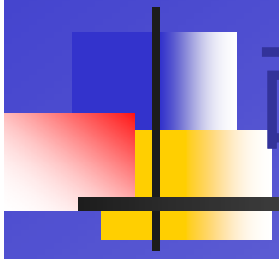
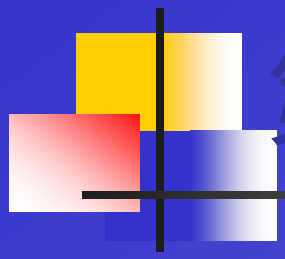


面向对象的程序设计

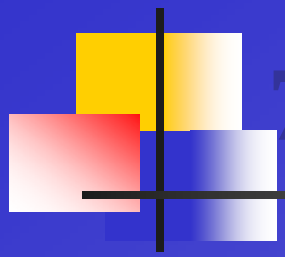
华中科技大学计算机学院

李瑞轩



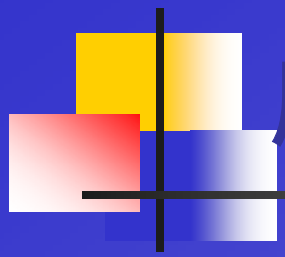
第7章 虚函数

- 7.1 虚函数
- 7.2 虚析构函数
- 7.3 抽象类
- 7.4 友元、绑定
- 7.5 类的存储空间



7.1 虚函数

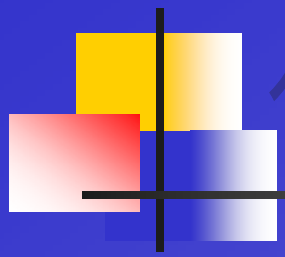
- 虚函数：是一种动态多态函数，通过动态绑定完成虚函数的调用。（静态多态与静态绑定）
  - “单界面，多实现版本”的思想
- 虚函数到对象的函数成员的映射通过存储在对象中的一个指针完成。（虚函数入口地址表VFT）
- 定义：
  - virtual 函数原型　　【看例7.1】



虚函数使用说明1

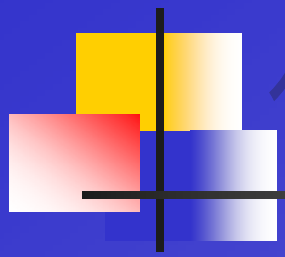
1. 虚函数必须是类的成员函数，非成员函数不能说明为虚函数
2. 虚函数一般是基类的public或protected部分函数，可在一个或多个public派生类中重新定义（即，可有不同的实现版本）

   函数原型必须完全相同
3. 虚函数只有在具有继承关系的类层次结构中定义才有意义，否则引起额外开销（通过VFT访问）
4. 虚函数有隐含的this参数，参数表后可出现const和volatile



使用说明2

5. 用虚函数实现程序运行时，必须用指向基类(父类)的指针(或引用)访问虚函数。根据父类指针指向的对象类型不同，动态绑定调用相应对象不同版本的虚函数，实现表现不同行为的操作，这就是“虚函数根据对象类型表现出的多态性”。【分析例7.1】
6. 一旦在基类中定义为虚函数，则所有后续派生类中原型相同的no-static函数将自动成为虚函数，即使没有“virtual”声明（无限传递性）【分析例7.2】
7. 虚函数同普通函数成员一样，可以声明为inline 函数，也可以重载、缺省和省略参数。【分析例7.3】



使用说明3

8. 下列函数成员不能定义为有this参数的虚函数：
   - 静态函数成员（没有this 指针）
   - 构造函数（构造时对象类型是确定的，不需根据类型表现出多态性）
9. 析构函数可通过基(父)类指针(引用)调用，而父类指针指向的对象类型是不确定的，因此，析构函数可定义为虚函数，以便必要时表现出多态特性。如delete *f调用析构函数。

使用说明4

10. 虚函数只能定义为其他类的友元，而不能定义为当前类 的友元(友元非当前类的成员)。即不能同时用virtual 和friend 定义函数。
11. 虚函数能根据对象类型适当地绑定函数成员，且绑定函数成员的效率非常之高，因此，最好将普通函数成员全部定义为虚函数。
12. 注意：虚函数主要通过基类和派生类对象表现出多态特性，由于union既不能定义基类又不能定义派生类，故不能在union中定义虚函数。 转7.2

【例7.1】定义父类POINT 和子类CIRCLE 的绘图函数成员show

```
#include <iostream.h>
class POINT{
    int  x, y;
public:
    int getx( ) { return x; }
    int gety( ) { return y; }
    virtual void show( ) { cout<<"Show a point\n"; }
    POINT(int x, int y) { POINT::x=x; POINT::y=y; }
};
class CIRCLE: public POINT{
    int r;
public:
    int getr( ) { return r; }
    void show( ) { cout<<"Show a circle\n"; }
    CIRCLE(int x, int y, int r):POINT(x, y) { CIRCLE::r=r; }
};
```

Return定义

```
void main(void)
{
   CIRCLE c(3, 7, 8);
   POINT *p=&c;      //父指针p实际指向的是子类Circle对象
   cout<<"The circle with radius "<<c.getr( );
   cout<<" is at ("<<p->getx( )<<", "<<p->gety( )<<")\n";
   p->show( );          //p->show( ) 动态绑定，并调用相应的虚函数
}
```

输出：

```
The circle with radius 8 is at (3, 7)
Show a circle
```

考虑：若去掉virtual，结果如何？

Return说明2

【例7.2】虚函数的使用方法

```
#include <iostream.h>
struct A{
   virtual void f1( ){ cout<<"A::f1\n"; }
   virtual void f2( ){ cout<<"A::f2\n"; }
   virtual void f3( ){ cout<<"A::f3\n"; }
   virtual void f4( ){ cout<<"A::f4\n"; }
};
class B: A{
   virtual void f1( )    //virtual可省略
    {    cout<<"B::f1\n";     }
   void f2( )      //f2自动成为虚函数
    {    cout<<"B::f2\n";     }
};
class C: B{
   void f4( )    //f4自动成为虚函数
   {    cout<<"C::f4\n";     }
};
```

```
void main(void)
{
   C  c;
   A *p=(A *)&c;
   p->f1( );      //调用B::f1( )
   p->f2( );      //调用B::f2( )
   p->f3( );      //调用A::f3( )
   p->f4( );      //调用C::f4( )
   p->A::f2( );//调用A::f2( )
}
```

输出：B::f1
B::f2
A::f3
C::f4
A::f2

语法检查静态进行，考虑(对否)：
c.f1( ); //?
c.f2( ); //?

Return说明2

【例7.3】虚函数的重载方法

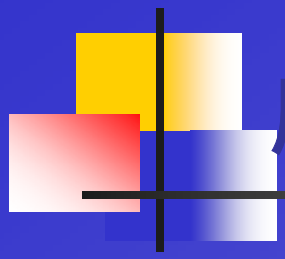
```
#include <iostream.h>
struct A{
   virtual void f1( ) { cout<<"A0"; }
   virtual void f1(char c) { cout<<"A1"; }
   void f1(int x) { cout<<"A2"; }
};
class B: A{
   void f1( ) { cout<<"B0"; }  //自动成为虚函数
   void f1(int x) { cout<<"B2";  }//普通函数成员
};
class C: B{
   void f1( ) { cout<<"C0"; }   //自动成为虚函数
   void f1(char c) { cout<<"C1"; } //自动成为虚函数
public:
   void f1(long x) { cout<<"C3"; } //普通函数成员
};
```

```
void main(void)
{
   C c;
   A *p=(A *)&c;
   p->f1('X');   //调用C::f1(char)
   p->f1( );     //调用C::f1( )
   c.f1(23L);    //调用C::f1(long)
   p->f1(3)      //调用A::f1(int)
}
```

输出：C1C0C3A2

考虑：

1. c.f1(3); // 对否
2. 若去掉A::f1(int)
   p->f1(3) //输出？



虚函数与重载成员函数的区别

两者貌似相同，但有本质区别：

1. 重载使用静态联编（绑定）机制；虚函数采用动态联编机制。
2. 对于父类A中声明的虚函数f( )，若在子类B中重定义f( )，必须确保子类B::f( )与父类A::f( )具有完全相同的函数原型，才能覆盖原虚函数f( )而产生虚特性，执行动态联编机制。否则，只要有一个参数不同，编译系统就认为它是一个全新的函数，而不实现动态联编。

Return说明2

7.2 虚析构函数

- 如果基类的析构函数定义为虚析构函数，则派生类的析构函数就会自动成为虚析构函数。
- 注意：如果为基类和派生类的对象分配了动态内存，或者为派生类的对象成员分配了动态内存，则一定要将基类和派生类的析构函数定义为虚析构函数，否则便可能造成内存泄漏，导致操作系统出现内存保护错误。【分析例7.4】
- 最好将所有的析构函数都定义为虚析构函数，以便delete运算调用适合对象类型的析构函数，释放对象所分配的所有动态内存。 转补充说明

【例7.4】输入职员的花名册,如果职员的姓名、编号和年龄等信息齐全,则登记该职员的个人信息,否则只登记职员的姓名 。

```
#include <stdio.h>
#include <string.h>
class STRING{
    char *str;
public:
    STRING(char *s);
    virtual ~STRING( )
    {
        if (str) { delete str; str=0; }
    }
};
STRING::STRING(char *s)
{
    str=new char[strlen(s)+1];
    strcpy(str, s);
}
```

```
class CLERK: public STRING{
    STRING clkid;
    int age;
public:
    CLERK(char *n, char *i, int a);
    ~CLERK( ){ } //自动调用clkid.~STRING( )和STRING::~STRING( )
};
CLERK::CLERK(char *n, char *i, int a):STRING(n), clkid(i)
{
    age=a;
}
```

```
const  int  max=10;
void main(void)
{
    STRING *s[max];
    int a, k, m;
    char n[12], i[12], t[256];
    printf("Please input name, number and age:\n");
    for(k=0; k<max; k++) {
        gets(t);
        m=sscanf(t,“%8s %8s %d”,n,i,&a)!=3; //m记录信息是否齐全
        s[k]=m?new STRING(n):new CLERK(n,i,a);
    }
    for(k=0; k<max; k++)
        delete s[k];  //若s[k]指向STRING的对象，则用s[k]->~STRING( )析构
                      //若s[k]指向CLERK的对象，则用s[k]->~CLERK( )析构
}
```

若STRING类没有定义虚析构函数，结果如何？

Return 7.2

补充说明

- 如果用指向父类的引用实现动态多态性，需要注意：若被引用的对象自身不能析构，如，被引用的对象是通过new生成的，那么被引用的对象就必须用delete &析构。如：

```
STRING  &z=*new CLERK("zang","982021",23);
delete &z;    //析构对象z并释放对象z占用的内存
```

上述delete &z完成了两个任务：

①调用析构函数析构CLERK（“zang”，“982021”，23），释放基类和对象成员各自为字符指针str分配的空间；

②释放CLERK（"zang"，"982021"，23）对象占用的存储空间.

如果将上述delete &z改为z.~CLERK( )，则只完成任务①而没完成任务②；如果改为free(&z)，则只完成任务②而没完成任务①。因此，只有delete &z才是唯一正确的用法。

7.3 抽象类

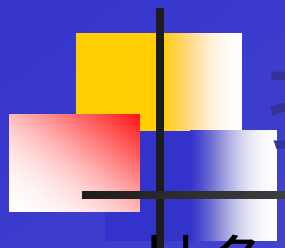
- 纯虚函数：不必定义函数体的特殊虚函数(可以重载，缺省参数，省略参数，内联等，相当于Java的interface)
  - 定义格式：virtual 函数原型=0;（即，函数体=0 ）
  - 纯虚函数有this参数，故不能定义为静态函数成员.
  - 构造函数不能定义为虚函数，故不能定义为纯虚函数；
  - 析构函数可以定义为虚函数，故可定义为纯虚函数。
- 抽象类：含有纯虚函数的类
  - 抽象类常用作派生类的基类，不应该有对象或类实例(相当于Java的interface)
  - 如果派生类继承了抽象类的纯虚函数，却没有重新定义之，或者派生类定义了基类所没有定义的纯虚函数，则派生类就会自动成为抽象类。在多级派生的过程中，如果到某个派生类为止，所有纯虚函数都已全部重新定义，则该派生类就会成为非抽象类。【例7.5】

【例7.5】多级派生中的抽象类与非抽象类用法

```
#include <iostream.h>
struct A{              //A被定义为抽象类
   virtual void f1( )=0;
   virtual void f2( )=0;
};
void A::f1( ){ cout<<"A1"; }
void A::f2( ){ cout<<"A2"; }
class B: public A{     //重新定义f2，未定义f1，B为抽象类
   void f2( ) { this->A::f2( ); cout<<"B2"; }
};
class C: public B{     // f1和f2均重新定义，C为非抽象类
   void f1( ) { cout<<"C1"; }
};
```

```
void main(void)
{
   C c;
   A *p=(A *)&c;
   p->f1( );   //调用C::f1( )
   p->f2( );   //调用B::f2( )
}
```

输出：C1A2B2

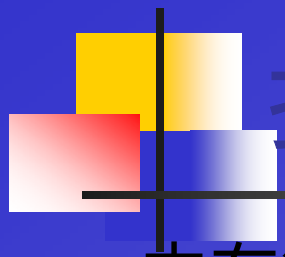


抽象类使用说明1

- 抽象类不能定义或产生任何对象，包括用new创建对象以及用作函数参数的类型和定义函数返回对象（原因：纯虚函数可以不定义函数体, 产生的对象要调用纯虚函数怎么办？）
- 抽象类可作派生类的父类，若定义相应的父类引用和指针，就可以引用或指向（抽象类的）非抽象子类对象。
- 通过抽象类指针和引用可以调用抽象类的纯虚函数，此时调用的应该是抽象类的非抽象子类的虚函数（由纯虚函数和虚函数的多态特性所决定）。如果抽象类的非抽象子类没有重新定义这样的虚函数，就会导致程序出现不可意料的运行错误。调用抽象类的普通函数成员不会出现不可意料的运行错误。
- 【分析例7.6】

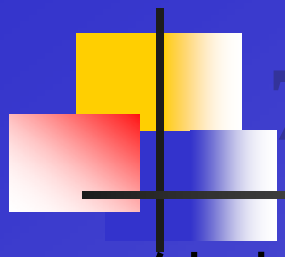
【例7.6】本例说明抽象类不能产生对象

```
#include <iostream.h>
struct A{              //定义类A为抽象类
   virtual void f1( )=0;
   void f2( ) { };
};
struct B: A{           //定义抽象类A的非抽象子类
   void f1( ){ };
};
A  f( );               // ×，返回类A意味着抽象类要产生一个对象
int g(A  x);           // ×，调用时要传递一个类A的对象
A &h(A &y); //√，可以引用非抽象子类B的对象
void main(void)
{
   A a;                //×，抽象类不能产生对象a
   A *p;               //√，可以指向非抽象子类B的对象
   p->f1( );           //×，运行时无A::f1( )或其他派生类函数
   p->f2( );           //√，调用A::f2( )
}
```



抽象类使用说明2

- 内存管理函数malloc可以为抽象类分配空间，但不调用构造函数构造抽象类的对象，因此，内存管理函数malloc实质上不产生任何抽象类对象。只有成功地产生了某个类的对象，才能通过抽象类指针或引用调用这个类的虚函数。
- 抽象类作为抽象级别最高的类，主要用于定义派生类共有的数据和函数成员。抽象类的纯虚函数没有函数体，意味着目前尚无法描述该函数的功能。例如，如果图形是点、线和圆等类的抽象类，那么抽象类的绘图函数就无法绘出具体的图形 。<看教材【例7.7】>



7.4 友元、绑定

- 纯虚函数和虚函数都能定义成另一个类的成员友元。由于纯虚函数一般不会定义函数体，故纯虚函数一般不要定义为其他类的成员友元。
- 如果类A的函数成员f定义为类B的友元，那么f就可以访问类B的所有成员，但是，f并不能访问从类B派生的类C的所有成员，除非f也定义为类C的友元或者类A就是类C。（即友元对派生不具备传递性）

【例7.8】说明纯虚函数和虚函数定义为友元的用法

```
#include <iostream.h>
class C;
struct A {
   virtual void f1(C &c)=0;
   virtual void f2(C &c);
};
class B: A{
public:
   void f1(C &c);   //f1自动成为虚函数
};
class C {
   char c;
  //允许但无意义，A::f1(C &c)无函数体
   friend void A::f1(C &c);
   friend void A::f2(C &c);
public:
   C(char c) { C::c=c; }
};
```

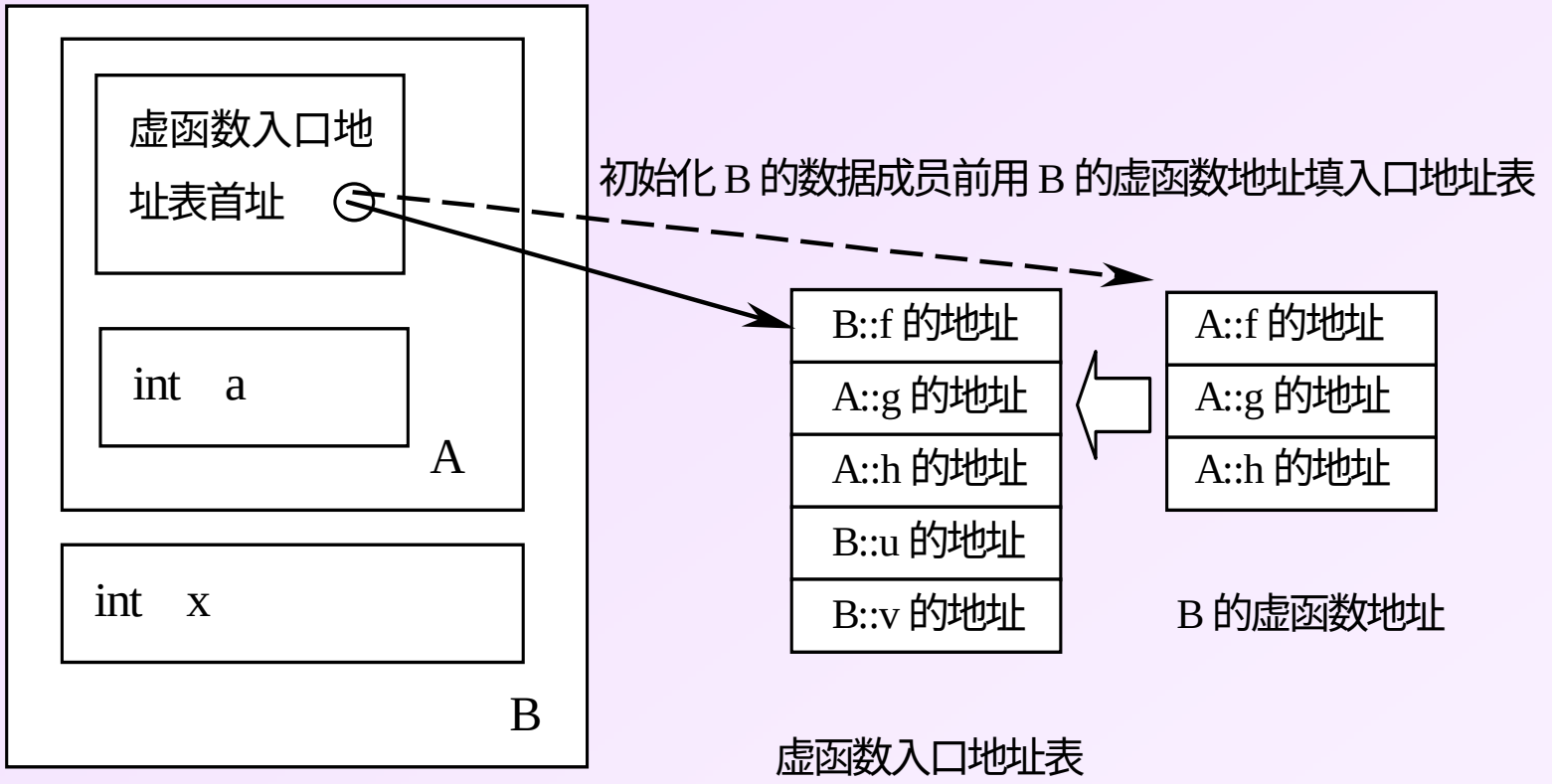
```
void A::f1(C &c)
    { cout<<"B outputs "<<c.c<<"\n"; }
void A::f2(C &c)
    { cout<<"A outputs "<<c.c<<"\n"; }
void B::f1(C &c)
    { cout<<c.c;  }     //×，B::f1不是C的
                        友元，不能访问c.c
void main( void)
{
   B b;
   C c('C');
   A *p=(A *) new B;
   p->f1(c);          //调用B::f1( )
   p->f2(c);          //调用A::f2( )
}
```

7.5 类的存储空间

- 单继承派生类的存储空间由基类和派生类的非静态数据成员构成。当基类或派生类包含虚函数或纯虚函数时，派生类的存储空间还包括虚函数地址表首址所占的存储单元(VFT指针)。
- 如果基类定义了虚函数或者纯虚函数，则派生类对象就将其起始单元作为共享VFT指针，用于存放基类和派生类的虚函数地址表首址。派生类的构造函数和析构函数会选择合适时机，在共享VFT指针中存放派生类和基类的虚函数地址表首址，由此首址或指针实现动态绑定和多态。分析【例7.10】
- 如果基类没有定义虚函数，而单继承派生类定义了虚函数，则单继承派生类的存储空间由三个部分组成：第一部分为基类存储空间，第二部分为派生类虚函数入口地址表首址，第三部分为该派生类新定义的数据成员。分析【例7.11】

【例7.10】计算单继承派生类存储空间的方法

```
#include <iostream.h>
class A{
    static int  b;
    int  a;
    virtual int  f( );
    virtual int  g( );
    virtual int  h( );
};
class B: A{
    static int y;
    int x;
    int f( );
    virtual int u( );
    virtual int v( );
};
void main(void)
{
    cout<<"Size of int="<<sizeof(int)<<"\n";
    cout<<"Size of pointer="<<sizeof(void *)<<"\n";
    cout<<"Size of A="<<sizeof(A)<<"\n";
    cout<<"Size of B="<<sizeof(B)<<"\n";
}
```

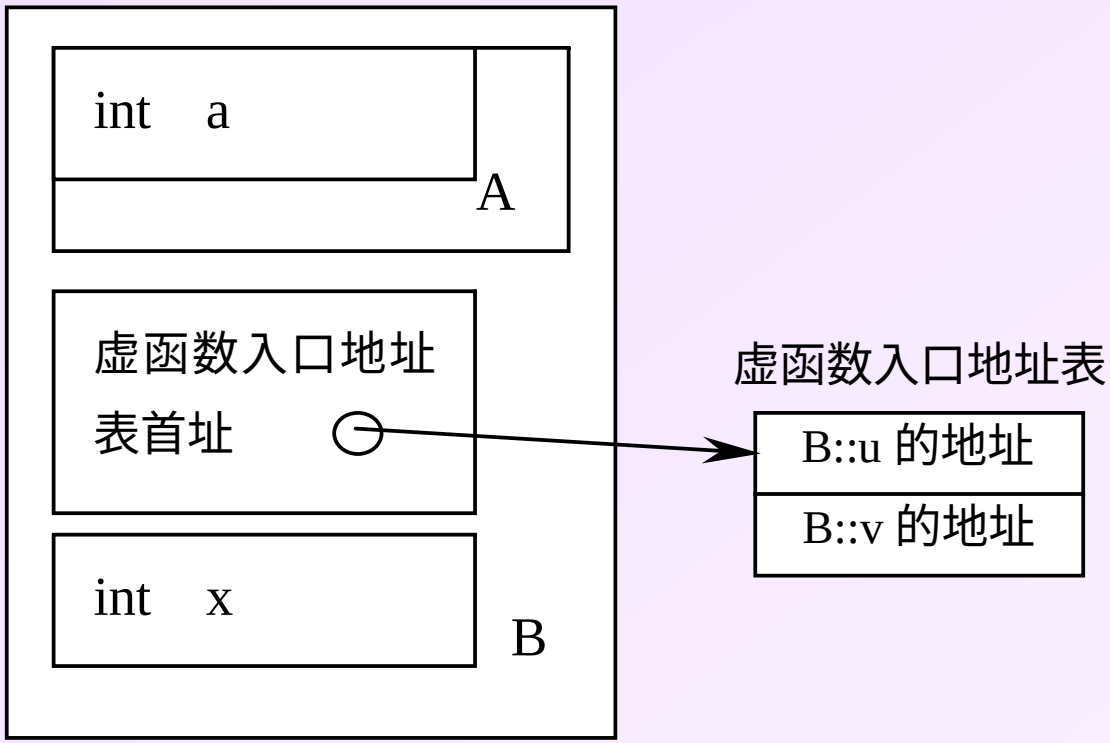


sizeof(A)＝sizeof(a)＋sizeof(void *)，
sizeof(B)＝sizeof(A)＋sizeof(x)

return

【例7.11】当基类没有虚函数时计算派生类存储空间的方法

```
#include <iostream.h>
class A{
    static int  b;
int  a;
};
class B: A{
    static int y;
    int x;
public:
    virtual void u( ) { };
    virtual void v( ) { };
};
void main(void)
{
    cout<<"Size of int="<<sizeof(int)<<"\n";
    cout<<"Size of pointer="<<sizeof(void *)<<"\n";
    cout<<"Size of A="<<sizeof(A)<<"\n";
    cout<<"Size of B="<<sizeof(B)<<"\n";
}
```



sizeof(A)＝sizeof(a)，sizeof(B)＝sizeof(A)＋sizeof(void *)＋sizeof(x)