网站：http://www.cplusplus.com/doc/

**一、变量和类型**

**1.1 变量-初始化变量**

|  |  |
| --- | --- |
| // initialization of variables  #include  using namespace std;  int main ()  { int a=5; // initial value: 5  int b(3); // initial value: 3  int c{2}; // initial value: 2  int result; // initial value undetermined   a = a + b;   result = a - c;   cout << result;  return 0; } | 6 |

**1.2 类型推倒（Type deduction）: auto and decltype**

When a new variable is initialized, the compiler can figure out what the type of the variable is automatically by the initializer. For this, it suffices to use auto as the type specifier for the variable:

|  |  |  |
| --- | --- | --- |
| 1 2 | int foo = 0; auto bar = foo; // the same as: int bar = foo; |  |

Here, bar is declared as having an auto type; therefore, the type of bar is the type of the value used to initialize it: in this case it uses the type of foo, which is int.  
  
Variables that are not initialized can also make use of type deduction with the decltype specifier:

|  |  |
| --- | --- |
| 1 2 | int foo = 0; decltype(foo) bar; // the same as: int bar; |

**1.3 初始化字符串**

string mystring = "This is a string";

string mystring ("This is a string");

string mystring {"This is a string"};

**二 、常量**

**2.1 整数**

|  |  |
| --- | --- |
| **Suffix** | **Type modifier** |
| u *or* U | unsigned |
| l *or* L | long |
| ll *or* LL | long long |

75 // int

75u // unsigned int

75l // long

75ul // unsigned long

75lu // unsigned long

**2.2 浮点数**

3.14159 // 3.14159

6.02e23 // 6.02 x 10^23

1.6e-19 // 1.6 x 10^-19

3.0 // 3.0

**三 运算符**

**3.1 递增和递减**

|  |  |
| --- | --- |
| **Example 1** | **Example 2** |
| x = 3; y = ++x; // x contains 4, y contains 4 | x = 3; y = x++; // x contains 4, y contains 3 |

Here there are some examples:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 | (7 == 5) // evaluates to false (5 > 4) // evaluates to true (3 != 2) // evaluates to true (6 >= 6) // evaluates to true (5 < 5) // evaluates to false |  |

Of course, it's not just numeric constants that can be compared, but just any value, including, of course, variables. Suppose that a=2, b=3 and c=6, then:

|  |  |
| --- | --- |
| 1 2 3 4 | (a == 5) // evaluates to false, since a is not equal to 5 (a\*b >= c) // evaluates to true, since (2\*3 >= 6) is true (b+4 > a\*c) // evaluates to false, since (3+4 > 2\*6) is false ((b=2) == a) // evaluates to true |

**3.2逗点运算符（，）**

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the right-most expression is considered.  
  
For example, the following code:

|  |  |
| --- | --- |
|  | a = (b=3, b+2); |

**结果：**a=5

**3.3 sizeof**

This operator accepts one parameter, which can be either a type or a variable, and returns the size in bytes of that type or object:

|  |  |  |
| --- | --- | --- |
|  | x = sizeof (char); |  |

Here, x is assigned the value 1, because char is a type with a size of one byte.

**四、标准输入\输出**

**4.1输入Standard input (cin)**

The extraction operator can be used on cin to get strings of characters in the same way as with fundamental data types:  
**输入单词，**

|  |  |  |
| --- | --- | --- |
| 1 2 | string mystring;  cin >> mystring; |  |

However, **cin extraction** always considers **spaces (whitespaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an** entire sentence.  
  
To get an entire line from cin, there exists a function, called **getline（输入句子）**, that takes the stream (cin) as first argument, and the string variable as second. For example:

|  |  |
| --- | --- |
| // cin with strings  #include <iostream>  #include <string>  using namespace std;  int main ()  { string mystr;   cout << "What's your name? ";   getline (cin, mystr);   cout << "Hello " << mystr << ".\n";   cout << "What is your favorite team? ";   getline (cin, mystr);   cout << "I like " << mystr << " too!\n";  return 0; } | What's your name? Homer Simpson Hello Homer Simpson. What is your favorite team? The Isotopes I like The Isotopes too! |

**五、控制流（while for switch等）**

**http://www.cplusplus.com/doc/tutorial/control/**

**六 函数和返回值**

**6.1 main函数的返回值**

ou may have noticed that the return type of main is int, but most examples in this and earlier chapters did not actually return any value from main.  
  
Well, there is a catch: If the execution of main ends normally without encountering a return statement the compiler assumes the function ends with an implicit return statement:

|  |  |  |
| --- | --- | --- |
|  | return 0; |  |

Note that this only applies to function main for historical reasons. All other functions with a return type shall end with a proper return statement that includes a return value, even if this is never used.  
  
When main returns zero (either implicitly or explicitly), it is interpreted by the environment as that the program ended successfully. Other values may be returned by main, and some environments give access to that value to the caller in some way, although this behavior is not required nor necessarily portable between platforms. The values for main that are guaranteed to be interpreted in the same way on all platforms are:

|  |  |
| --- | --- |
| **value** | **description** |
| 0 | The program was successful |
| [EXIT\_SUCCESS](http://www.cplusplus.com/EXIT_SUCCESS) | The program was successful (same as above). This value is defined in header [<cstdlib>](http://www.cplusplus.com/%3Ccstdlib%3E). |
| [EXIT\_FAILURE](http://www.cplusplus.com/EXIT_FAILURE) | The program failed. This value is defined in header [<cstdlib>](http://www.cplusplus.com/%3Ccstdlib%3E). |

Because the implicit return 0; statement for main is a tricky exception, some authors consider it good practice to explicitly write the statement.

**6.2传引用和传值**

|  |  |
| --- | --- |
| // passing parameters by reference  #include <iostream>  using namespace std;  void duplicate (int& a, int& b, int& c)  { a\*=2; b\*=2; c\*=2; }  int main ()  { int x=1, y=3, z=7;   duplicate (x, y, z);   cout << "x=" << x << ", y=" << y << ", z=" << z;   return 0; } | x=2, y=6, z=14 |

**6.3 默认值**

|  |  |
| --- | --- |
| // default values in functions  #include <iostream>  using namespace std;  int divide (int a, int b=2)   { int r; r=a/b; return (r); }  int main ()  { cout << divide (12) << '\n';   cout << divide (20,4) << '\n';   return 0;   } | 6 5 |

**七 重载和模板**

**7.1 重载（overloads）**

In C++, two different functions can have the same name if their parameters are different; either because they have a different number of parameters, or because any of their parameters are of a different type. For example:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | // overloading functions  #include <iostream>  using namespace std;  int operate (int a, int b)  { return (a\*b); }  double operate (double a, double b)  { return (a/b); }  int main ()  { int x=5,y=2;   double n=5.0,m=2.0;   cout << operate (x,y) << '\n';   cout << operate (n,m) << '\n';  return 0; } | 10 2.5 |

**7.2 模板**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | // overloaded functions  #include <iostream>  using namespace std;  int sum (int a, int b)  {  return a+b;  }  double sum (double a, double b)  {  return a+b;  }  int main ()  {  cout << sum (10,20) << '\n';  cout << sum (1.0,1.5) << '\n';  return 0;  } | 30  2.5 | [Edit & Run](http://www.cplusplus.com/doc/tutorial/functions2/) |

Here, sum is overloaded with different parameter types, but with the exact same body.  
  
The function sum could be overloaded for a lot of types, and it could make sense for all of them to have the same body. For cases such as this, C++ has the ability to define functions with generic types, known as *function templates*. Defining a function template follows the same syntax than a regular function, except that it is preceded by the template keyword and a series of template parameters enclosed in angle-brackets <>:  
  
template <template-parameters> function-declaration   
The template parameters are a series of parameters separated by commas. These parameters can be generic template types by specifying either the class or typename keyword followed by an identifier. This identifier can then be used in the function declaration as if it was a regular type. For example, a generic sum function could be defined as:

|  |  |
| --- | --- |
| 1 2 3 4 5 | template <class SomeType>  SomeType sum (SomeType a, SomeType b)  {  return a+b;  } |

**实际例子1：**

|  |  |
| --- | --- |
| // function template  #include <iostream>  using namespace std;  template <class T>  T sum (T a, T b)  {  T result;  result = a + b;  return result;  }  int main () {  int i=5, j=6, k;  double f=2.0, g=0.5, h;  k=sum<int>(i,j);  h=sum<double>(f,g);  cout << k << '\n';  cout << h << '\n';  return 0;  } | 11  2.5 |

**实际例子2：**

|  |  |  |
| --- | --- | --- |
| // function templates  #include <iostream>  using namespace std;  template <class T, class U>  bool are\_equal (T a, U b)  {  return (a==b);  }  int main ()  {  if (are\_equal(10,10.0))  cout << "x and y are equal\n";  else  cout << "x and y are not equal\n";  return 0;  } | x and y are equal | [Edit & Run](http://www.cplusplus.com/doc/tutorial/functions2/) |

**7.3 Non-type template arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| 2 3 4 5 6 7 8 9 10 11 12 13 14 | // template arguments  #include <iostream>  using namespace std;  template <class T, int N>  T fixed\_multiply (T val)  {  return val \* N;  }  int main() {  std::cout << fixed\_multiply<int,2>(10) << '\n';  std::cout << fixed\_multiply<int,3>(10) << '\n';  } | 20  30 | [Edit & Run](http://www.cplusplus.com/doc/tutorial/functions2/) |

**8Name Visible**

**http://www.cplusplus.com/doc/tutorial/namespaces/**

**d**

**d**

**d**