到具体函数调用时所生成，并调用执行

C:函数模版需要实例化为模版函数后才能执行

D:当函数模版和一般函数同名时，系统先去匹配函数模版，将其实例化后进行调用

课堂练习

有关函数模版和模版函数说法错误的是（　）。

A:函数模版只是对函数的描述，编译器不为其产生任何执行代码，所以它不是一个实实在在的函数

B:模版函数是实实在在的函数，它由编译系统在遇到具体函数调用时所生成，并调用执行

C:函数模版需要实例化为模版函数后才能执行

D:当函数模版和一般函数同名时，系统先去匹配函数模版，将其实例化后进行调用

答案：D

9.2.1 类模板概念

通过类模板，可以实例化一个个的类。继承机制也是在一系列的类之间建立某种联系，这两种涉及多个类的机制是有很大差异的。类是相同类型事物的抽象，有继承关系的类可以具有不同的操作。而模板是不同类型的事物具有相同的操作，实例化后的类之间没有联系，相互独立。

9.2.1 类模板概念

声明类模板的一般格式如下：

```
template <模板参数表>
class 类模板名
{
        类体定义
};
```

其中，“模板参数表”的形式与函数模板中的“模板参数表”完全一样。类体定义与普通类的定义几乎相同，只是在它的成员变量和成员函数中通常要用到模板的类型参数。

类模板的成员函数既可以在类体内进行说明，也可以在类体外进行说明。如果在类体内定义，则自动成为内联函数。如果需要在类模板以外定义其成员函数，则要采用以下格式。

```
template <模板参数表>
返回类型名 类模板名<模板参数标识符列表>::成员函数名(参数表)
{
    函数体
}
```

类模板声明本身并不是一个类，它说明了类的一个家族。只有当被其他代码引用时，模板才根据引用的需要生成具体的类。

9.2.1 类模板概念

不能使用类模板来直接生成对象，因为类型参数是不确定的，必须先为模板参数指定“实参”，即模板要“实例化”后，才可以创建对象。也就是说，当使用类模板创建对象时，要随类模板名给出对应于类型形参或普通形参的具体实参，格式如下：

类模板名 <模板参数表> 对象名1,...,对象名n;

或是

类模板名 <模板参数表> 对象名1(构造函数实参),...,对象名构造函数实参);

编译器由类模板生成类的过程称为类模板的实例化。由类模板实例化得到的类称为模板类。

要注意的是，与类型形参相对应的实参是类型名。

课堂练习

下列有关模板的描述中，错误的是（　）。

A:模板把数据类型作为一个设计参数，称为参数化程序设计

B:使用时，模板参数与函数参数相同，是按位置而不是名称对应的

C:模板实例化参数类型包括数据类型和值

D:类模板与模板类是同一个概念

课堂练习

下列有关模板的描述中，错误的是（ ）。

A:模板把数据类型作为一个设计参数，称为参数化程序设计

B:使用时，模板参数与函数参数相同，是按位置而不是名称对应的

C:模板实例化参数类型包括数据类型和值

D:类模板与模板类是同一个概念

答案：D

课堂练习

允许用户为类定义一种模式，使得类中的某些数据成员及某些成员函数的返回值能取任意类型，这是一个（ ）。

A:类模板

B:模板类

C:函数模板

D:模板函数

课堂练习

允许用户为类定义一种模式，使得类中的某些数据成员及某些成员函数的返回值能取任意类型，这是一个（ ）。

A:类模板

B:模板类

C:函数模板

D:模板函数

答案：A

9.2.2 类模板示例

二元组是常用的一种结构。可以定义两个值的二元组，如平面坐标系下点的横、纵坐标组成的二元组。还可以定义两个字符串的二元组，如字典中单词与释义组成的二元组。还可以定义学生姓名及其成绩的二元组。二元组的例子非常多，不胜枚举。

如果要定义二元组的类，则需要根据组成二元组的类型定义很多不同的类。现在可以使用类模板来解决问题。

```
#include<iostream>
using namespace std;
template<class T>
class TestClass
{
public:
    T buffer[10];
    T getData(int j);
};
template<class T>
T TestClass<T>::getData(int j)
{
    return *(buffer+j);
};
```

```
int main()
{
    TestClass<char>ClassInstA;   //char取代T，从而实例化为一个具体的类
    int i;
    char cArr[6]="abcde";
    for(i=0;i<5;i++)
        ClassInstA.buffer[i]=cArr[i];
    for(i=0;i<5;i++)
    {   char res=ClassInstA.getData(i);
        cout<<res<<" ";
    }
    cout<<endl;
    TestClass<double>ClassInstF;   //实例化为另外一个具体的类
    double fArr[6]={12.1,23.2,34.3,45.4,56.5,67.6};
    for(i=0;i<6;i++)
        ClassInstF.buffer[i]=fArr[i]-10;
    for(i=0;i<6;i++)
    {   double res=ClassInstF.getData(i);
        cout<<res<<" ";
    }
    cout<<endl;
    return 0;
}
```

```
a b c d e
2.1 13.2 24.3 35.4 46.5 57.6
```

```
#include<iostream>
using namespace std;
template<int i>
class TestClass
{public:
    int buffer[i];          //使buffer的大小可变化，但其类型则固定为int
    int getData(int j);
};
template<int i>
int TestClass<i>::getData(int j)
{
    return *(buffer+j);
};
int main()
{   TestClass<6>ClassInstF;
    int i;
    double fArr[6]={12.1,23.2,34.3,45.4,56.5,67.6};
    for(i=0;i<6;i++)
        ClassInstF.buffer[i]=fArr[i]-10;
    for(i=0;i<6;i++)
    {   double res=ClassInstF.getData(i);   cout<<res<<" ";}
    cout<<endl;
}
```

2 13 24 35 46 57

课堂练习

关于类模板的说法正确的是（　）。

A:类模板的主要作用是生成抽象类

B:类模板实例化时，编译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具有同样类型

D:类模板中的成员函数没有返回值

课堂练习

关于类模板的说法正确的是（　）。

A:类模板的主要作用是生成抽象类

B:类模板实例化时，编译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具有同样类型

D:类模板中的成员函数没有返回值

答案：B

课堂练习

用类模板定义对象的一般格式正确的是（　）。

A:类名<模板实例化参数类型> 对象名　(构造函数实参列表)；

B:类名<构造函数实参列表> 对象名；

C:类名 对象名　(<模板实例化参数类型>构造函数实参列表)；

D:类名 对象名　(构造函数实参列表)；

课堂练习

用类模板定义对象的一般格式正确的是（　）。

A:类名<模板实例化参数类型> 对象名 （构造函数实参列表）；

B:类名<构造函数实参列表> 对象名；

C:类名 对象名 （<模板实例化参数类型>构造函数实参列表）；

D:类名 对象名 （构造函数实参列表）；

答案：A

课堂练习

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

课堂练习

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

课堂练习

类模板template<class T> class?({...}；，其中友元函数f对特定类型T(如int)，使函数f(X<int>&)成为X<hat>模板类的友元，则其说明应为（　）。

A: friend void( );

B:friend void f(X<T>&);

C:friend void A::f();

D:friend void C(T);

课堂练习

类模板template<class T> class?({...}；，其中友元函数f对特定类型T(如int)，使函数f(X<int>&)成为X<hat>模板类的友元，则其说明应为（　）。

A: friend void( );

B:friend void f(X<T>&);

C:friend void A::f();

D:friend void C(T);

答案：B

9.2.3 类模板与继承

类之间允许继承，类模板之间也允许继承。具体来说，类模板和类模板之间、类模板和类之间可以互相继承，它们之间的常见派生关系有以下4种情况：

1)普通类继承模板类。

2)类模板继承普通类。

3)类模板继承类模板。

4)类模板继承模板类。

根据类模板实例化的类即是模板类。

9.2.3 类模板与继承

```
#include<iostream>
using namespace std;
template<class T>          //类模板，基类
class TBase
{
      T data;
public:
      void print()
      {
            cout<<data<<endl;
      }
};
class Derived:public TBase<int>   //从模板继承，普通类
{
};
int main()
{
     Derived d;             //普通派生类的对象
     d.print();             //调用类模板中的成员函数
     return 0;
}
```

```
#include<iostream>
#include<string>
using namespace std;
class TBase
{     int k;
public:
      void print( )
      {   cout<<"TBase::"<<k<<endl;  }
};
template<class T>
class TDerived:public TBase
{     T data;
public:
       void setData(T x)
       {      data=x;    }
       void print( )
       {      TBase::print();    cout<<"TDerived::"<<data<<endl;}
};
int main( )
{
       TDerived<string>d;
       d.setData("2019");
       d.print();
}
```

```
TBase::-858993460
TDerived::2019
```

```
#include<iostream>
#include<string>
using namespace std;
template<class T>
class TBase
{     T data1;
public:
      void print( )
      {   cout<<"TBase::"<<data1<<endl;  }
};
template<class T1,class T2>
class TDerived:public TBase<T1>
{
      T2 data2;
public:
      void print( )
      {   TBase<T1>::print();   cout<<"TDerived::"<<data2<<endl;  }
};
void main( )
{
      TDerived<int,int>d;  //类模板实例化，并声明对象
      d.print();
      TDerived<string,string>d2;
      d2.print();
}
```

```
TBase::-858993460
TDerived::-858993460
TBase::
TDerived::
```

```
#include<iostream>
#include<string>
using namespace std;
template<class T>
class TBase
{public:
        T data1;
public:
        void print( )
        {
                cout<<"TBase::"<<data1<<endl;
        }
};
template<class T1,class T2>
class TDerived:public TBase<T1>
{
public:
        T2 data2;
public:
        void print( )
        {
                TBase<T1>::print();
                cout<<"TDerived::"<<data2<<endl;
        }
};
void main( )
{
        TDerived<int,int>d;  //类模板实例化，并声明对象
        d.data1=5;d.data2=8;
        d.print();
        TDerived<string,string>d2;
        d2.data1="happy";d2.data2="new year";
        d2.print();
        TDerived<int,string>d1;
        d1.data1=2020;d1.data2="good luck";
        d1.print();
 }
```

```
TBase::5
TDerived::8
TBase::happy
TDerived::new year
TBase::2020
TDerived::good luck
```

```
#include<iostream>
#include<string>
using namespace std;
template<class T>
class TBase   //类模板
{
        T data1;
public:
        void print( )
        {
              cout<<"TBase::"<<data1<<endl;
        }
};
template<class T2>
class TDerived:public TBase<int> //类模板继承于模板类
{
        T2 data2;
public:
        void print( )
        {
              TBase<int>::print( );
               cout<<"TDerived::"<<data2<<endl;
        }
};
int main( )
{
      TDerived<int>d;
      d.print( );
      TDerived<string>d2;
      d2.print( );
}
```

```
TBase::-858993460
TDerived::-858993460
TBase::-858993460
TDerived::
```

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

template<class T>
class TBase   //类模板
{
public:
     T data1;
     void print( )
    {
          cout<<"TBase::"<<data1<<endl;
    }
};
template<class T2>
class TDerived:public TBase<int> //类模板继承于模板类
{
public:
     T2 data2;
    void print()
    {
          TBase<int>::print( );
           cout<<"TDerived::"<<data2<<endl;
    }
};
int main()
{
      TDerived<int>d;
      d.data1=5;d.data2=8;
      d.print( );
      TDerived<string>d2;
      d2.data1=2020;d2.data2="good luck";
      d2.print( );
}
```

```
TBase::5
TDerived::8
TBase::2020
TDerived::good luck
```

课堂练习

函数模板template<typename T> void Func(T，T)可具有下列哪种实例化形式（　　）。

A:void Func(float，int)

B:void Func(char，char)

C:void Func(int，double)

D:void Func(bool，float)

课堂练习

函数模板template<typename T> void Func(T，T)可具有下列哪种实例化形式（　　）。

A:void Func(float，int)

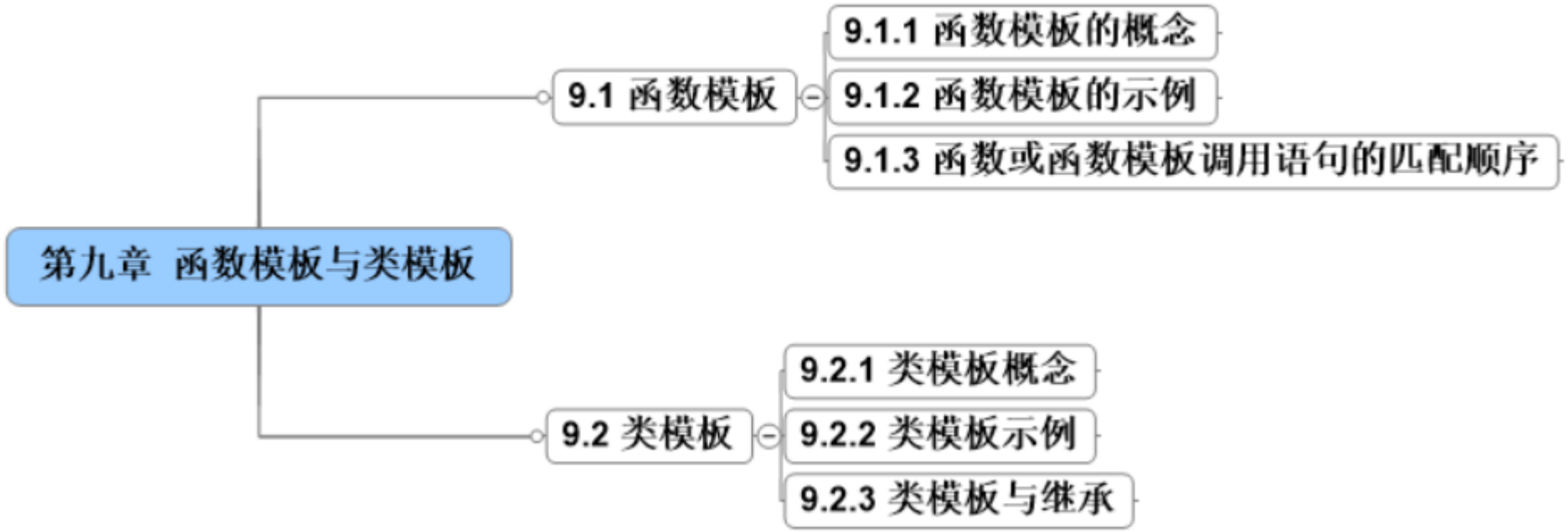
B:void Func(char，char)

C:void Func(int，double)

D:void Func(bool，float)

答案：B

本章总结


第九章 函数模板与类模板
9.1 函数模板
9.1.1 函数模板的概念
9.1.2 函数模板的示例
9.1.3 函数或函数模板调用语句的匹配顺序
9.2 类模板
9.2.1 类模板概念
9.2.2 类模板示例
9.2.3 类模板与继承

祝大家顺利通过考试！