# C++第二次作业

Y01914541 李剑寒

**4-3**

构造函数的作用就是在对象被创建时利用特定的值构造对象，将对象初始化一个特定的状态；析构函数与构造函数的作用正好相反，它用来完成对象被删除前的一些清理工作，也就是专门做扫尾工作的。

**4-6**

复制构造函数是一种特殊的构造函数，具有一般构造函数的所有特性，其形参是本类的对象的引用。其作用是使用一个已经存在的对象（由复制构造函数的参数指定），去初始化同类的一个新对象。

复制构造函数在以下情况被调用：（1）当用该类的一个对象去初始化该类的另一个对象时；（2）如果函数的形参是类的对象，调用函数时，进行形参和实参结合时；（3）如果函数的返回值是类的对象，函数执行完成返回调用者时。

**4-13**

#include<iostream>

using namespace std;

const float PI = 3.141593;

class circle

{

public:

circle(float r);

circle();

float getarea();

private:

float radius;

};

circle::circle(float r)

{

radius = r;

}

circle::circle()

{

radius = 0;

}

float circle::getarea()

{

return radius \* radius \* PI;

}

int main()

{

float r1;

cout << "Please input the radius: ";

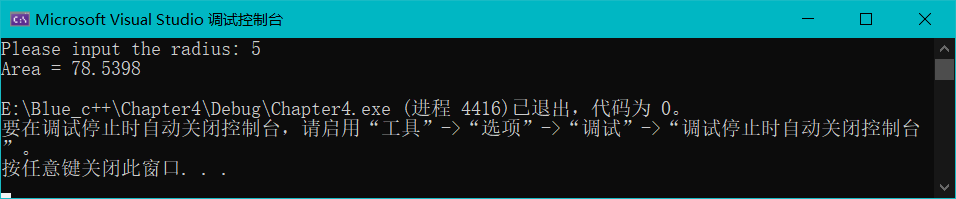
cin >> r1;

circle round1(r1);

cout<<"Area = "<<round1.getarea()<<endl;

return 0;

}



**4-14**

#include<iostream>

using namespace std;

class tree {

public:

void grow(int y);

int age();

tree();

private:

int ages;

};

tree::tree()

{

ages = 0;

}

void tree::grow(int y)

{

ages = y;

}

int tree::age()

{

return ages;

}

int main()

{

tree x;

int y;

cout << "Please input the tree's age : ";

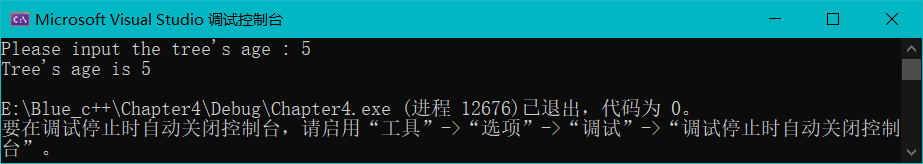
cin >> y;

x.grow(y);

cout << "Tree's age is " << x.age() << endl;

return 0;

}



**4-16**

|  |
| --- |
| Person |
| -Name:char |
| -Age:int |
| +GetName() |
| +Person() |
| +~Person() |
| +Getdate() |

↑

|  |
| --- |
| ZRF |
|  |
|  |

↑

|  |
| --- |
| SSH |
|  |
|  |

**4-19**

#include<iostream>

using namespace std;

enum core{single,dual,quad};

enum words{bit64,bit32};

enum multithreading{support,notsupport};

class cpu {

public:

cpu(unsigned frequence, core type,words length,multithreading mode):

frequence(frequence),coretype(type),wordsize(length),mode(mode){}

void show();

private:

unsigned frequence;

core coretype;

words wordsize;

multithreading mode;

};

void cpu::show()

{

cout << "frequence: " << frequence << endl;

cout << "core: ";

switch (coretype)

{

case single:cout << "single-core" << endl; break;

case dual:cout << "dual-core" << endl; break;

case quad:cout << "quad-core" << endl; break;

}

cout << "words: ";

switch (wordsize)

{

case bit32:cout << "32-bits len" << endl; break;

case bit64:cout << "64-bits len" << endl; break;

}

cout << "multithreading: ";

switch (mode)

{

case support:cout << "suport multithreading" << endl; break;

case notsupport:cout << "not suport multithreading" << endl; break;

}

}

int main()

{

cpu cpu1(3000000000, quad, bit64, support);

cpu cpu2(1500000000, dual, bit32, notsupport);

cout << "size of class cpu1: " << sizeof(cpu1) << endl;

cpu1.show();

cout << "size of class cpu2: " << sizeof(cpu1) << endl;

cpu2.show();

return 0;

}



**4-20**

#include<iostream>

using namespace std;

class complex {

public:

complex(double r,double i):

real(r),image(i){}

complex(double r):

real(r),image(0){}

void show();

void add(complex c2) ;

private:

double real;

double image;

};

void complex::add(complex c2)

{

real += c2.real;

image += c2.image;

}

void complex::show()

{

cout << real << "+" << image << "i" << endl;

}

int main(){

complex c1(1.5, 2.5);

complex c2 = 4.5;

c1.show();

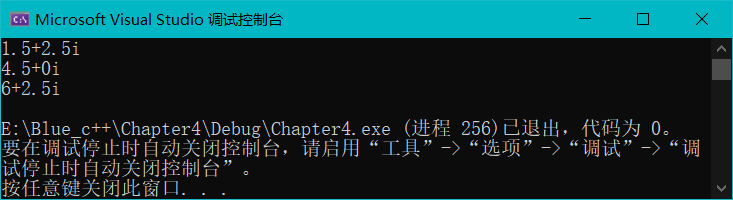
c2.show();

c1.add(c2);

c1.show();

return 0;

}



**4-22**

#include<iostream>

using namespace std;

enum weekday{

Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday

};

int main()

{

weekday day1;

day1 = Friday;

int i;

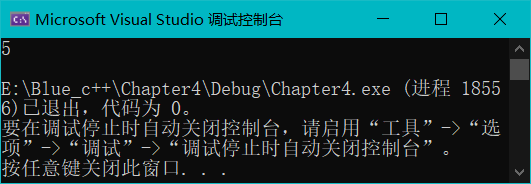
i = day1;

cout <<i << endl;

return 0;

}

**答：可以赋Weekday类型值**



**5-3**

#include<iostream>

using namespace std;

int x = 5, y = 7;

void myFunction() {

int y = 10;

cout << "x from myFunction: " << x << "\n";

cout << "y from myFunction: " << y << "\n\n";

}

int main()

{

cout << "x from main: " << x << "\n";

cout << "y from main: " << y << "\n\n";

myFunction();

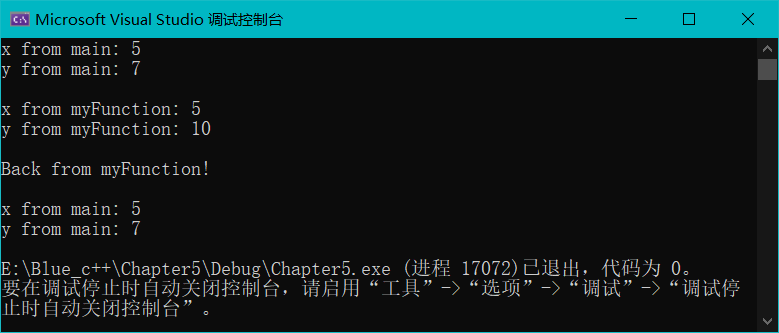
cout << "Back from myFunction! \n\n";

cout << "x from main: " << x << "\n";

cout << "y from main: " << y << "\n";

return 0;

}



**5-4**

class fuel;

class engine {

friend class fuel;

private:

int powerlevel;

public:

engine() { powerlevel = 0; }

void engine\_fn(fuel& f);

};

class fuel {

friend class engine;

private:

int fuellevel;

public:

fuel() { fuellevel = 0; }

void fuel\_fn(engine& e);

};

**5-7**

#include<iostream>

using namespace std;

class cat {

public:

cat(int age) :

itsAge(age) {

numOfCats++;

}

virtual ~cat()

{

numOfCats--;

}

virtual int getAge()

{

return itsAge;

}

virtual void setAge(int age)

{

itsAge = age;

}

static int getNumOfCats()

{

return numOfCats;

}

private:

int itsAge;

static int numOfCats;

};

int cat::numOfCats = 0;

void telepathicFuncton();

int main()

{

const int maxCats = 5;

cat\* catHouse[maxCats];

int i;

for (i = 0; i < maxCats; i++)

{

catHouse[i] = new cat(i);

telepathicFuncton();

}

for (i = 0; i < maxCats; i++)

{

delete catHouse[i];

telepathicFuncton();

}

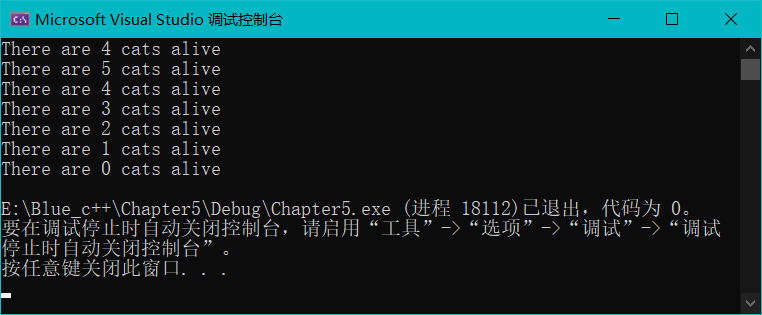
return 0;

}

void telepathicFuncton() {

cout << "There are " << cat::getNumOfCats() << " cats alive" << endl;

}



**5-12**

#include<iostream>

using namespace std;

void fn1()

{

static int n = 0;

n++;

cout << "n为 " << n << endl;

}

int main()

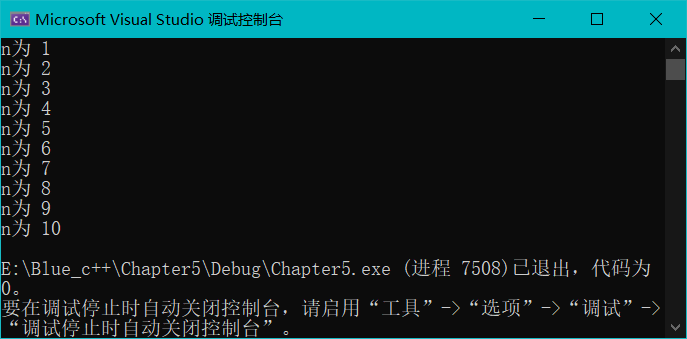
{

for (int i = 0; i <10; i++)

fn1();

return 0;

}



**5-14**

#include<iostream>

using namespace std;

class boat;

class car {

private:

int weight;

public:

car(int j) {

weight = j;

}

friend int getTotalWeight(car& acar, boat& aboat);

};

class boat {

private:

int weight;

public:

boat(int j){

weight = j;

}

friend int getTotalWeight(car& acar, boat& aboat);

};

int getTotalWeight(car& acar, boat& aboat) {

return acar.weight + aboat.weight;

}

int main()

{

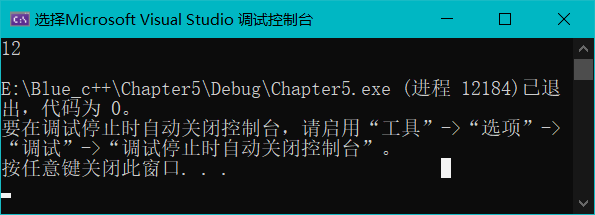
car c1(3);

boat b1(9);

cout << getTotalWeight(c1, b1) << endl;

return 0;

}



**5-15**

局部作用域中静态变量的特点是：它并不会随着每次函数调用而产生一个副本，也不会随着函数返回而失效，定义时未指定初值的基本类型静态生存周期变量，会被以0值初始化；局部作用域中的全局变量诞生于声明点，结束于声明所在的块执行完毕之时，并且不指定初值意味着初值不确定；

普通局部变量存放于栈区超出作用域后，变量被撤销，其所占用的内存也被收回；静态局部变量存放于静态数据存储区，全局可见，但是作用于是局部作用域，超出作用域后变量仍然存在。

**5-16**

编译输入源文件，输出目标文件；连接输入目标文件，输出可执行文件；编译器对源代码进行编译，是将 以文本形式存在的源代码翻译为机器语言形式的目标文件的过程。连接是将各个编译单元的目标文件和运行库当中被调用过的单元加以合并后生成的可执行文件的过程。

（1）编译时报错，函数参数不匹配；

（2）连接错误，函数未定义；

（3）不报错；

（4）连接错误，函数重复定义。