C++ Learning Notes

C++完全支持面向对象编程，包含面向对象编程的4个特点：

- 封装
- 继承
- 多态
- 抽象

实例解析：

```
#include <iostream>//头文件，使用系统提供的一些功能，其中#include被称为预处理指令
using namespace std;//针对命名空间的指令

int main()//程序入口，一个程序有且只有一个main()函数
{
   cout << "Hello World"endl; //cout是一个输出流对象，<<为输出操作符，endl表示换行
   return 0;//退出main()函数并以0作为返回值
}
```

C++ 数据类型和表达式

1. 基本的内置类型

类型名	长度（字节）	取值范围
bool	1	false，true
char	1	-128——127
signed char	1	-128——127
unsigned char	1	0——255
short（signed short）	2	-32768——32767
unsigned short	2	0——65535
int（signed int）	4	——

类型名	长度（字节）	取值范围
unsigned int	4	——
long（unsigned long）、signed long	4	——
float	4	——
double	8	——
long double	8	——

有符号整数在计算机内以二进制补码形式存储，最高位为符号位，0正1负；无符号整数只能是正数，在计算机内以绝对值方式存储。

2. 常量

C++中的常量主要有**整型常量，实型常量，字符常量，字符串常量和布尔常量。**整型常量可以加u，l等后缀，大小写随意，实型常量有一般形式和指数形式两种，字符常量是单引号括起来的一个字符，如'a','?'等，字符串常量是一对双引号括起来的字符序列，如‘abc’， 布尔常量只有两个：false和true。
在C++中，有两种定义常量的方式：

- 使用 #define 预处理器，如 `#define Width 10`
- 使用 const 关键字，如 `const int a = 10`，程序中间不能改变其值。
  注意：把常量定义成大写字母形式，是一种很好的编程习惯。
  还有一些字符是不可显示字符，C++提供了一种称为转义字符的表示方法：

字符常量形式	ASCII码	含义
\a	07	响铃
\n	0A	换行
\t	09	水平制表
\v	0B	垂直制表
\b	08	退格
\r	0D	回车

字符常量形式	ASCII码	含义
\	5C	字符‘\’
\'	27	单引号
\"	22	双引号

3. 变量

```
int num,total;
float v,t,h;
int a = 3;
char c = 's';
int a(3);
```

变量存储类型有以下几种：

1. auto类型，采用堆栈方式分配内存空间，属于暂时性存储，其存储空间可以被若干变量多次覆盖使用。
2. register类型，存放在通用寄存器中。
3. extern类型，在所有函数和程序段中都可引用。
4. static类型，在内存中以固定方式存放，在整个程序运行期间都有效。
   示例：`extern int d = 3, f = 5;`
   实例如下：

```
#include <iostream>
using namespace std;

// 变量声明
extern int a, b;
extern int c;
extern float f;

int main ()
{
  // 变量定义
  int a, b;
  int c;
  float f;

  // 实际初始化
  a = 10;
  b = 20;
  c = a + b;

  cout << c << endl ;

  f = 70.0/3.0;
  cout << f << endl ;

  return 0;
}
```

代码输出结果：

```
30
23.3333
```

同样的，在函数声明阶段，提供一个函数名，函数的实际定义可以在任何时候编写，示例如下：

```
// 函数声明
int func();

int main()
{
    // 函数调用
    int i = func();//声明需在调用之前
}

// 函数实际定义
int func()
{
    return 0;
}
```

4. 运算符与表达式

1. 算术运算符与算术表达式
   基本算术运算符：+ - * / %
   另外，i++与++i的区别（i++先人后己，++i先己后人）：

```
cout<<i++;//首先i自增为2，再输出i自增前的值1
cout<<++i;//首先i自增为2，然后输出i的当前值2
```

2. 赋值运算符与赋值表达式
   “="为赋值运算符，`n=b+5`为赋值表达式。
3. 逗号运算符
4. 逻辑运算与逻辑表达式
   `< == != >= <= ! && ||`
5. 条件运算符与条件表达式
   C++中唯一一个三元运算符是条件运算符‘？’，实现简单的选择功能：
   `表达式1？ 表达式2：表达式3`
   其中表达式1必须是bool类型，2和3可以是任意类型，且类型可以不同。求解顺序为：先判断表达式1的值，若为true，则计算表达式2，否则计算表达式3.示例：
   `a>b? a:b//求解a和b中的较小值`
   `cout<<(score>=60? "pass":"fail");`
6. sizeof运算符
   sizeof用于计算某种类型的对象在内存中所占的字节数。使用形式为：
   `sizeof(type_name)`

`sizeof 表达式`

7. 位运算

- 按位与&
- 按位或|
- 按位异或^
- 按位取反~
- 移位:左移<< 右移>>

8. 混合运算时数据类型的转换
   （1）隐含转换
   算术运算符、关系运算符、逻辑运算符、位运算符和赋值运算符这些二元运算符，要求两个操作数的类型一致。在算术运算和关系运算中，如果参与运算的操作数类型不一致，编译系统会自动对数据进行转换，转换的基本原则是将**低类型向高类型转换**，类型越高，能表示的数的范围就越大。隐含转换是安全的，因为在转换过程中精度没有损失。逻辑运算符要求参与运算的操作数必须是bool型，如果是其它类型，就会进行转换，规则是：非0为true，0为false。赋值运算要求左值与右值类型相同，如果不同，则一律将右值转换为左值的类型。
   （2）显式转换
   `Type(表达式)`
   `(Type)表达式`
   这两种写法只是形式上不同，功能上完全相同。
   注意：显式转换应注意以下两点：

- 这种转换可能不安全，可能会损失精度。
- 这种转换是暂时的，内存单元中本来的值并未改变。

5. 自定义数据类型

c++允许用户自定义数据类型，包括：枚举类型，结构类型，联合类型，数组类型，类类型等。

1. typedef声明
   在编写程序时，可以为一个已有的数据类型另外命名，提高程序可读性。类型声明的语法形式是：
   `type 已有类型名 新类型名;`
   其中，新类型名可以有多个标识符，之间以逗号隔开，即为一个已有数据类型声明多个别名。

2. 枚举类型enum
   如果将要处理的数据只有有限的几种可能值，则可以使用枚举类型，省去了对数据合法性的判断。枚举类型的声明格式如下：
   `enum 枚举类型名 {变量值列表}`
   对枚举类型按照常量处理，不能对其赋值。枚举类型具有默认值，依次为：0，1，2……也可以在声明时另行定义枚举元素的值。整数值不能直接赋给枚举变量，如需将整数赋值给枚举变量，应进行强制类型转换。

数据的输入与输出

I/O流

将数据从一个对象到另一个对象的流动抽象为“流”，流在使用前要被建立，使用后要被删除。从流中获取数据的操作称为提取操作，向流中添加数据的操作称为插入操作。cin和cout是预定义的流类对象，分别处理标准输入和输出。

预定义的插入符和提取符

“<<”是预定义的插入符，作用在cout上实现屏幕输出，格式如下：
`cout<<表达式1<<表达式2<<`
“>>”作用在cin上实现键盘标准输入：
`cin>>表达式1>>表达式2>>`
当使用cin和cout进行数据的输入和输出时，无论处理的是什么类型的数据，都能够按照正确的默认格式处理，但是经常还需要设置特殊的格式。I/O流类库提供了一些操作符，可以直接嵌入到输入输出语句中来实现格式控制，要使用操作符，必须在源程序开头包含“iomanip”的头文件。

基本控制结构

算法的基本控制结构有三种：顺序结构、选择结构、循环结构。

if语句实现选择结构

一个 if 语句由一个布尔表达式后跟一个或多个语句组成。

```
if(boolean_expression)
{
   // 如果布尔表达式为真将执行的语句
   //C 语言把任何非零和非空的值假定为 true，把零或 null 假定为 false。
}
```

实例如下：

```
#include <iostream>
using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // 使用 if 语句检查布尔条件
   if( a < 20 )
   {
       // 如果条件为真，则输出下面的语句
       cout << "a 小于 20" << endl;
   }
   cout << "a 的值是 " << a << endl;

   return 0;
}
```

代码输出结果：

```
a 小于 20
a 的值是 10
```

if-else语句实现选择结构

格式如下：

```
if(boolean_expression)
{
   // 如果布尔表达式为真将执行的语句
}
else
{
   // 如果布尔表达式为假将执行的语句
}
```

实例如下：

```
#include <iostream>
using namespace std;

int main ()
{
   // 局部变量声明
   int a = 100;

   // 检查布尔条件
   if( a < 20 )
   {
       // 如果条件为真，则输出下面的语句
       cout << "a 小于 20" << endl;
   }
   else
   {
       // 如果条件为假，则输出下面的语句
       cout << "a 大于 20" << endl;
   }
   cout << "a 的值是 " << a << endl;

   return 0;
}
```

代码输出结果：

```
a 大于 20
a 的值是 100
```

同时，一个if语句后面可跟多个else if语句，最后有一个else语句，逻辑与c语言相同，不再赘述。

switch语句实现选择结构

一个 switch 语句允许测试一个变量等于多个值时的情况。每个值称为一个 case，且被测试的变量会对每个 switch case 进行检查。switch语句格式如下：

```
switch (表达式1)
{
    case 常量表达式1: 语句1
    //break
    case 常量表达式2: 语句2
    //break
    ……
    default: 语句n+1//default语句是可选的
}
```

注意：每个case语句只是一个入口标号，并不能确定执行的终止点，因此每个case分支最后应该加一个break语句，用来结束整个switch结构，否则会从入口点开始执行一直到switch结构的结束点。同时当若干个分支需要执行相同操作时，可以使多个case分支共用一组语句。
实例如下：

```
#include <iostream>
using namespace std;

int main ()
{
   // 局部变量声明
   char grade = 'D';

   switch(grade)
   {
   case 'A' :
      cout << "很棒! " << endl;
      break;
   case 'B' :
   case 'C' :
      cout << "做得好" << endl;
      break;
   case 'D' :
      cout << "您通过了" << endl;
      break;
   case 'F' :
      cout << "最好再试一下" << endl;
      break;
   default :
      cout << "无效的成绩" << endl;
   }
   cout << "您的成绩是 " << grade << endl;

   return 0;
}
```

代码输出结果：

```
您通过了
您的成绩是 D
```

while语句实现循环结构

语法形式如下：

```
while(condition)
{
   statement(s);
}
```

执行顺序是：先判断表达式的值，若为true，则执行循环语句。因此**while语句可能一次都不会执行**。

do-while语句实现循环结构

do-while 循环是在循环的尾部检查它的条件，do-while 循环与 while 循环类似，但是 do-while 循环会确保至少执行一次循环。语法如下：

```
do
{
   statement(s);

}while( condition );//不要忘记分号
```

如果条件为真，控制流会跳转回上面的 do，然后重新执行循环中的 statement(s)。这个过程会不断重复，直到给定条件变为假为止。

for语句实现循环结构

for 循环允许编写一个执行特定次数的循环的重复控制结构。语法如下：

```
//init 会首先被执行，且只会执行一次。这一步允许声明并初始化任何循环控制变量。也可以不在这里写任何语句，只要
//接下来，会判断 condition。如果为真，则执行循环主体。如果为假，则不执行循环主体，且控制流会跳转到紧接着 fo
//在执行完 for 循环主体后，控制流会跳回上面的 increment 语句。该语句允许更新循环控制变量。该语句可以留空，
for ( init; condition; increment )
{
   statement(s);
}
```

注意：c++语法中break与continue的区别和c语言相同，不再赘述。同时goto语句的用法与格式也与c语言相同。

函数

可以把代码划分到不同的函数中，在逻辑上，划分通常是根据每个函数执行一个特定的任务来进行。函数声明告诉编译器函数的名称、返回类型和参数。函数定义提供了函数的实际主体。

函数的定义和使用

1. 函数定义的语法形式

```
return_type function_name( parameter list )
{
   body of the function
}
```

一个函数所有的组成部分：

- 返回类型：一个函数可以返回一个值。return_type 是函数返回的值的数据类型。有些函数执行所需的操作而不返回值，在这种情况下，return_type 是关键字 void。
- 函数名称：这是函数的实际名称。函数名和参数列表一起构成了函数签名。
- 参数：参数就像是占位符。当函数被调用时，您向参数传递一个值，这个值被称为实际参数。参数列表包括函数参数的类型、顺序、数量。参数是可选的，也就是说，函数可能不包含参数。
- 函数主体：函数主体包含一组定义函数执行任务的语句。
  实例分析(返回较大值)：

```
int max(int num1, int num2)
{
   // 局部变量声明
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

当您在一个源文件中定义函数且在另一个文件中调用函数时，函数声明是必需的。在这种情况

下，您应该在调用函数的文件顶部声明函数。

2. 函数调用

调用函数时，传递所需参数，如果函数返回一个值，则可以存储返回值。实例如下：

```
#include <iostream>
using namespace std;

int max(int num1, int num2);// 函数声明

int main ()
{
   int a = 100;
   int b = 200;
   int ret;
   ret = max(a, b);// 调用函数来获取最大值
   cout << "Max value is : " << ret << endl;
   return 0;
}

int max(int num1, int num2) // 函数体部分
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

代码输出结果：

```
Max value is : 200
```

3. 函数参数

如果函数要使用参数，则必须声明接受参数值的变量。这些变量称为函数的**形式参数。**形式参数就像函数内的其他局部变量，在进入函数时被创建，退出函数时被销毁。当调用函数时，有三种向函数传递参数的方式：

- 传值调用：向函数传递参数的传值调用方法，把参数的实际值复制给函数的形式参数。在这种情况下，修改函数内的形式参数不会影响实际参数。默认情况下，C++使用传值调用方法来传递参数。一般来说，这意味着函数内的代码不会改变用于调用函数的实际参数。
- 指针调用：向函数传递参数的指针调用方法，把参数的地址复制给形式参数。在函数内，该地址用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响

实际参数。当然，在函数声明时需要将参数声明为指针类型。

- 引用调用：向函数传递参数的引用调用方法，把引用的地址复制给形式参数。在函数内，该引用用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响实际参数。
  函数在定义时可以预先声明默认的形参值，调用时如果给出实参，则使用实参的值，如果没有，则使用形参的默认值。**有默认值的形参必须在形参列表的最后。**

函数的重载

两个以上的函数，具有相同的函数名，但是形参的个数或者类型不同，编译器会根据实参和形参的类型及个数的最佳匹配，自动确定调用哪一个函数，这就是函数的重载。
注意：重载函数的形参必须不同，个数或者类型不同，否则会编译报错。

C++系统函数

系统函数在**http://www.cppreference.com**中查询原型、头文件和用法。

数组

数组是可以存储一个固定大小的相同类型元素的顺序集合，数组的特定元素可以通过索引来访问。所有的数组都是由连续的内存位置组成。最低的地址对应第一个元素，最高的地址对应最后一个元素。

数组的声明与使用

数组类型声明的一般格式为：
`type arrayName [ arraySize ];`
如：`int a[10]`表示一个10个元素的一维数组；`int b[5][3];`表示一个5行3列的二维数组。
使用数组时，只能分别对数组的各个元素进行操作。数组的元素是由下标来区分的。数组的下标表达式可以是任何合法的算术表达式，其结果必须是整数。另外数组的下标不可越界。实例如下：

```
#include <iostream>
using namespace std;
int main() {
   int a[10],b[10];
   for (int i = 0;i < 10;i ++){
      a[i] = i * 2 - 1;
      b[10 - i - 1] = a[i];
   }
   for (int i = 0;i < 10;i ++){
      cout<<"a["<<i<<"]="<<a[i]<<" ";
      cout<<"b["<<i<<"]="<<b[i]<<endl;
   }
   return 0;
}
```

代码输出结果：

```
a[0]=-1 b[0]=17
a[1]=1 b[1]=15
a[2]=3 b[2]=13
a[3]=5 b[3]=11
a[4]=7 b[4]=9
a[5]=9 b[5]=7
a[6]=11 b[6]=5
a[7]=13 b[7]=3
a[8]=15 b[8]=1
a[9]=17 b[9]=-1
```

数组的初始化就是在声明数组时给部分或者全部元素赋初值。例如：

```
int a[3] = {1,1,2};
int b[] = {2,3,4,5};
int c[5] = {3,4,6,7};//对前4个元素赋值

int d[2][3] = {1,1,1,1,1,1};
int e[][3] = {1,2,1,2,1,2};
int f[2][2] = {{2,3},{4,5}};
```

数组作为函数参数

数组元素和数组名都可以作为函数的参数以实现函数间数据的传递和共享。可以使用数组元素作为调用函数时的实参，这其实和使用同类型变量完全相同。如果使用数组名作为函数的参数，则

实参和形参都应该是数组名，而且类型相同，**使用数组名传递数据时，传递的是地址**。形参数组和实参数组的首地址重合，后面的元素按照各自在内存中的存储顺序进行对应。把数组作为参数时，一般不指定数组第一维的大小，即使指定，也会被忽略。实例如下：

```
#include <iostream>
using namespace std;
void rowSum(int a[][4],int nRow) {
   for (int i = 0;i < nRow;i ++){
      for (int j = 1;j < 4;j ++){
         a[i][0] += a[i][j];
      }
   }
}
int main() {
   int table[3][4] = {{1,2,3,4},{2,3,4,5},{3,4,5,6}};
   for (int i = 0;i < 3;i ++){
      for (int j = 0;j < 4;j ++){
         cout<<table[i][j]<<" ";
        }
         cout<<endl;

   }
   rowSum(table,3);
   for (int i = 0;i < 3;i++){
      cout<<"sum of row"<< i <<" is "<<table[i][0]<<endl;
       }
      return 0;
}
```

代码输出结果：

```
1 2 3 4
2 3 4 5
3 4 5 6
sum of row0 is 10
sum of row1 is 14
sum of row2 is 18
```