## 构造函数

< 1 > 对象的初始化

在建立一个对象时，常常需要作某些初始化的工作，例如对数据成员赋初值。如果一个数据成员未被赋值，则它的值是不可预知的，因为在系统为其分配内存时，保留了这些存储单元的原状态、成为了这些数据成员的初始值。这种状况显然与人们的要求不相符，对象是反应客观事物属性的实体，应当有确定的值。

如果一个类中的成员都是公有的（即没有私有成员和受保护的成员），则可以在定义对象时对数据成员进行初始化。

例如：

#include <iostream>

using namespace std;

class test{

public: /\* 公有成员 \*/

int num\_1;

string str;

double num\_2;

}; /\* 这个类中没有私有或受保护的成员 \*/

/\* 主函数 \*/

int main(){

test a = {12,"AlexMeson",34.56}; /\* 与结构体类似 \*/

return 0;

}

注意：类的数据成员不能在声明类时初始化，类是抽象类型，不占存储空间。

< 2 > 构造函数的作用

如果使用一个类定义了多个对象，且类中的数据复杂繁多，如何初始化？

C++提供了**构造函数**(constructor)来处理对象的初始化。构造函数是一种特殊的成员函数，与其他成员函数不同，不需要调用，而是在建立对象时自动执行。

构造函数的名字必须与类同名，而不能自由命名，这样便于编译系统能将其识别为构造函数去处理。构造函数不具有任何数据类型和返回值。构造函数的功能是开发者去定义的，能够根据初始化的要求被自主设计函数体和参数。

在类外定义的构造函数只需要在函数名前加上域名和限定符即可。

构造函数可以携带参数，在调用同类型不同对象的构造函数时，可以从外部传不同的数据给构造函数，以实现不同的初始化。写法如下。

例：有两个长方体，长宽高分别为（1）12，23，34；（2）55，66，77。求它们的体积。

#include <iostream>

using namespace std;

typedef unsigned int u32;

class box{

private:

u32 length;

u32 width;

u32 height;

public:

box(u32 l,u32 w,u32 h):length(l),width(w),height(h){}

inline u32 getBoxVolume(){

return length \* width \* height;

}

};

/\* 主函数 \*/

int main(){

box a(12,23,34);

box b(55,66,77);

cout << "Va = " << a.getBoxVolume() << endl;

cout << "Vb = " << b.getBoxVolume() << endl;

return 0;

}

可以观察到：

1. 带参数的构造函数中的形参，其对应的实参在定义对象时给定；
2. 可以为不同的对象进行不同的初始化；
3. 这个例中的写法使用的是**参数初始化表**的写法，也可以在函数体中书写；

注意：

1. 何时调用构造函数？在类对象进入其作用域时调用构造函数（声明周期开始）；
2. 构造函数没有返回值；
3. 构造函数无需也不能被用户调用，例如a.test()。

< 3 > 构造函数的重载

在一个类中可以定义多个构造函数，以便对类的对象提供不同的初始化方法供用户选择。这些构造函数具有相同的名字，而参数的个数或参数的类型不相同。这称为构造函数的重载。

例：对之前的长方体例题进行修改

class box{

private:

u32 length;

u32 width;

u32 height;

public: /\* 构造函数重载 \*/

box():length(0),width(0),height(0){}

box(u32 l,u32 w,u32 h):length(l),width(w),height(h){}

inline u32 getBoxVolume(){

return length \* width \* height;

}

};

int main(){

box a(12,23,34);

box c;

cout << "Va = " << a.getBoxVolume() << endl;

cout << "Vc = " << c.getBoxVolume() << endl;

return 0;

}

注意：

1. 尽管在一个类中可以使用多个构造函数，但是对于每一个对象来说，建立对象时只执行其中一个构造函数，并非每个都执行；
2. 如果在一个类中定义了全部时默认参数的构造函数，不能再定义重载构造函数，因为会出现歧义。如果构造函数中的参数只有部分默认参数，就要根据具体情况分析。

## 析构函数

**析构函数**（destructor）也是一个特殊的成员函数，它的作用与构造函数相反，名字是类名前加一个按位取反的符号。

当对象的声明周期结束时，会自动执行析构函数。具体地说如果出现一下几种情况，程序就会自动执行析构函数：

1. 自动局部对象所在的函数被调用结束时，在对象释放前执行析构函数；
2. 静态局部对象所在的进程结束时，调用该对象的析构函数；
3. 对于全局对象，程序流程离开其作用域时，调用该对象的析构函数；
4. 使用new动态的建立了一个对象，使用delete释放其空间时调用其析构函数。

**析构函数的作用并不是删除对象，而是在撤销对象占用的内存之前完成一些清理动作，使这部分内存可以被进程重新分配给新对象使用，程序设计者事先设计好析构函数，以完成所需的功能，只要对象的生命期结束，程序就自动执行析构函数来完成这些工作。**

析构函数不返回任何值，没有函数类型，也没有函数参数。由于没有函数参数，因此它不能被冲载。一个类可以有多个构造函数，但是只能有一个析构函数。

实际上，析构函数的作用并不仅限于释放资源方面，它还可以被用来执行“用户希望在最后一次使用对象之后所执行的任何操作"，例如输出有关的信息。析构函数是在声明类的时候定义的。也就是说，析构函数可以完成类的设计者所指定的任何操作。

了解构造函数和析构函数的执行顺序：

**user.h**

#ifndef \_USER\_H

#define \_USER\_H

#include <string>

#include <iomanip>

#include <iostream>

class user{

private: /\* 私有 \*/

std::string user\_name;

std::string user\_gender;

std::string user\_id;

public: /\* 公有 \*/

user(char \*name,char \*gender,char \*id);

~user();

void setUserData();

void getUserData();

};

#endif /\* \_USER\_H \*/

**main.cpp**

#include "user.h"

#include <iostream>

using namespace std;

/\* 主函数 \*/

int main(){

user a("Alex","M","221178232");

user b("Meson","M","221178233");

a.getUserData();

b.getUserData();

return 0;

}

**user.cpp**

#include "user.h"

using namespace std;

void user::setUserData(){

cout << "Please input user message:" << endl;

cout << "User name :";

getline(cin,user\_name);

cout << "User gender:";

getline(cin,user\_gender);

cout << "User id :";

cin >> user\_id;

}

void user::getUserData(){

cout << "\tUser name :" << user\_name << endl;

cout << "\tUser gender:" << user\_gender << endl;

cout << "\tUser id :" << user\_id << endl;

}

user::user(char \*name,char \*gender,char \*id)\

:user\_name(name),user\_gender(gender),user\_id(id){}

user::~user(){

cout << user\_name << " object over." << endl;

}

可以观察到，先构造的后析构，后构造的先析构。相当于栈，先进后出。