构造函数是类的特殊的成员函数，其名称与类名相同，不具有返回值，由编译器自动调用。

构造函数可以重载，不能为静态函数(只能访问静态变量)，不能为虚函数（虚函数需要实例对象的虚函数表，而实例对象需要构造函数），不能为const函数（不能修改成员变量）。

定义类的构造函数时，应该定义一个默认的构造函数，这样只定义一个对象而不初始化时，可以使用默认构造函数。

class S

{

public:

S(); // 默认构造函数

S(int i);

}

之所以要自己定义默认构造函数，原因是：如果自己定义了一个其它的构造函数（也包括拷贝构造函数），编译器就不在提供默认的构造函数。

类的构造函数是为类对象分配空间并初始化类的成员变量，这是类的定义可以放在头文件中的原因，其不分配内存空间。所以一般情况下，在构造函数中对所有的成员变量都要初始化，以防止在成员函数时使用这些变量时，值是未知的。

C++中，类的成员变量初始化分为两种：使用构造函数的初始化列表和在构造函数体中初始化。例：

#ifndef A\_H

#define A\_H

class A

{

public:

A();

~A();

private:

int a;

int b;

};

#endif // A\_H

源文件：

#include "a.h"

#include <iostream>

A::A():a(10) // 构造函数初始化列表

{

b = 20; // 在构造函数体中初始化

}

A::~A()

{

}

在下面三种情况下，必须使用构造函数初始化列表，这也意味着类必须自己定义构造函数，不能使用编译器提供的默认构造函数。

1. const成员
2. 引用类型成员
3. 没有默认构造函数的类类型成员。

例：

a.h

#ifndef A\_H

#define A\_H

class A

{

public:

// A();

A(int i);

~A();

};

#endif // A\_H

a.cpp

#include "a.h"

#include <iostream>

A::A(int i)

{

std::cout << "调用A构造函数" << std::endl;

}

A::~A()

{

}

b.h

#ifndef B\_H

#define B\_H

#include "a.h"

class B

{

public:

B(int& i);

~B();

public:

void PrintValue();

private:

const int ten;

int& ref;

A a;

};

#endif // B\_H

b.cpp

#include "b.h"

#include <iostream>

// 注意，这里构造函数的形参应定义为引用类型，如果定义为普通类型的i，

// 用其初始化引用ref后，构造函数执行完毕后，i会析构掉，引用ref的值就会是未知的随机数

B::B(int& i):ten(10), ref(i), a(i)

{

std::cout << "调用B构造函数" << std::endl;

}

B::~B()

{

}

void B::PrintValue()

{

std::cout << ten << std::endl;

std::cout << ref << std::endl;

}

int i = 5;

int& ref = i;

B b(ref);

b.PrintValue();

输出为：

调用A构造函数

调用B构造函数

10

5

可见：先执行了A的构造函数，后执行B的构造函数。

类成员初始化的次序：

构造函数的初始化列表仅指定用于初始化成员的值，并不指定初始化执行的顺序。

类成员初始化的次序是按照其定义次序来确定的。

例：

#ifndef C\_H

#define C\_H

class C

{

public:

C();

~C();

private:

int age;

int num;

};

#endif // C\_H

源文件：

#include "c.h"

#include <iostream>

C::C():num(15), age(num + 1)

{

std::cout << num << std::endl;

std::cout << age << std::endl;

}

C::~C()

{

}

C c;

输出的结果，num = 15，age为一个随机数。原因：在头文件中，先定义的是age变量，后定义的num变量，所以在初始化时，也会先初始化age变量，后初始化num变量，在初始化age变量时，num变量未初始化，其值为随机数，所以age的值也是随机数。

在类的构造函数初始化列表中，按照成员声明的顺序进行初始化是一个好习惯，并且尽量避免用一个成员变量去初始化另一个成员变量。

隐式类类型转换：

将其他类型的数据转换为类类型，也可以将类类型的数据转换为其他类型。

将其他类型的数据转换为类类型：

使用单个形参的构造函数可以实现从形参类型到该类类型的转换

例：程序class\_test3

class Test1

{

public:

Test1(int n)

{

std::cout << "调用构造函数" << std::endl;

num = n;

}

Test1(Test1& a)

{

std::cout << "调用拷贝构造函数" << std::endl;

num = a.num;

}

private:

int num;

};

Test1 t1 = 5; // ok，输出“调用构造函数”，注意：编译器将5隐式转换为类对象，并将其赋给了t1，但并没有调用拷贝构造函数。

防止其它数据类型隐式转换为类类型，使用explicit关键字，

class Test1

{

public:

explicit Test1(int n)

{

std::cout << "调用构造函数" << std::endl;

num = n;

}

Test1(Test1& a)

{

std::cout << "调用拷贝构造函数" << std::endl;

num = a.num;

}

private:

int num;

};

Test1 t1 = 5; // error，无法从int类型转换为Test1类型。

全局类对象中构造函数的执行：

例：class\_test46

#ifndef A\_H

#define A\_H

class A

{

public:

A();

~A();

};

#endif // A\_H

#include "a.h"

#include <iostream>

using std::cout;

using std::endl;

A::A()

{

cout << "类A的构造函数" << endl;

}

A::~A()

{

cout << "类A的析构函数" << endl;

}

// 全局类对象中构造函数和析构函数

#include <iostream>

using std::cout;

using std::endl;

#include "a.h"

A a; // 全局类对象，其构造函数是在main函数调用之前调用的。

int main(int argc, char\* argv[])

{

return 0;

}

在main函数的左大括号加一个断点，然后调试，发现运行到断点时，类A的构造函数已经调用，说明其构造函数是在main函数执行之前调用的。