

Chapter 1

1. C++是一门面向对象、编程的语言，比C更优略
2. 字符串可以实行加减，并且可以当作参数传递，函数会开辟一块空间，作为其值的承担
3. 字符串复习时需要知道的函数大致有

```
getline(cin,s)与cin>>s
s.length()://用于求长度
s.insert(int pos,string ss)//从pos位置开始插入一个字符串 比如
s="0123",s.insert(1,"aa")->0aa123
s.erase(int pos) s.erase(int pos,int len)
s2=s1.substr(int pos,int len)
i=s.find(string s1,int pos)//其中pos可以没
s.compare(ss)//一样的时候为0
s.replace(int pos,int len,string ss)//替换从pos开始长度为len的部分为ss
```

4.区分new与delete

```
int *p=new int
int *q=new int[10]
delete p
delete[] q//这里注意[]的位置是在delete后面紧跟的
```

5.析构函数:在自动清空内存时做一些事情

```
#include <iostream>
using namespace std;
class CDemo {
public:
    ~CDemo() { cout << "destructor" << endl; }
};

void Func(CDemo obj) {
    cout << "func" << endl;
}

CDemo d1;
CDemo Test() {
    cout << "test" << endl;
    return d1;
}

int main() {
    CDemo d2;
    Func(d2);
    Test();
    cout << "after test" << endl;
    return 0;
}

/*
destructor
test
destructor
*/
```

```
after test
destructor
destructor
*/
```

程序共输出 destructor 四次：

- 第一次是由于 Func 函数结束时，参数对象 obj 消亡导致的。
- 第二次是因为：第 20 行调用 Test 函数，Test 函数的返回值是一个临时对象，该临时对象在函数调用所在的语句结束时就消亡了，因此引发析构函数调用。
- 第三次是 main 函数结束时 d2 消亡导致的。
- 第四次是整个程序结束时全局对象 d1 消亡导致的。

Chapter 2

1. 定义与声明的区别：

- a. 变量定义：用于为变量分配存储空间，还可为变量指定初始值。程序中，变量有且仅有一个定义。
- b. 变量声明：用于向程序表明变量的类型和名字。
- c. 定义也是声明：当定义变量时我们声明了它的类型和名字。
- d. `extern`关键字：通过使用`extern`关键字声明变量名而不定义它。
- e. 定义也是声明，`extern`声明不是定义，即不分配存储空间。`extern`告诉编译器变量在其他地方定义了。

例如：`extern int i; //声明，不是定义`

`int i; //声明，也是定义`

`struct node //声明`

`struct node {int a,int b} //定义`

`typedef int INT //声明`

f. 如果声明有初始化式，就被当作定义，即使前面加了`extern`。只有当`extern`声明位于函数外部时，才可以被初始化。

例如：`extern double pi=3.1416; //定义`

g. 函数的声明和定义区别比较简单，带有`{ }`的就是定义，否则就是声明。

例如：`extern double max(double d1,double d2); //声明`

h. 不要把变量定义放入`.h`文件，这样容易导致重复定义错误，可以申明。

i. 对于局部变量以及全局静态变量是不能通过`extern`进行前置申明的，即不能在定义之前通过申明来引用，因为局部变量的作用域是当前代码块，全局静态变量的作用域是当前源文件，**都不是全局作用域**，所以不能通过`extern`进行前置申明。全局变量允许在定义之前通过前置申明进行引用。

```
#include <stdio.h>

extern int a;
extern static int b;    //报错
int main()
{
    extern int c;        //报错
    printf("a=%d,b=%d,c=%d",a,b,c);
    int c=2;
}
int a=0;
static int b=1;
```

2. 对象中的一些小注意点

a. 同名时

```
using namespace std;
class Point{
private:
    int a,b;
public:
    void print();
    Point(int aa,int bb){a=aa; b=bb;}
};
int a=10;
void Point::print()
{
    cout<<"The part a="<<a<<endl; //这里相当于调用类中的变量的意思
    ::a=20; //这里是全局变量的意思
    cout<<"The global a="<<::a<<endl;
}
```

b. private、public与protected:

类的public成员可以被任意实体访问，可以认为它就是c语言中的struct结构体，可以直接用a.x这种形式访问；

类的private成员不能直接被类的实体访问，也不能被子类的实体访问，但是可以被类的成员函数访问；

类的protected成员不能直接被类的实体访问，但是可以被子类访问，也可以被类的成员函数访问；

3. default constructor

它是指没有参数的构造函数（不管是编译器隐式生成的，还是程序员显式声明的）


[(25条消息) [C++ default constructor默认构造函数 李正浩大魔王的博客-CSDN博客c++ constructor default](#)]

4. auto

它让编译器自己想出一个类型

Chapter 3

1. #include有关知识

 41MZ~4LC7Y3BIP~\$KLVK_R3

在理论上来说可以是自由命名的，但每个头文件的这个“标识”都应该是唯一的。标识的命名规则一般是头文件名全大写，前后加下划线，并把文件名中的“.”也变成下划线，如：

 41MZ~4LC7Y3BIP~\$KLVK_R3

Chapter 4

1. Vector库

```
//下面展示几个比较常见的用法
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
    vector<int>value;
    int a=2,d=0;
    value.push_back(3);
    value.push_back(a);
    value.push_back(d);
    cout<<value.size()<<endl;
    for (vector<int>::iterator i=value.begin(); i<value.end(); i++)
        cout<<"*i="<<*i<<endl;//输出
    sort(value.begin(),value.end());
    for (int i=0; i<value.size(); i++)
        cout<<value[i]<<endl;//数组方法输出
    value.push_back(-3);
    value.push_back(10);
    sort(value.rbegin(),value.rend());
    for (int i=0; i<value.size(); i++)
        cout<<value[i]<<endl;
    if (value.empty()) cout<<"yes"; else cout<<"no";//cout NO
    value.pop_back();//移除队尾元素
} //正向与逆向的排序

vector初始化
vector<int>v1;
vector<int>v2(v1);
vector<int>v3=v1
vector<int>v4(3,6)//3个6
vector<int>v5(3)//3个0
vector<int>{2,4,6}
```

2. map库

```

#include<map>
map<int, string> m;
//有两种插入的方法
m.insert(pair<int, string>(111, "kk"));
m.insert(map<int, string>::value_type(222, "pp"));
m[123]="dd"
//这两个的区别是 后者会进行覆盖 而前面的插入 对于每个范围只能有一次
map<int, string>::iterator it;
it=maps.find(123);->it=map.end();
map.erase(it)//消除这个map值
map.size()//返回大小

```

3. list就是数据结构中的双向链表，因此它的内存空间是不连续的，通过指针来进行数据的访

```

list<int> l
l.push_back(1);
l.push_front(2);
for(list<int>::iterator iter = l.begin() ; iter != l.end() ; iter++)
    cout<<*i<<endl;
l.empty();//比count快

```

4. 迭代器

正向迭代器。假设 p 是一个正向迭代器，则 p 支持以下操作：++p, p++, *p。此外，两个正向迭代器可以互相赋值，还可以用 == 和 != 运算符进行比较。

双向迭代器。双向迭代器具有正向迭代器的全部功能。除此之外，若 p 是一个双向迭代器，则 --p 和 p-- 都是有定义的。--p 使得 p 朝和 ++p 相反的方向移动。

随机访问迭代器。随机访问迭代器具有双向迭代器的全部功能。若 p 是一个随机访问迭代器，i 是一个整型变量或常量，则 p 还支持以下操作 p+i 等

输入迭代器：支持==与!=，支持访问成员，支持后缀增加

输出迭代器：支持修改元素，支持后缀增加

Chapter 5

1. field(成员变量)、local variable (局部变量)、parameter variable (参数变量)

field是跟随类的生成而生成，释放而释放的，最主要是区分private、protected、public的区别

formal parameter形参具体要关注下string类型

2. 函数重载

编译器会自动寻找到最匹配适合的

3. 默认参数

默认参数一定要从右往左赋值void fun(int a,int b=1,char c='a',float d=3.2);此时调用fun(2, 'c',3.14)错误，从左到右，所以fun(2,3)时，a=2,b=3

默认参数不能在声明和定义中同时出现，一般先声明再定义

```
void fun(int a=5)
```

```
void fun(int a){},不能在这里也给a=一个值
```

4. 友元函数

在当前类以外定义的、不属于当前类的函数也可以在类中声明，但要在前面加 friend 关键字，这样就构成了友元函数。友元函数可以是不属于任何类的非成员函数，也可以是其他类的成员函数。友元函数可以访问当前类中的所有成员，包括 public、protected、private 属性的

注意，友元函数不同于类的成员函数，在友元函数中不能直接访问类的成员，必须要借助对象。

```
class Student{
public:
    Student(char *name, int age, float score);
public:
    friend void show(Student *pstu); //将show()声明为友元函数
private:
    char *m_name;
    int m_age;
    float m_score;
};

Student::Student(char *name, int age, float score): m_name(name),
m_age(age), m_score(score){ }

//非成员函数
void show(Student *pstu){
    cout<<pstu->m_name<<"的年龄是 "<<pstu->m_age<<", 成绩是 "<<pstu-
>m_score<<endl;
} //借助对象，正确
void show(){
    cout<<m_name<<"的年龄是 "<<m_age<<", 成绩是 "<<m_score<<endl;
} //这样的写法是错误的
```

```
class Address//帮助student里声明函数
//声明Student类
class Student{
public:
    Student(char *name, int age, float score);
public:
    void show(Address *addr);
private:
    char *m_name;
    int m_age;
    float m_score;
};

//声明Address类
class Address{
private:
    char *m_province; //省份
    char *m_city; //城市
    char *m_district; //区（市区）
public:
    Address(char *province, char *city, char *district);
    //将Student类中的成员函数show()声明为友元函数
    friend void Student::show(Address *addr);
};

//实现Student类
Student::Student(char *name, int age, float score): m_name(name),
m_age(age), m_score(score){ }
void Student::show(Address *addr){
    cout<<m_name<<"的年龄是 "<<m_age<<", 成绩是 "<<m_score<<endl;
    cout<<"家庭住址: "<<addr->m_province<<"省"<<addr->m_city<<"市"<<addr-
>m_district<<"区"<<endl;
}
```

```

}

//实现Address类
Address::Address(char *province, char *city, char *district){
    m_province = province;
    m_city = city;
    m_district = district;
}

```

5. 友元类

类 B 声明为类 A 的友元类，那么类 B 中的所有成员函数都是类 A 的友元函数，可以访问类 A 的所有成员，包括 public、protected、private 属性的。

```

class Address; //提前声明Address类

//声明Student类
class Student{
public:
    Student(char *name, int age, float score);
public:
    void show(Address *addr);
private:
    char *m_name;
    int m_age;
    float m_score;
};

//声明Address类
class Address{
public:
    Address(char *province, char *city, char *district);
public:
    //将Student类声明为Address类的友元类
    friend class Student;
private:
    char *m_province; //省份
    char *m_city; //城市
    char *m_district; //区（市区）
};

//实现Student类
Student::Student(char *name, int age, float score): m_name(name),
m_age(age), m_score(score){ }
void Student::show(Address *addr){
    cout<<m_name<<"的年龄是 "<<m_age<<"，成绩是 "<<m_score<<endl;
    cout<<"家庭住址: "<<addr->m_province<<"省"<<addr->m_city<<"市"<<addr->m_district<<"区"<<endl;
}

//实现Address类
Address::Address(char *province, char *city, char *district){
    m_province = province;
    m_city = city;
    m_district = district;
}

```

6. 2

7. 内联函数：用来建议[编译器](#)对一些特殊[函数](#)进行内联扩展（有时称作[在线扩展](#)）；也就是说建议编译器将指定的函数体插入并取代每一处调用该函数的地方（[上下文](#)），从而节省了每次调用函数带来的额外时间开支。但在选择使用内联函数时，必须在程序占用空间和程序执行效率之间进行权衡，因为过多的比较复杂的函数进行内联扩展将带来很大的存储资源开支。

8. Const

1. **指针常量**：不能修改指针所指向的**地址**。在定义的同时必须初始化。

```
char * const a = &p; //指针常量。
*a = 'a'; // 操作成功
a = &b; // 操作错误，指针指向的地址不能改变
```

2. **常量指针**：不能修改指针所指向**地址的内容**。但可以改变指针所指向的地址。

```
char const * a; //同上
*a = 'a'; //操作错误，指针所指的值不能改变
a = &b; //操作成功
```

- 如果const位于*的右侧，则const就是修饰指针本身，即指针本身是常量（指针常量）
 - 如果const位于*的左侧，则const就是修饰指针所指向的变量，即指针指向常量（常量指针）；
- 重要的是：const 通常用在函数形参中，如果形参是一个指针，为了防止在函数内部修改指针指向的数据，就可以用 const 来限制。

9. Static

静态全局变量：只在当前文件可见，别的文件都不可见

静态局部变量：同平时理解

静态函数：同静态全局变量去理解

静态数据成员：所有类用同一个，在全局内存区

```
#include <stdio.h>
class TempClass
{
public:
    TempClass(int a, int b, int c);
    void Show();
private:
    int a,b,c;
    static int T;
}
int TempClass::T = 0; //初始化静态数据成员
TempClass::TempClass(int a, int b, int c)
{
    this->a = a;
    this->b = b;
    this->c = c;
    T = a + b + c;
}
void TempClass::Show()
{
    printf("T is %d\n", T);
}
```



```
}  
int main()  
{  
    TempClass ClassA(1,1,1);  
    ClassA.Show(); //输出1+1+1 = 3;  
    TempClass ClassB(3,3,3);  
    ClassB.Show(); //输出3+3+3 = 9;  
    ClassA.Show(); //输出9  
    return 0;  
}
```

静态成员函数：与普通成员函数的根本区别在于：普通成员函数有 this 指针，可以访问类中的任意成员；而静态成员函数没有 this 指针，只能访问静态成员（包括静态成员变量和静态成员函数）。