1. 基本结构

#include "stdafx.h" //引入头文件，stdafx.h常用来存放一些常用且共用的的头文件，从而不必多次引入头文件

#include <iostream> //引入头文件，iostream是C++中用于数据的流式输入与输出的头文件

using namespace std; // 命名空间，真正项目中不建议这样写

int main() //主函数，是所有程序运行的入口

{

cout << "Hello World"; // 输出 Hello World

return 0; //终止 main( )函数，并向调用进程返回值 0。

}

在 C++ 中，分号是语句结束符。也就是说，每个语句必须以分号结束。它表明一个逻辑实体的结束。

C++以 // 作为行注释，多行注释以 /\* 开始，以 \*/ 终止

1. 基本的输入输出

**头文件<iostream>**，该文件定义了**cin、cout**、cerr 和 clog 对象，分别对应于标准输入流、标准输出流、非缓冲标准错误流和缓冲标准错误流。

**标准输出流（cout）**

举例：

#include "stdafx.h"

#include <iostream>

using namespace std;

int main()

{

int a = 0;

cout << a;

return 0;

}

**标准输入流（cin）**

举例：

#include "stdafx.h"

#include <iostream>

using namespace std;

int main()

{

int a = 0;

cout << a<<endl;

cin >> a;

cout << a<<endl;

system("PAUSE");

return 0;

}

**endl是标准C++的换行风格，使用**"\n"**也可以换行，但是这是C语言的语法风格，在两者都可以的情况下建议使用endl**

1. 基本类型

常用数据类型

整型 int 4 个字节 2^32

浮点型 float 4个字节 2^32

双浮点型 double 8个字节 2^64

字符型 char 1个字节 即2^8

布尔型 bool

无类型 void

1. 变量的命名规则、创建以及赋初值
   1. 变量名由数字字母下划线组成
   2. 必须以字母或下划线开头
   3. 不能与c++原有关键字冲突
   4. 变量名区分大小写

int i, j, k; //创建多个变量

float a=1,b=0; //创建多个变量并赋初值

1. 变量作用域
   1. 在所有函数外部声明的变量，称为全局变量。全局变量的值在程序的整个生命周期内都是有效的。全局变量可以被任何函数访问。也就是说，全局变量一旦声明，在整个程序中都是可用的
   2. 在函数或一个代码块内部声明的变量，称为局部变量，只能被函数内部或者代码块内部的语句使用。
   3. 在函数参数的定义中声明的变量，称为形参
   4. 局部变量和全局变量的名称可以相同,但是在不同情况下优先级不同

#include "stdafx.h"

#include <iostream>

using namespace std;

int a=0; //全局变量

void func(int a) //形参

{

a = 3;

}

int main()

{

int a=1; //局部变量

cout << a<<endl; // 调用局部变量，输出 1，因为此时局部变量的优先级高；若想调用全局变量需要加：：

system("PAUSE");

return 0;

}

1. #define宏定义

当执行到宏时进行替换，尤其注意有表达式的宏

常见格式：

#define LENGTH 10

#define LENGTH(x) x\*x

1. 常量

常量是固定值，在程序执行期间不会改变。常量可以是任何的基本数据类型，可分为整型数字、浮点数字、字符、字符串和布尔值。常量就像是常规的变量，只不过常量的值在定义后不能进行修改。不得重名。

const

const int LENGTH = 10;

举例：

#include "stdafx.h"

#include <iostream>

using namespace std;

#define LENGTH 10

int main()

{

const int WHITE = 10;

return 0;

}

1. Static静态变量

static 存储类指示编译器在程序的生命周期内保持局部变量的存在，而不需要在每次它进入和离开作用域时进行创建和销毁。因此，使用 static 修饰局部变量可以在函数调用之间保持局部变量的值。

static 修饰符也可以应用于全局变量。当 static 修饰全局变量时，会使变量的作用域限制在声明它的文件内。

#include "stdafx.h"

#include <iostream>

using namespace std;

int func()

{

static int num = 0;

return ++num;

}

int main()

{

cout << func() <<",";

cout << func() << endl;

system("PAUSE");

return 0;

}

输出1,2

1. 运算符
   1. 算术运算符

+，-，\*，/，%，++（[自增运算符](http://www.runoob.com/cplusplus/cpp-increment-decrement-operators.html)，整数值增加 1）, --（[自减运算符](http://www.runoob.com/cplusplus/cpp-increment-decrement-operators.html)，整数值增加 1）

* 1. 关系运算符

== 检查两个操作数的值是否相等，如果相等则条件为真。

!= 检查两个操作数的值是否相等，如果不相等则条件为真。

> 检查左操作数的值是否大于右操作数的值，如果是则条件为真。

< 检查左操作数的值是否小于右操作数的值，如果是则条件为真。

>= 检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。

<= 检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。

* 1. 条件运算符

Exp1 ? Exp2 : Exp3;

其中，Exp1、Exp2 和 Exp3 是表达式。请注意冒号的使用和位置。? : 表达式的值取决于 Exp1 的计算结果。如果 Exp1 为真，则计算 Exp2 的值，且 Exp2 的计算结果则为整个 ? : 表达式的值。如果 Exp1 为假，则计算 Exp3 的值，且 Exp3 的计算结果则为整个 ? : 表达式的值。

* 1. 逻辑运算符

&& 称为逻辑与运算符。如果两个操作数都非零，则条件为真。

|| 称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。

! 称为逻辑非运算符。用来逆转操作数的逻辑状态。

例子：

#include <iostream>

using namespace std;

int main()

{

int a = 5;

int b = 20;

int c;

if (a && b)

{

cout << "Line 1 - 条件为真" << endl;

}

if (a || b)

{

cout << "Line 2 - 条件为真" << endl;

}

/\* 改变 a 和 b 的值 \*/

a = 0;

b = 10;

if (a && b)

{

cout << "Line 3 - 条件为真" << endl;

}

else

{

cout << "Line 4 - 条件不为真" << endl;

}

if (!(a && b))

{

cout << "Line 5 - 条件为真" << endl;

}

getchar();

return 0;

}

运行结果：

Line 1 - 条件为真

Line 2 - 条件为真

Line 4 - 条件不为真

Line 5 - 条件为真

1. 赋值运算符

= 简单的赋值运算符，把右边操作数的值赋给左边操作数

+= 加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数

-= 减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数

\*= 乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数

/= 除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数

%= 求模且赋值运算符，求两个操作数的模赋值给左边操作数

1. 运算符优先级

在表达式中，较高优先级的运算符会优先被计算。下面将按运算符优先级从高到低列出各个运算符。

**类别 **

1. 循环
   1. While循环

当条件为真时执行循环。当条件为假时结束循环，继续执行紧接着While循环的下一条语句。

格式：

while (判断条件)

{

执行语句;

}

* 1. For循环

格式：

for ( init; condition; increment )

{

statement(s);

}

init 会首先被执行，且只会执行一次。这一步允许您声明并初始化任何循环控制变量。您也可以不在这里写任何语句，只要有一个分号出现即可。

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 for 循环的下一条语句。

在执行完 for 循环主体后，控制流会跳回上面的 increment 语句。该语句允许您更新循环控制变量。该语句可以留空，只要在条件后有一个分号出现即可。

条件再次被判断。如果为真，则执行循环，这个过程会不断重复（循环主体，然后增加步值，再然后重新判断条件）。在条件变为假时，for 循环终止。

举例：

for (int a = 10; a < 20; a = a + 1)

{

cout << "a 的值：" << a << endl;

}

1. 判断
   1. If，else if,else

任何**非零**和**非空**的值假定为 **true**，把**零**或 **null** 假定为 **false**。

格式：

If（判断条件）

{

语句块

}

else if（判断条件）

{

语句块

}

else

{

语句块

}

* 1. Switch

格式：

switch(expression){

case constant:

statement(s);

break; // 可选的

case constant:

statement(s);

break; // 可选的

// 您可以有任意数量的 case 语句

default : // 可选的

statement(s);

}

switch 语句中的 expression 必须是一个整型或枚举类型，或者是一个 class 类型，其中 class 有一个单一的转换函数将其转换为整型或枚举类型。

在一个 switch 中可以有任意数量的 case 语句。每个 case 后跟一个要比较的值和一个冒号。

case 的 constant必须与 switch 中的变量具有相同的数据类型，且必须是一个常量或字面量。

当被测试的变量等于 case 中的常量时，case 后跟的语句将被执行，直到遇到 break 语句为止。

当遇到 break 语句时，switch 终止，控制流将跳转到 switch 语句后的下一行。

不是每一个 case 都需要包含 break。如果 case 语句不包含 break，控制流将会 继续 后续的 case，直到遇到 break 为止。

一个 switch 语句可以有一个可选的 default case，出现在 switch 的结尾。default case 可用于在上面所有 case 都不为真时执行一个任务。default case 中的 break 语句不是必需的。

举例：

#include "stdafx.h"

#include <iostream>

using namespace std;

int main()

{

// 局部变量声明

char grade = 'B';

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;

getchar();

return 0;

}