第一章 C++的初步知识

written by SCNU第一帅----->孔文康

C++的执行过程

1、C++语言

C++语言是高级语言。

用高级语言编写的程序称为“源程序”（source program）。

C++的源程序是以.cpp作为后缀的（cpp是c plus plus的缩写）。

2、源程序编译

计算机只能识别0和1组成的二进制指令。

C++程序想要让计算机识别并运行需要进行语言翻译（编译）。

编译系统把源程序编译成目标程序，让计算机识别并运行。

目标程序一般以.obj或.o作为后缀(object的缩写)。

编译的作用是对源程序进行查漏补缺（词法检查和语法检查）。

词法检查就是检查单词的拼写，语法检查就是检查语法的错误。

编译出错信息分为两类，一类是警告，一类是错误。

3、目标文件连接

全部通过编译后，得到一个或多个目标文件。

此时需要用系统提供的连接程序将这个程序的所有目标程序和系统的库文件以及系统提供的其他信息连接起来，最终成为一个供计算机运行的二进制文件，其后缀是.exe，可以直接执行。

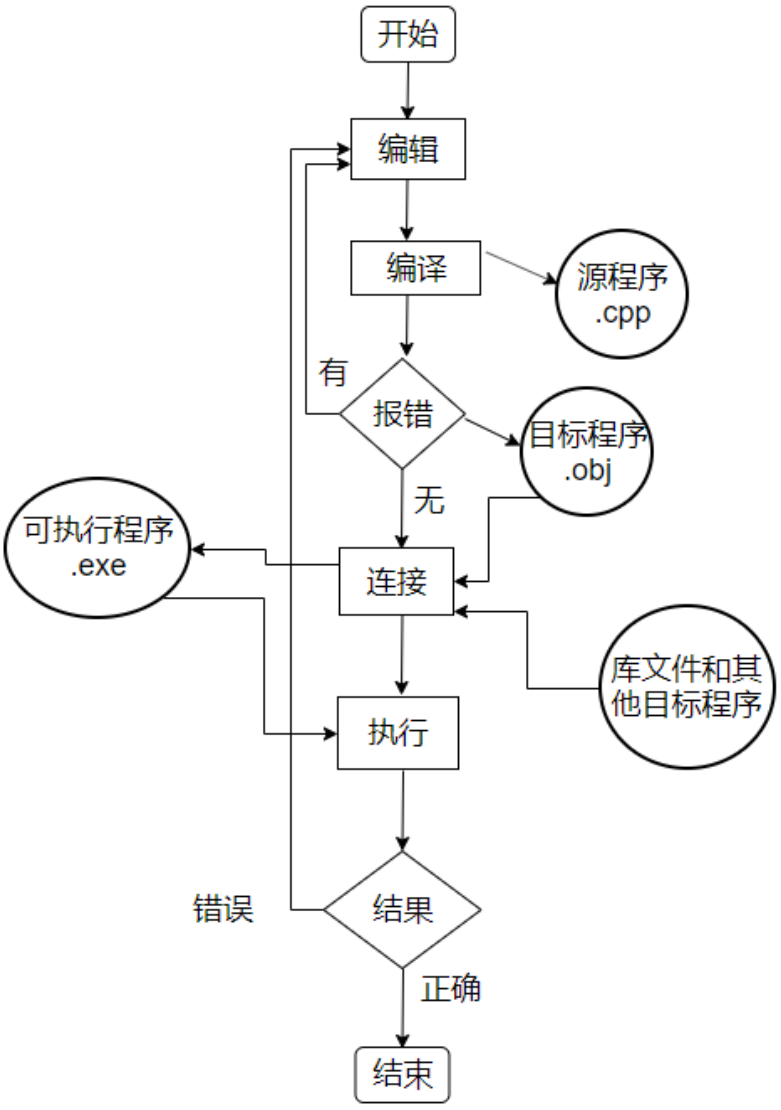
4、运行程序

运行最终得到的exe文件

5、运行程序

如果运行结果不正确 ， 对程序或算法进行检查和改正。

6、具体流程



setw函数

如果要指定输出所占的列数，可以用控制符setw进行设置，例如setw(5)的作用为其后面一个输出预留5列的空间，如果输出的数据项长度不足5列，则数据向右对齐，若超过5列，则按实际长度输出。

说明：如果使用setw,应当在程序的开头包含头文件iomanip(或iomanip.h)

用const定义常变量

C++提供了用const来定义常变量的方法，如

```
const float PI = 3.14159;
```

定义了**常变量PI**，它具有变量的属性，有数据类型，占用内存存储单元，有地址们可以用指针指向它。

只是在程序运行期间的值是固定的不能改变。

函数原型声明

在C++程序中，如果函数调用的位置在函数定义之前，则函数调用之前对所调用的函数作声明，但如果所调用的函数是整型，也可以不进行函数声明。

函数的声明形式**建议采用函数原型声明。**

例子

```
int max(int x, int y); //max函数原型声明
int max(); //不列出max函数的参数表
max(); //若max是整型函数，可以省略函数类型，但是为了可读性，一般不作如此声明
```

函数声明的一般形式为

函数类型　函数名（参数列表）

参数表中一般包括参数类型和参数名，也可以只包括参数类型而不包括参数名。

例子

```
int max(int x, int y);
int max(int, int);
```

在编译时，只检查参数类型，而不检查参数名。

函数的重载

C++中，函数可以重载。

C++允许在同一作用域中用同一函数名定义多个函数们这些函数的参数个数和参数类型不同，这些同名的函数用来实现不同的功能。这就是**函数的重载。**（即**一个函数名字多用**）

例子

```
int max(int a, int b, int c); //求3个整数中的最大者
float max(float a, float b, float c); // 求3个实数中的最大者
long max(long a, long b, long c); // 求3个长整数中的最大者
int max(int a, int b); //求2个整数中的最大者
```

函数模板

所谓函数模板，就是建立一个通用函数，其函数类型和形参类型不具体指定，用一个虚拟的类型去代表，这个通用函数就称为函数模板。

凡是函数体相同的函数都可以用这个模板来代替，不必定义多个函数，只需要在模板中定义一次即可。

在调用函数时，系统会根据实参的类型来取代模板中的虚拟类型，从而实现了不同函数的功能。

例子

```
template <typename T>
T max(T a, T b, T c) {
 a = b > a ? b : a;
 a = c > a ? c : a;
 return a;
}
int i;
double d;
long g;
 int i1 = 1;
 double d1 = 1.9;
 long g1 = 67843;
 i = max(i1, i1, i1); //调用此模板时，T被int替换
 d = max(d1, d1, d1); //调用此模板时，T被double替换
 g = max(g1, g1, g1); //调用此模板时，T被long替换
```

通用函数定义

template <typename T>

template含义是模板，尖括号中关键字typename后面跟一个类型参数T，这个T是虚拟的类型名可以自己定义，也可以多个虚拟参数，例如

```
template <typename T1, T2>
```

有默认参数的函数

一般情况下，在函数调用时形参会从实参那里取得值，因此实参的个数应与形参相同。（函数内的变量为形参，传进来的变量为实参）

但是有时候多次调用同一函数用的同样的实参，如果多次传入值，显得臃肿。

C++提供了简单的处理方法，给形参一个默认值，这样就不必一定要从实参取值了。如有一函数声明

```
int num(int a = 5);
```

指定a的初始默认值为5。如果在调用此函数时，知道实参的值为5的时候，则可以不必给出实参。例如

```
num(); //相当于nun(5)
```

如果不想让形参取默认值，则可以通过实参另行给出。例如

```
num(7); //形参得到的值为7，不是5
```

如果有多个形参，可以使每个形参有一个默认值，也可以只对一部分形参指定默认值。

如有一个求工资税的函数，形参s代表基本工资，r代表税率。函数原型如下：

```
float salary(float s, float r = 0.2); //默认最低工资的税率是20%
```

函数调用可以使用以下的形式:

```
 salary(1000.0); //相当于salary(1000.0, 0.2)
 salary(2000.0, 0.3); //相当于salary(2000.0, 0.3)
```

实参与形参的结合是从左到右顺序进行的，因此，指定默认值的参数必须放在形参列表的最右端，否则出错。例如：

```
float salary(float s, float r = 0.2); //正确
float salary(float r = 0.2, float s); //错误
```

注意事项：

1. 如果函数定义在函数调用之前，则应该在函数定义中给出默认值。
2. 如果函数定义在函数调用之后，则在函数定义之前要有函数声明，此时必须在函数声明中给出默认值。
3. 一个函数不能既作为重载函数，又作为带有默认参数的函数。

变量的引用

1、引用的概念

C++中，变量的引用，又称变量的别名。

因此，引用又称别名。

建立引用的作用是为一个变量再起另一个名字，以便在需要时候可以方便、间接地引用该变量，这就是引用名称的由来。

对一个变量的引用的所有操作，实际上都是对其所代表的变量的操作。

例如，有一个变量name, 想给他起一个别名sn,可以这样编写：

```
int name;
```

```
int &sn = name;
```

这就是声明了sn是name的引用，即name的别名。

这样声明后，使用sn或name的作用相同，都代表同一变量。

注意事项：

1. 在上述声明中，&是引用声明符，此时它并不代表地址。
2. 不要理解为把name的值赋值给sn的地址
3. 对变量声明一个引用，并不另开辟内存单元，sn和name代表同一变量单元。
4. 在声明一个引用时，必须同时使之初始化，也就是声明他代表哪一个变量。

请注意

当声明一个变量的引用后，在本函数执行期间，该引用一直与其代表的变量相联系，不能再作为其他变量的别名。下面的用法不对

```
int a1,a2;
int &b = a1; //使b成为变量a1的引用
int &b = a2; //又企图使b成为变量a2的引用
```

2、引用的简单说明

1. 引用并不是一种独立的数据类型，他必须与类型相同的某一种类型的数据相联系。
2. 声明引用时，必须指定它代表的是哪个变量，对他进行初始化

```
int &b = a; //正确，指定b是整型变量a的引用
int &b; //错误，没有指明b代表哪个变量
float a;int &b = a; //错误，声明b是一个整型变量的别名，而a不是
```

请注意

不要把声明语句”int &b = a;"理解为将变量a的值赋值给引用b，

它的作用是使b成为a的引用，即a的别名。

3. 引用与其所代表的变量共享同一内存单元，系统并不为引用另外分配内存空间。实际上，编译系统使引用和其代表的变量具有相同的地址。

```
int a = 3; //声明定义整型a
int &b = a; //声明b使整型变量a的别名
cout<<&a<<"->"<<&b<<endl; //输出a和b的地址
```

输出a和b的地址是相同的，这就表示了引用的地址就是其代表变量的地址，如果运用sizeof测量a和b的字节数，可以发现a和b的长度是相同的。

4. 当看到&a这样的形式时，怎样区别是声明式引用变量还是取地址的操作呢?

请记住

当&a的前面有类型符时(例如int &a)它必然是对引用的声明；如果前面没有类型符（例如p=&a)，此时的&是取地址运算符。

5. 引用在初始化后不能再被重新声明为另一变量的别名，如

```
int a = 3,b = 4; //定义a和b是整型变量
int &c = a; //声明c是整型变量a的引用
c = &b; //企图使c改变为整型b的别名，错误
int &c = b; //企图重新声明c为整型b的别名，错误
```

3、将引用作为参数函数

C++之所以增加引用，主要是利用它作为函数参数，以扩充函数传递数据的功能。

```
void swap(int &a, int &b) {
 int temp;
 temp = a;
 a = b;
 b = temp;
}
```

程序分析

在swap函数的形参中声明a和b是整型变量的引用。

实际上，实参传给形参的是实参的地址，也就是形参a和b与传进来的实参具有相同的地址，从而共享同一单元。

为了方便理解，我们把a,b称为实参的别名。

4、对引用的进一步说明

1. 不能建立void类型的引用，如

```
void &a = 9; //错误
```

2. 不能建立引用的数组，如

```
char c[6] = "hello";
```

```
char &rc[6] = c; //错误
```

企图建立一个包含6个元素的引用的数组，是不行的。

数组名c只代表数组的首地址，本身并不是一个占有存储空间的变量。

3. 可以将变量的引用地址赋给一个指针，此时指针指向的是原来的变量。

```
int a = 3; //定义a是整型变量
int &b = a; //声明b是a的别名
int *p = &b; //指针变量p指向变量a的引用b，相当于指向a，合法
// 相当于下面一行
int *p = &a;
```

如果输出*p的值，就是b的值，也就是a的值。

但是不能定义指向引用类型的指针变量不能写成

```
int & *p = &a; //企图定义指向引用类型的指针变量p，错误
```

由于引用不是一种独立的数据类型，因此不能建立指向引用类型的指针变量。

4. 可以建立指针变量的引用

```
int i = 5; //定义整型变量i，初始值为5
int *p = &i; //定义指针变量p，指向i
int * &pt = p; //pt是一个指向整型变量的指针变量的引用，初始化为p
```

由定义的形式可以看出，&pt表示pt是一个变量引用，它代表一个int *类型的数据对象（即指针变量），如果输出*pt的值，也就是*p的

值，也就是i的值5

5. 可以用const对引用加以限定，不允许改变该引用的值。

```
int i = 5; //定义整型变量i，初始值为5
const int &a = i; //声明常引用，不允许改变a的值
a = 3; //企图改变a的值，错误
```

请注意

但是它并不阻止改变引用所代表的变量的值。

```
i = 3; //合法
```

此时输入i和a的值都是3

这一特性在使用引用作为函数形参时是有用的，因为有时候希望保护形参的值不被改变。

6. 可以用常量或表达式对引用进行初始化，但此时必须用const作声明。

```
int i = 5;
const &a = i + 3; //合法
```

此时编译系统是这样处理的： 生成一个临时变量，用来存放该表达式的值，引用是该临时变量的别名。系统将const &a = i + 3;转换为

```
int temp = i + 3; //先将表达式的值放在临时变量temp中
const int &a = temp; //声明a是temp的别名
```

临时变量是内部实现的，用户不能访问临时变量。

用这种办法不但可以用表达式初始化，还可以用不同类型的变量对之初始化（要求能兼容的类型）

```
double d = 3.14159; //d是double类型变量
const int &a = d; //用d初始化a
```

此时编译系统将const int &a = d;编译为

```
int temp = d;//先将double类型变量d转换为int，存在temp中
const int &a = temp; //temp和a同类型
```

请注意

此时如果输出引用a的值，将是3而不是3.14159

如果在上面声明引用时不用const,则会发生错误。如

```
double d = 3.14159; //d是double类型变量
int &a = d; //未加const，错误
```

内置函数

调用函数时需要一定的时间，如果有些函数需要频繁使用，累计所用的时间会很长，从而降低程序的执行效率。

C++提供一种提高效率的方法，就是在编译时所调用的函数代码嵌入到主调函数中。

这种嵌入到主调函数中的函数称为内置函数，又称内嵌函数。

定义方法很简单，在函数首行的左端加一个关键字inline即可。

```
inline int max(int a, int b, int c) {
 a = b > a ? b : a;
 a = c > a ? c : a;
 return a;
}
```

请注意

使用内置函数可以节省时间，但却增加了目标程序的长度。

假设要调用10次max函数，则在编译时先后10次将max的代码复制并插入main函数，大大增加了main函数的长度。

因此，只有对于规模很小且使用频繁的函数，才可大大提高运行速度。

作用域运算符

每一个变量都有其有效的作用域，只能在变量的作用域内使用该变量，不能直接使用其他作用域中的变量。

例如

```
#include <iostream>
using namespace std;
float a = 11.11;
int main () {
 int a = 5;
 cout<<a;
 return 0;
}
```

运行结果
5

程序中有两个变量a，一个是全局变量a，浮点型。一个是main函数中的变量a，整型。

根据规定，作用域中的变量优先度>全局变量。
所以在main函数中使用变量a时，优先使用的是main函数中的整型变量a。

请注意

那有什么办法可以取到全局变量a的值呢?

C++中提供作用域运算符::，他能指定所需要的作用域。
例如如果想取全局浮点型变量a，可以使用::a。

字符串变量

除了可以使用字符数组处理字符串外，C++还提供了一种更简便的方法——
用字符串类型定义字符串变量。

实际上string并不是C++语言的基本类型，它是C++标准库声明的一个字符串类，用这种字符串类可以定义对象。

1、定义字符串变量

和其他类型变量一样，字符串变量必须先定义，后使用，定义字符串变量要用类名string。如

```
string string_a; //定义string_a为字符串变量
string string_b = "China"; //定义string_b同时对其初始化
```

可以看出，这和基本类型定义char,int,double,float等类型变量的方法是类似的。

请注意

要使用string类的功能时，必须在本文件的开头将C++标准库中的“string”头文件包含进来，如

```
#include<string> //注意头文件名字不是string.h
```

这一点是与定义基本数据类型不同的。

2、对字符串变量的赋值

在定义了字符串变量后，可以用赋值语句对它赋值以一个字符串常量。

```
string_a = "China";
```

而字符数组是不能这样做的：

```
char str[10];
str = "Hello!"; //错误
```

既可以用字符串常量给字符串变量赋值，也可以用一个字符串变量给另一个字符串变量赋值。

```
string_b = string_a; //假设string_b和string_a均已定义为字符串变量
```

请注意

在定义字符串变量时，不需要指定其长度，它的长度随其中的字符串长度而改变。

可以对字符串变量中某一字符进行操作，如

```
string word = "China"; //定义并初始化字符串变量word
word[2] = 'a'; //修改序号为2的字符，修改后word的值为"Chana"
```

请注意

字符串类只存放字符串本身，不包括’ \0 ‘

因此，字符串变量word中字符为“China”一共5个字符，而不是6个。

3、字符串变量的输入输出

可以在输入输出语句中用字符串变量名，直接输出字符串，如

```
cin>>string1; //从键盘输入一个字符串给字符串变量string1
cout<<string1; //将字符串变量string1输出
```

4、字符串变量的运算

1. 字符串复制用复制号

```
string1 = string2;
```

2. 字符串连接用+号

```
string string1 = "C++"; //定义string1并赋初始值
string string2 = "hello world!"; //定义string2并赋初始值
string1 = string1 + string2; //连接string1和string2
```

3. 字符串比较直接用关系运算符

可以直接用==（等于）、>（大于）、<（小于）、!=（不等于）、>=（大于或等于）、<=（小于或等于）等关系运算符来进行字符串比较。

5、字符串数组

不仅可以用string定义字符串变量，也可以用string定义字符串数组。

```
string name[5]; //定义一个字符串数组，包含5个字符串元素
// 定义一个字符串数组并初始化
string name[5] = {"ZHANG SAN", "LI SI", "WANG WU", "LAO LIU", "ZHENG QI"};
```

- 在一个字符串数组中，每一个元素相当于一个字符串变量。
- 并不要求每个字符串具有相同长度。

动态分配/撤销内存的运算符new和delete

在C++开发中，常常需要动态地分配和撤销内存空间。

```
// 开辟一个存放整数的空间，返回一个指向整型数据的指针
new int;
```

```
// 开辟一个存放整数的空间，并指定该整数的初始值为100
new int(100);
// 开辟一个存放字符数组的空间，该数组有10个元素
// 返回一个指向字符数据的指针
new char[10];
// 开辟一个存放二维整型数组的空间，该数组大小为5*4
new int[5][4];
 // 开辟一个存放实数的空间，并指定该实数初始值为3.14159
 // 将返回的指向实型数据的指针赋值给变量a
 float *a = new float(3.14159);
```

new运算符使用的一般格式为

new 类型[初值];

请注意

用new分配数组空间时候不能指定初值。

delete运算符使用的一般格式为

delete [] 指针变量

例如，撤销上面用new开辟的存放实数的空间。

```
delete p;
```

撤销用new char[10]开辟的空间，如果把返回的指针赋值给了指针变量ch，则可以这样撤销。

```
delete []ch;
```

开辟结构体存放结构体变量。（假设定义结构体Student）

```
Student * stu;
stu = new Student;
delete stu;
```

请注意

new和delete是运算符，不是函数，因此执行效率高。

new和delete要配合使用。