1. 空类的长度为1.编译器会为空类加一个隐藏字节，当空类进行实例化时，其对象可以占据一个内存地址。

例：程序class\_test42

例：

class A

{

};

cout << "空类A的大小为：" << sizeof(A) << endl;

1. 类的非虚成员函数和静态变量均不会增加类的大小。

class B

{

public:

void func()

{

}

static void f()

{

}

private:

static int a;

// int b;

};

int B::a = 0;

// 类的非虚成员函数和静态变量均不会增加类的大小

cout << "类B的大小为：" << sizeof(B) << endl; // 大小为1

1. 一般情况下，类的长度的大小和结构体长度的大小类似。

class C

{

private:

char c;

int a;

double f;

};

// 一般情况下类的长度和结构体的大小计算类似

// 参考C语言学习-sizeof的用法与结构体的长度计算.docx

cout << "类C的大小为：" << sizeof(C) << endl; // 大小为16

1. 类中含有虚函数（或纯虚函数）

class D

{

public:

virtual void func()

{

}

virtual void h()

{

}

private:

char i;

int a;

double f;

};

cout << "类D的大小为：" << sizeof(D) << endl; // 大小为24

当类中含有虚函数，因为类对象的内存需要存放一个虚函数表，这是一个指针变量，为vptr。所以相当于类中增加了一个大小为4字节的变量。

类D的内存分布：cc为填充字节

vptr vptr vptr vptr i cc cc cc a a a a cc cc cc cc f f f f f f f f

再看下例：

class E

{

public:

virtual void f()

{

}

private:

char i;

};

cout << "类E的大小为：" << sizeof(E) << endl; // 大小为8

类E的内存分布：cc为填充字节

vptr vptr vptr vptr i cc cc cc cc

1. 类继承中类的大小

例1：

class Base4

{

private:

int n;

double f;

};

class Base5 : public Base4

{

private:

char i;

};

class H : public Base5

{

private:

char i;

};

cout << "类Base4的大小为：" << sizeof(Base4) << endl; // 大小为16

cout << "类Base5的大小为：" << sizeof(Base5) << endl; // 大小为24

cout << "类H的大小为：" << sizeof(H) << endl; // 大小为32

计算类H的大小时，将其直接基类Base5看成它的第一个（当派生类中有虚函数，而基类中没有虚函数时，直接基类不再是第一个成员，见例3）成员。

类H的内存分布：cc为填充字节

Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 Base 5 i cc cc cc cc cc

cc cc

根据参考C语言学习-sizeof的用法与结构体的长度计算.docx中的字节对齐原则，类H最宽基本类型为直接基类Base5的最宽基本类型，而Base5的最宽基本类为Base4的最宽基本类型double，所以类H的最后要有7个填充字节。

例2：

class Base

{

public:

virtual void f()

{

}

virtual void h()

{

}

};

class Base1

{

public:

virtual void f()

{

}

virtual void h()

{

}

};

class Base2

{

public:

virtual void f()

{

}

virtual void h()

{

}

};

class F : public Base

{

public:

virtual void f()

{

}

virtual void g()

{

}

};

class F1 : public Base, public Base1, public Base2

{

};

cout << "类F的大小为：" << sizeof(F) << endl; // 大小为4

cout << "类F1的大小为：" << sizeof(F1) << endl; // 大小为12

类F的大小为4，原因是：基类与派生类共享了虚函数表指针vptr。

类F1的大小为12，原因是：其有3个直接基类，每个直接基类都与其共享一个vptr指针。

例3：基类中没有虚函数，而派生类中有虚函数

class Base3

{

private:

char i;

double d;

};

class G : public Base3

{

public:

virtual void f()

{

cout << "G::f()" << endl;

}

private:

char i;

// int n;

};

typedef void (\*Fun)(void);

cout << "类Base3的大小为：" << sizeof(Base3) << endl; // 大小为16

cout << "类G的大小为：" << sizeof(G) << endl; // 大小为32

Fun Fun\_ptr = NULL;

G g;

Fun\_ptr = (Fun)\*((int\*)\*(int\*)(&g));

Fun\_ptr(); // 执行G::f()函数

可以看出，如果基类中没有虚函数，而派生类有虚函数，则派生类的虚函数表存放在内存中首地址中。

类G的内存分布：cc为填充字节

vptr vptr vptr vptr cc cc cc cc Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 Base3 i cc cc cc cc cc cc

Base3中最宽基本类型为double型。

例4：虚拟继承