《C++Primer》自我知识点总结

# 第一章：一个简单的C++函数：

int main()

{

Return 0;

}

一个函数拥有四部分构成：返回类型：int 函数名：main 参数列表：（） 函数体{}

Return会返回一个值，并且此值的类型必须与函数的返回类型相同

大多数系统中，main的返回值用来指示状态，0表明成功，非0的含义由系统定义，指出作用类型。

## 1.输入输出：

2.1包含了一个标准库来提供IO机制（以及其他很多设施）。大多数采用了iostream库，其中包含了两个基础类型，istream和ostream，表示输入流和输出流。

2.2一个流就是一个字符序列，是从IO设备读出或者写入的。

2.3标准库定义了4个IO对象，cin-istream类型的对象：标准输入 cout-ostream：标准输出 cerr:标准错误 clog:用来输出程序运行时的一般性信息。

程序示例：

#include "iostream"

using namespace *std*;

int *main*(){

*cout*<<"Enter two numbers"<<*endl*;

int v1=0,v2=0;

*cin*>>v1>>v2;

*cout*<<"the sum of "<<v1<<"and"<<v2<<"is"<<v1+v2<<*endl*;

*system*("pause");

}

用到标准库：放在开头，#include “iostream”

Main函数中第一条语句是个表达式，在C++中一个表达式对应一个结果。通常由多个对象和一个运算符组成。这条语句的运算符是：”<<”输出运算符

“<<”接受两个运算对象，左侧是一个ostream对象，右侧是运算对象要打印的值。

Enter two numbers则是输出的一个：字符串字面值常量。

Endl：含义是一个：操纵符。结束当前行，并且将设备关联的缓冲区中的内容刷到设备中去。保证程序产生的输出都真正写在输出流中。

” >>”输入运算符：“>>”同样接受两个运算对象，左侧是个istream对象，右侧是运算对象要输出的值。

1. C++中含有两种注释，单行注释：// 多行注释/\*…\*/

3.2：注释界定符不能嵌套

## 2.控制流

4.1while语句：语法

While（val<=10){

Sun+=val;

++val;

}

语句块：用花括号包围的多条语句的序列。

小于等于运算符：<= 复合赋值运算符：+= 前缀递增运算符：++

4.2for语句

每个for语句包含两部分：循环头和循环体。

循环头：用来控制循环体的执行次数。 循环体：进行逻辑执行。

4.3读取数量不定的输入数据

int *main*()

{

int sum=0;

int value=0;

while(*cin*>>value){

sum+=value;

}

*cout*<<"Sum is"<<sum<<*endl*;

*system*("pause");

return 0;

}

此处主要是利用while循环，直到用户输入一个无效字符为止。将用户的输入保存在value中，然后整体赋值给cin对象（返回左侧的运算对象）。

一些注意的关键点：一些容易出错的语法，冒号，引号，还有多个输入输出时候中间的运算符。还有常犯的类型错误与声明错误。

4.4if语句

If语句用来做基本的条件判断，常常和for,while搭配使用。

## 3.类简介

类机制是C++最重要的特性之一。最初的设计焦点就是创造一个类类型。 一个类的作者决定了类类型对象上可以使用的所有操作，应该明白类的成员变量和成员方法。（关于类的说明后面会有详细的说明和介绍）。

# 第二章：变量和基本类型

## 1.关于数据类型的一般知识点：

1.1C++的基本类型：算术类型和空类型。其中算术类型又包含了字符类型，整数型，布尔型和浮点型。一般当函数不返回任何值的时候使用空类型作为返回类型。

1.2除此之外，其他整型可以划分为带符号的和无符号的两种类型。带符号的可以表示整数，0，或者负数。而不带符号的则只能表示大于等于0的值。

1.3int,short,long和long long类型都是带符号的，如果想不带符号则可以在类型名前添加unsigned，就可以得到无符号类型了。例如unsigned long,unsigned int型等。但是：char被分为了三种，char,signed char和unsigned char，但是表现形式只有两种，具体情况由编译器来决定。

给出的建议：使用int执行整数运算，执行浮点数运算选用double。

## 2.关于类型转换的一些知识点：

2.1当把一个非布尔类型的算术值赋给布尔类型时，值是0则表示false，非0则是true。

2.2把一个整型给浮点型则会精度缺失，仅保留浮点数中小数点前的部分。

2.3如果给一个无符号的赋值超过范围时，则实际结果等于该值对表示的值总数取绝对值后的余数。例如：（8bit大小的unsigned char表示0到255区间的值）

Unsigned char c=-1;

则实际c的值为: |-1/256|%=1 256/1=1...255

2.4当我们给一个带符号类型的值超出表示的范围时，结果是未定义的，有可能会造成系统奔溃！！！例如

Signed char c2=256 c2的结果是未定义的

## 3.含有无符号类型的表达式：

3.1当一个算术表达式中既有无符号数又有int值时，int值就会转换成无符号数。其过程就类似于把int的值直接赋给无符号变量一样。（只会出现int是负数的情况）：

Unsigne u=10;

Int i=-42;

Cout<<i+u<<endl; 值为4294967264;

但是： Unsigned u=-10;

Int i=42;

Cout<<i+u<<endl: 值为32

3.2当从无符号中减去一个值时，要确保结果不能是负值。

Unsigned u1=42,u2=10;

Cout<<u1-u2; //正确 输出32

Cout<<u2-u1; //结果是取模之后的结果，超级大的那种

3.3切勿混用带符号类型和无符号类型，此时带符号类型取值为负时会出现异常，因为会自动地转换成无符号数。

## 4.字面值常量

例如：42这个值就被称为字面值常量。

4.1整型和浮点型字面值常量

整型字面值常量具体的数据类型由它的值和符号决定。默认情况下十进制字面值是带符号数，八进制和16进制字面值常量既可以带符号也可能是无符号的类型。

浮点型字面值常量表现为小数或者以科学计数法表示的指数。

4.2字符和字符串字面值

单引号括起来的一个字符称为char型字面值，双括号括起来的零个或者多个字符则构成了字符串型字面值。

‘a’ “Hello World”

注意：字符串字面值的类型实际上是由常量字符构成的数组。并且编译器在每个字符串的结尾处添加一个空字符 ‘\0’，因此字符串字面值的实际常量要比内容多1。

4.3转移序列

程序员不能使用的两类字符：一是不可打印的字符，如退格或者其他控制字符。第二类是在C++中含有特殊含义的字符：例如单引号，双引号，问号，反斜线等。C++的转移序列包括如下：

换行符：\n 反斜线：\\ 回车符:\r

问号：\? 双引号\” 单引号：\’

4.4布尔字面值和指针字面值

True和false是布尔类型的字面值。Nullptr是指针字面值。

4.5字面值的类型

可以通过添加前缀和后缀改变整型，浮点型和字符型的默认类型。

前缀：u U L u8 后缀u or U l or L ll or LL f or F

## 5.变量

5.1变量提供一个具名的，可供程序操作的存储空间。简而言之，变量就是对应的一段存储空间。变量类型决定着所占内存的大小以及布局方式。变量和对象可以互换。

5.2string是一种表示可变长字符序列的数据类型。C++提供了对string的几种初始化方法，其中一种是把字面值拷贝给string对象。book变量

string book(“0-201-x”); //拷贝的方式，字面值拷贝

5.3对象的概念：

对象是指一块能存储数据并具有某种类型的内存空间。对象-内存空间的别名

在C++中，初始化是一个异常复杂的问题。在C++中，初始化和赋值是两个完全不同的操作，虽然有时这种区别无关紧要，但是这个概念是至关重要的！！！！！

5.4C++进行初始化变量的方式有几种，例如以下四种都可以：

Int units\_sold = 0;

Int units\_sold = {0};

Int units\_sold{0};

Int units\_sold(0);

以上这种初始化的方式成为：列表初始化。采用花括号的形式

5.5默认初始化

如果定义变量时没有指定初值，则变量被默认初始化。默认值到底是什么由变量类型决定，定义变量的位置也会有影响。

注意：如果是内置类型的变量未被初始化时，它的值由定义的位置决定，定义在函数体之外的变量被初始化为0。每个类各自决定其初始化对象的方式，也可以决定对象的初始值到底是什么。

建议：每一个内置类型的变量尽可能地为其赋一个初值，会为了以后的安全所着想。

5.6变量声明和定义

C++支持分离式编译。允许将程序分隔为若干个文件，每个文件可以被单独编译。因此声明和定义区分开来。声明：让名字被程序所知，包含对名字的声明。定义：负责创建与名字关联的实体。

想声明一个变量而非定义，就在变量明前添加关键字extern，并且不要初始化。例如：

extern int i; //只声明而非定义

Int ; //声明并定义j

如果将extern声明的变量进行了赋值，就变成了定义，在函数体内就可能引发错误。

变量能切只能被定义一次，但是可以被多次声明。也就是分配空间只能有一次。使用extern声明的变量是不会分配内存的，因此可以多次进行声明。

5.7作用域

区分全局作用域和块作用域。建议：当第一次使用变量时再进行定义。

很重要的一点：如果函数可能用到某全局变量，则不宜再定义一个同名的局部变量。！！！！

## 6.复合类型

复合类型：基于其他类型定义的类型。这里主要介绍了两种：引用和指针。

### 6.1引用：

为对象起了另外一个名字，引用类型引用另外一种类型。书写格式：&d 例如：

Int ival=1024;

Int &refVa=ival; //引用

注意：引用必须要初始化，指明是对谁的引用。在初始化变量时初始值会被拷贝到新建的对象中去，但是引用则会把初始值绑定在一起。一旦初始化完成引用和它的对象就会一直绑定在一起。！！！！！！！！ 引用并非对象，仅仅是一个已经存在的对象所起的别名。

6.2为引用赋值，就是对引用所绑定的对象赋值。获取引用的值就是获取引用所绑定的对象的值。以引用作为初始值其实就是以引用绑定的对象作为初始值。

6.3C++不允许定义引用的引用。因为引用本身不是一个对象，但是可以间接得来引用一个“引用：例如：

Int b=1024;

Int& a=b; //正确,a引用了b

Int& c=a; //正确，c引用了a 也就意味着都是代表了b这个变量

6.4引用的类型与它所绑定的对象类型严格匹配。而且引用只能绑定在对象上，不能绑定到表达式。例如：

Int& a=10 //错误 不能绑定在一个常量上

Double dval=3.14

Int& reve=dval //错误，引用的类型和对象的类型不匹配

### 6.2指针

指针是“指向”另外一种类型的复合类型。作用：对其他对象的间接访问。区别点：指针本身是一个对象，允许对指针赋值和拷贝，并且可以先后指向不同的对象。第二：指针无需在定义时候初始化，如果不被初始化，也将拥有一个不确定的值。语法说明：

Int \*ip1,\*ip2; //定义指向Int类型的两个变量

Double dp,\*dp2; //定义一个double型变量和指针

获取对象的地址： 指针存放的是地址，如果想获取到一个变量的地址需要用到取地址符：&

Int val=42; //定义一个int类型变量

Int \*p=&val; //定义一个指针，获取到val的地址

注意：指针所指向的类型应该严格与对象的类型相一致，除了几种特殊情况外。

指针的值应该属于以下四种状态之一：

* + - 1. 指向一个对象
      2. 指向紧邻对象所占空间的下一个位置
      3. 空指针，没有指向任何对象
      4. 无效指针 //除了第一种，其他的如果出现问题后果将无法预计

利用指针访问对象：需要用到解引用符：\* 来访问该对象

Int ival=42;

Int \*p=&ival;

\*p=42;

对指针接引用会得到该指针所指向对象的值。

注意：解引用操作仅适用于那些确实指向了某个对象的有效指针。！！！！！

### 6.3空指针：

不指向任何对象的指针。下列几种生成空指针的方法：

Int \*p1=nullptr //最直接的办法

Int \*p2=0;

Int \*p3=NULL

空指针可以转换成任意其他的指针类型。但是，不允许给指针赋一个值为0的变量

int pr=42;

int a=0;

int \*value=a; //会报错

int pr1=41;

value=&pr1;

*cout*<<\*value<<*endl*;

*system*("pause");

预处理器：运行于编译过程之前的一段程序。当用到一个预处理变量时，预处理器会自动地将它替换为实际值。

建议：定义时就初始化所有的指针！！！

赋值和指针:引用当绑定一个变量时就不允许再次绑定一个对象，而且引用并不是对象，只是一个别名，但是指针可以存放一个新的地址，从而指向一个新的对象。

方法：当你不知道是指针所指向的对象的值发生了改变还是指针的值时，记住赋值永远改变的是等号左侧的对象！！

其他指针操作：把指针用来进行判断，放到bool表达式中，如果指针的值为0则是false，非0则是true。记住，无论作为条件还是参与比较运算，都必须使用合法指针，否则将产生不可估计的后果。

### 6.4 Void\*指针：

是一种特殊指针类型，用于存放任意对象的地址。例如：

Double obj=3.14,\*pd=&obj

Void \*pv=&obj;

Pv=pd;

利用void\*指针能做的事情比较有限：和别的指针比较，作为函数的输入输出，或者给另外一个void\*指针赋值，并不能直接操作void\*所指的对象，因为不知道到底是什么类型。

也就是说void\*所看内存空间就是内存空间，没办法访问其中的对象。

指向指针的指针：\*\*表示指向指针的指针。

Int ival=1024;

Int \*pi=&ival; //指向Ival

Int \*\*ppi=&pi; //指向了pi

如果想得到ival的值就需要将Pi进行一次解引用，将ppi进行两次解引用。

指向指针的引用：

因为引用不是一个对象，因此指针不能指向引用，但是指针是一个对象，因此可以有指针的引用：

Int i=42;

Int \*p;

Int \*&r=p;

r=&I;

\*r=0; //得出i=0

关键：要清楚r到底是什么含义，给出的建议就是从右往左阅读，离变量最近的那个修饰符将决定了该变量的含义，因此这里&最近，代表的是引用，其他的则是表示了引用的类型是个指针类型！！！！！！

## 7.Const限定符（第一次迷糊的地方）

Const关键字用来对变量的类型加以限定。注意：是对类型加以限定，就变成了一个常量，任何试图改变值的行为都将引发错误。  
 Constint bufSize=512; //将bufSize定义为512

注意：const对象必须初始化，创建后其值就不能再改变。其初始化可以是任意复杂的表达式。 Const int i=get\_size()

Const int j=42;

初始化和const：要知道，const对象和非const对象的操作有稍微的区别，其中只要不改变const的值的操作都是成立的，包括利用const对象去给另外一个变量赋值，只要不改变值就都是成立的。因为赋值是拷贝对象，只要拷贝过去之后两个对象便没有什么联系了。

如果想让const对象在多个文件中共享，那么就在const关键字前加上extern关键字即可，不管是声明还是定义，只要加上extern便可实现共享。因为const默认实在本文件中使用的。因此考虑到多个文件时就需要加上关键字extern了。

### 7.1const的引用

可以把引用绑定到const对象上，称之为：对常量的引用。对常量的引用不能用作修改它所绑定的对象。

Const int ci=1024;

Const int &r1=ci; //正确引用

ri=42; //错误，不能改变其值

Int &r2=ci; //错误，让一个非常量引用来指向一个常量对象

如果成立的话就可以利用r2来改变ci的值，这是不正确的。

也就是对const的引用必须是由const类型的引用。

### 7.2初始化和对const的引用

之前说到引用的类型必须与所引用的对象类型一致，以下是几点例外：

Int i=42;

Const int &r1=i; //引用一个普通Int对象身上

Const int &r2=42; //常量引用

Const int&r3=r1\*2 //可以进行转换得到int类型的引用

Int &4=r1\*2 //错误，因为不是常量引用

### 7.3对const的引用可能引用一个并非const的对象

Int i=42;

Int &r1=I; //绑定OK

Const int &r2=I; //const引用的并非是const对象，而是一个变量

R1=0; //可以改变 OK

R2=0; //不能改变，因为是常量引用

### 7.4指针和const

要想存放常量的地址，只能使用指向常量的指针。 例如：

Const double pi=3.14; //常量

Double \*ptr=&pi; //错误，必须是常量指针

Const double \*cpr=&pi; //正确

\*cptr=42; //不允许修改值

注意:指向常量的指针可以指向一个非常量对象：

Double dval=3.14; //常量

Cptr=&dval; //正确，但是不能改变值，不过可以通过其他方式改变

这么理解：这些指向常量的指针以为自己指向了常量，所以不能也不会去改变指向对象的值

### 7.5const指针

常量指针必须初始化，初始化完成后其值不能再改变。把\*放在const关键字之前说明指针是一个常量：不变的是指针的值，而不是指向的那个值。

Const指针：以为着必须该指针一直指向该对象，但是可以通过该指针来修改指向对象的那个值，除非该对象也是一个常量对象。例如：  
 int errNumb=0;

Int \*const curErr=&errNumb; //curErr将一直指向该对象  
 const int pi=3.1415926;

Cont int \*const pip=&pi; //指向的对象和对象的值都不能改变

上例中，因为errNum是个可以改变的值，因此可以通过设置curErr的指针来改变errNumb的值，但是因为pi是常量，并且指针也是常量，所以无法改变这两个对象中的值。

注意：不允许用非const指针来指向一个const变量，会报错。必须用const指针来指向const变量才OK。.

### 7.6顶层const

顶层const和底层const的含义：当一个const对象是指针本身的时候：意味着只能指向该对象，那么此const就是顶层const。如果一个const对象是对象本身值的时候那么就叫做底层const。

总结归纳const与指针的几种用法：

1. Const修饰数据类型的时候代表是一个常量。
2. Const在变量之前，\*之后意味只能指向该对象的指针。
3. Const既修饰数据类型又修饰变量，意味着这是一个只能指向该

变量的常量指针，意味着两个值都不能改变。因为指针是个常量，并且所指向的对象也是个常量。

### 7.7constexpr和常量表达式

常量表达式：是指不会改变，并且在编译阶段就能得到计算机结果的表达式。

一个对象是不是常量表达式由它的数据类型和初始值共同决定。例如：

Const int max\_files=20; //max\_file2是常量表达式

Const int limit=max\_files+1 //是常量表达式

Int staff\_size=77; //不是常量表达式

Const int sz=get\_size() //不是

其中第三个是因为其只是一个普通类型，而不是const int。

第四个是因为在编译阶段得不到值，知道运行时才行，所以不是常量表达式。

Constexpr:C++11新标准。允许将变量声明为Constexpr类型以便由编译器来验证变量的值是否是一个常量表达式。使用constexpr声明的变量一定是一个常量。而且必须用常量表达式初始化。！！！！

Constexpr int i=20; //没问题

Constexpr int a=i+1; //没问题

Constexpr int sz=size();

注意：只有当size是一个constexpr函数时才是一条正确的声明语句。一般来说，如果认定变量是一个常量表达式，就声明为constexpr类型.

7.8字面值类型

字面值类型:当你用constexpr定义的常量表达式时,其数据类型也需要做一些严格的限定,这样的数据类型叫做字面值类型.这类类型比较简单,值也显而易见,容易得到.

注意:指针和引用都可以定义成constexpr,但是有严格限定,constexpr指针的初始值必须是null或者0或者是存储与某个固定地址中的对象.

如果constexpr声明中定义了一个指针,那么constexpr仅对指针有效,与指针所指的对象无关,例如:

Const int \*p=nullptr;

Constexpr int \*q=nullptr;

两者不同,第一个是底层指针,而第二个是顶层指针.意味着q是一个只能指向该对象的指针. 有个注意的要点,其变量都必须要定义于函数体之外.

### 7.8处理类型

Typedef:用来为数据类型命名的关键字.

Typedef double wages;

Typedef wages base,\*p //其中p是double\*的同义

新标准的规定:使用”别名声明”来定义类型的别名.

Using SI=sales\_item; 将SI取为sales\_item的别名

SI item Using 和 typedef含义差不多.

Auto类型说明符:

Auto会自动推断出来类型,但是必须要有初始值.

同时Auto可以在同一个语句中声明多个变量,但是必须是同一类型

Auto i=0,\*p=&I;

Auto sz=0.pi=3.14 //错误,这是两种数据类型

Auto一般会忽略掉顶层const,底层const则会保留下来。

Declype类型指示符

作用：选择并返回操作数的数据类型，此时编译器分析表达式并得到类型，但是不实际计算表达式的值。

自定义数据结构

C++允许用户以类的形式自定义数据类型，库类型string,istream,ostream都是以类的形式定义的。

预处理器：

作用是确保多次包含了头文件但是仍然能安全工作。当预处理器看到#include标记时就会用指定的头文件内容代替#include。

头文件保护符：  
 #ifndef SALES\_DATA\_H

#define SALES\_DATA\_H

.

.

#endif

有几个if，就有几个endif，头文件保护符只要习惯性地添加上就好，不用太在乎是否需要。

# 第三章 字符串 向量和数组

## 3.1string标准库类型

String表示可变长的字符序列，必须先包含string头文件。

String的初始化方式有两种，拷贝赋值和直接赋值。例如：

String s1(s2) 直接赋值，没有等号，s1是s2副本

String s2=s1 把s1拷贝给s2

当用到一个数字，一个字符时只能使用直接初始化。

String s4(2,’c’);

String对象上的操作

一般对象可以有以下操作：os<<s //将s的写到os中去，返回os

Is>>a 从is中读取字符串给a，返回is

读写string对象

1.利用IO操作，cin >> s ;

读取未知数量的string对象

While(cin>>word)

Cout<<word<<endl;

使用getline读取一整行

希望最终得到的字符串保留输入时的空白符，就用到getline函数代替原来的>>

While （getline(cin,line)）

{

Cour<<line<<endl;

}