1.内存四区：

代码区：存放函数体的二进制代码、注释等，由操作系统进行管理

1）代码区的特点：共享的、只读的

全局区：存放全局变量和静态变量和常量（字符串常量、用const修饰的全局变量），该区域的数据在程序结束后由操作系统释放

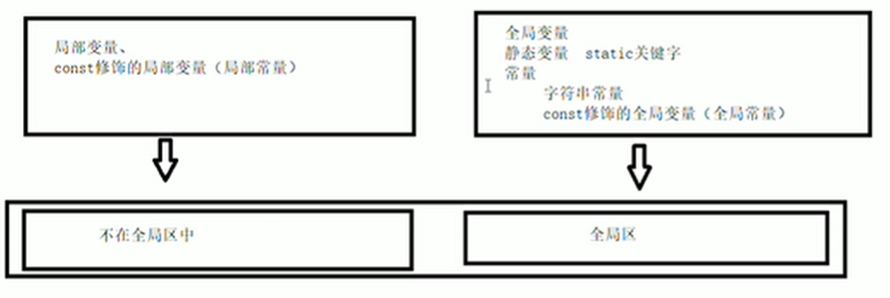
1）静态变量只被初始化一次，静态变量其内存空间是只有在程序结束后才由系统释放

2）全局变量和静态全局变量的区别：

非静态全局变量的作用域是整个源程序，

静态全局变量的作用域只在定义该变量的源文件内有效，在同一源程序的其他源文件不能使用它。静态局部变量虽然只能在函数体内可以使用，但其内存空间是只有在程序结束后才由系统释放，所以静态局部变量可以通过引用返回

3）用const修饰的变量可分为const修饰全局变量和const修饰局部变量，只有const修饰的全局变量和字符串常量才放在全局区中。



栈区：由编译器自动分配和释放，存放函数的形参和局部变量等

堆区：由程序员手动分配和释放，若程序员不释放，程序结束时由操作系统回收

2.new运算符，在堆区创建数据

int \*p = new int(10); //new返回的是该数据类型的指针

delete p; //释放堆区的数据

int \*p = new int(10); //创建一个int类型变量在堆区，并赋值为10

int \*p = new int[10]; //创建10个整型数据的数组在堆区，这里的10代表数组有10个元素，其返回值返回的是数组的首地址，按类似栈上使用数组的方式区使用 p[3]/p[5]

释放堆区数组 delete[] p

3.引用

1）引用必须初始化，并且初始化后就不可以改变了，引用就是给变量起别名，不要返回局部变量的引用。

2）引用的本质在C++内部实现是一个指针常量,即：int &ref = a; <==> int \* const ref = &a

3）void ShowValue(const int& v) 常量引用是用来修饰形参，防止函数内部的误操作

4.指针常量和常量指针

指针常量：其本质上是一个常量，指针用来说明常量的类型 int \* const p = &a //指向的地址不可改变，当其内容可以修改

常量指针：其本质上是一个指针，常量表示指针指向的内容 const int \*p = &a //指向的地址可以改变，但其内容不可以改变

const在\*的左边，则指针指向的变量的值不可直接通过指针改变（可以通过其他途径改变）；

const在\*的右边，则指针的指向不可变。简记为“左定值，右定向”。

5.指针函数和函数指针

指针函数：其本质是一个函数，只是其返回值是一个指针 int \* func\_sum(int n)

函数指针：其本质是一个指针，只是其指向的是一个函数 int (\*p)(int, int); //函数指针的定义

typedef int(\*pSum)(int a,int b); //有typedef时，pSum是一个指针类型，他可以初始化实例；int(\*pSum)(int a,int b); //无typedef时，pSum仅仅是一个指针。

6.函数默认参数

声明和实现只能有一个默认参数

int func(int a, int b = 10);

int func(int a, int b = 10) //错误  
{

}

如果某个位置有默认参数，那么从这个位置往后，从左向右都必须有默认值

int func(int a, int b = 10, int c) //错误

7.占位参数

int func(int a, int)

占位参数也可以有默认参数

int func(int a, int = 10)

8.函数重载

1）同一作用域下

2）函数名称相同

3）函数参数类型不同，或者个数不同，或者顺序不同

int func(int a)

int func(double a)

使用函数重载的时候尽量不要使用默认参数，否则可能会发生二义性

8.sizeof()是一个操作符，其作用是返回一个对象或类型的字节数

9.

char \*strcpy(char \*dest, const char \*src)

void \*memcpy( void \*dest, const void \*src, size\_t count );

strcpy和memcpy的不同

1）复制的内容不同。strcpy只能复制字符串，而memcpy可以复制任意类型的内容。strcpy只用于字符串复制，并且还会复制字符串的结束符。memcpy对于复制的内容没有限制，用途更广。

2）复制的方法不同。strcpy不需要指定长度，遇到结束符’\0’才会结束，所以容易溢出。memcpy则是根据第三个参数决定复制的长度

3）用途不同。通常在复制字符串时用strcpy，在复制其他类型数据时一般用memcpy。

10.

三种访问权限

公有权限：public 成员类内可以访问，类外也可以访问

（类外访问就比如说s1.m\_name=”132”）

保护权限：protected 成员类内可以访问，类外也可以访问

继承方式：private 成员类内可以访问，类外也可以访问

继承方式包括 public（公有的）、private（私有的）和 protected（受保护的），此项是可选的，如果不写，那么默认为 private。不同的继承方式会影响基类成员在派生类中的访问权限。

（1）public继承方式

基类中所有 public 成员在派生类中为 public 属性；

基类中所有 protected 成员在派生类中为 protected 属性；

基类中所有 private 成员在派生类中不能使用。

（2）protected继承方式

基类中的所有 public 成员在派生类中为 protected 属性；

基类中的所有 protected 成员在派生类中为 protected 属性；

基类中的所有 private 成员在派生类中不能使用。

（3）private继承方式

基类中的所有 public 成员在派生类中均为 private 属性；

基类中的所有 protected 成员在派生类中均为 private 属性；

基类中的所有 private 成员在派生类中不能使用。

注意，我们这里说的是基类的 private 成员不能在派生类中使用，并没有说基类的 private 成员不能被继承。实际上，基类的 private 成员是能够被继承的，并且（成员变量）会占用派生类对象的内存，它只是在派生类中不可见，导致无法使用罢了，需要调用基类的公有和保护成员来访问

11.C++ 中struct和class的唯一区别就是默认访问权限不同

区别：

struct默认权限为公共

class默认权限为私有

12. #pragma once // 可以防止头文件重复包含

13.默认构造函数是无参构造函数，调用默认构造函数时，不要加（）

Person p1；//调用默认构造函数

Person p1(10); //调用有参构造函数

Person(const Person &p) //拷贝构造函数写法

Person(10) //匿名对象，不要利用拷贝构造函数初始化匿名对象

14.

拷贝构造函数调用时机：1.使用一个已经创建完毕的对象来初始化一个新对象2.值传递的方式给函数参数传值3.以值方式返回局部对象

15.在默认情况下，创建一个类，C++编译器至少给一个类添加三个函数1.默认构造函数2.析构函数3.拷贝构造函数,

如果有定义有参构造函数，编译器就不会提供无参构造函数，

如果有定义拷贝构造函数，编译器就不会提供其他构造函数

16.

浅拷贝：简单的赋值拷贝操作

深拷贝：在堆区重新申请空间，进行拷贝操作

如果属性有在堆区开辟的，要自己提供拷贝构造函数，防止浅拷贝带来的问题

有在堆区分配空间的时候要在析构函数内释放其空间，

17.初始化列表

Person（int a, int b）：m\_A(a), m\_B (b) {}

18.当其他类对象作为本类成员，构造时候先构造其他类对象，再构造自身，析构的顺序和构造函数相反

19.静态成员变量1.所有对象共享同一份数据2，再编译阶段分配内存3类内声明，类外初始化

class Person

{

Public:

staic int m\_A;

}

int Person::m\_A = 100;

静态成员变量可以通过两种方式来访问

Person p；

p.m\_A = 10； //通过对象来进行访问

Person::m\_A = 10； //通过类名来进行访问

20.静态成员函数所有对象共享同一个函数，静态成员函数只能访问静态成员变量

也同静态成员变量一样可以通过两种方式来访问1. 通过对象来进行访问2. 通过类名来进行访问

21.C++中，类内的成员变量和成员函数是分开存储的，只有非静态成员变量才属于类的对象，每一个非静态成员函数也只会诞生一份函数实例，所以C++为了知道到底是哪一个对象调用自己，就通过this指针来实现

this指针指向被调用的成员函数所属的对象，this指针隐含在每一个非静态成员函数内。

this指针的本质是指针常量，指针的指向是不可以修改的

this指针的用途：

当形参和成员变量同名时，可以用this指针来区分

在类的非静态成员函数中返回对象本身，可以使用 return \*this；

Person &PersonAddAge（Person &p）

{

return \*this //返回对象本身的时候要注意返回值为引用

}

22.用const修饰成员函数，其称为常函数，常函数内不可修改成员属性，在成员属性声明时加上mutable后，在常函数中依然可以修改。

在声明对象前加const称该对象为常对象，常对象只能调用常函数，在成员属性声明时加上mutable后，常对象依然可以修改

void showPerson() const //常函数

{

}

const Person p; // 常对象

23. 如果是友元就可以访问类中的私有成员。

友元有三种实现，1.全局函数做友元2.类做友元3.成员函数做友元。如果是友元就可以访问类中的私有成员

class Building

{

friend void goodGay(building \*building); //全局函数做友元 （放在类的最前面声明）

};

void goodGay(building \*building)  
{

}

///////////////////////////////////////////////////////////////////////////////////////////

class Building

{

friend class Goodgay; //类做友元 （放在类的最前面声明）

}；

class Goodgay

{

};

///////////////////////////////////////////////////////////////////////////////////////////

class Building

{

friend void Goodgay::visit(); //成员函数做友元（放在类的最前面声明）

prvate:

string m\_Building;

}；

class Goodgay

{

public:

Goodgay()

{

m\_building = new building;

}

public:

void visit()

{

cout << m\_building->m\_BedRoom >> endl;

}

private:

Building \*m\_building;

};

24.对于加号运算符，如果是内置的数据类型，编译器知道如何进行运算，而如果是自定义类型，编译器就不知道如何进行运算，这时候就可以使用运算符重载

//加号运算符重载 1成员函数重载加号运算符， 2.全局函数重载加号运算符

class Person

{

public:

Person operator+(Person &p) //成员函数重载加号运算符

{

m\_A += p.m\_A;

m\_B += p.m\_B;

}

int m\_A;

int m\_B;

}

Person operator+(Person &p1, Person &p2) //全局函数重载加号运算符

{

p1.m\_A += p2.m\_A;

p1.m\_B += p2.m\_B;

}

///////////////////////////////////////////////////////////////////////////////////////////

利用全局函数重载左移运算符(用成员函数无法实现)

ostream &operator<<(ostream &cout, Person &p)

{

cout << “m\_A” << p.m\_A << “m\_B” << p.m\_B << endl;

}

///////////////////////////////////////////////////////////////////////////////////////////

class MyInteger

{

public:

MyInteger &operator++() //重置前置++运算符

{

m\_Num++;

return \*this

}

MyInteger operator++(int) //重置后置++运算符

{

MyInteger temp = \*this;

m\_Num++;

return temp;

}

private:

int m\_Num;

}