### 输入

cin >> a;

### 输出

cout << a; //输出变量

cout <<“你好，世界！” //输出文本信息

### 注释

// 单行注释

/\* \*/ 多行注释

### 变量

变量：可以更改的量

int、long long 整数

double 小数

变量的定义：数据类型+变量名;

int a;

变量的赋值

①int a = 5;

②int a;

a = 5;

### 常量

“常量”是指****值不可更改****的变量或数据

1. const int a = 100;
2. #define MAX\_SIZE 100 (不建议，容易出错)

### 标识符

****标识符****是编程语言中用于命名变量、常量、函数或其他实体的名称。

### 关键字

C++ 关键字是语言保留的词，用于表示特定的语法结构和功能，不能用作标识符名或标识符。

### 字符

char ch = ‘A’;

### 布尔

bool a = true;

true/任意非0值 真

false/0 假

### 转义

\n 换行

\t 跳一个tab

\\ 代表一个“\”

\0 空字符

### 字符串

string str = “Hello World!”

### 算术运算符

cout << 9 + 4; //13

cout << 9 - 4; //5

cout << 9 \* 4; //36

cout << 9 / 4; //2

cout << 9 % 4; //1

int a = 9;

a++; //a == 10

int b = 5;

b--; //b == 4

### 关系运算符

== 检查两个值是否相等  
 ！= 检查两个值是否不相等  
 > 检查左值是否大于右值

< 检查左值是否小于右值

>= 检查左值是否大于等于右值

<= 检查左值是否小于等于右值

### 逻辑运算符

&& 与 两者都为ture,结果才为ture

|| 或 两者都为false,结果才为false

！ 非 逆操作

### 赋值运算符

= int a = 3; //a为3

+= a += 3; //a = a + 3;

-= a -= 3; //a = a - 3;

\*= a \*= 3; //a = a \* 3;

/= a /= 3; //a = a / 3;

%= a %= 3; //a = a % 3;

### 位与：有0是0

cout << (3 & 5); //1

3： 011

5： 101

3&5： 001

### 位或：有1是1

cout << (3 | 5); //7

3： 011

5： 101

3&5： 111

### 位异或：同0异1

cout << (3 ^ 5); //6

3： 011

5： 101

3&5： 110

### 位非：补码基础上取反

|  |  |  |  |
| --- | --- | --- | --- |
| 整数（3/-3举例） | 原码 | 反码 | 补码 |
| 正数 | 原码 0000 0011 | 反码=原码 0000 0011 | 补码=原码 0000 0011 |
| 负数 | 原码 1000 0011 | 反码：符号位不变，其余取反 1111 1100 | 补码=反码+1 1111 1101 |

计算机的所有整数都以补码的形式进行存储，所以位非是在补码的基础上进行的。  
  
 cout << (~3); //-4

3的补码： 0000 0011

所以~3的补码： 1111 1100

~3的反码： 1111 1011

~3的原码： 1000 0100 //-4

### 右移

5： 0101

1. >2: 0001 //1

### 左移

5： 0101

5<<2: 10100 //20

### if语句

if(条件){

条件满足时执行的代码；

}  
if/else语句

if(条件){

条件满足时执行的代码；

}else{

条件不满足时执行的代码；  
 }

### else if语句

if(条件1){  
 条件1满足时执行的代码；

}else if(条件2){

条件2满足时执行的代码；  
}else{

条件1和条件2都不满足时的代码；

}

### switch语句

int day = 4;

switch(day){

case 1:

cout << “周一”;

break;

case 2:

cout << “周二”;

break;

.

.

.

.

.

.

case 7:

cout << “周七”;

break;

default：

cout << “无效”;

}

### while语句 int i = 1;

### while(i <= 5){ cout << “i = ” << i << endl; //1 2 3 4 5

i ++;

}

### do while语句

int i = 1;

do{  
 cout << “i = ” << i << endl; //1 2 3 4 5

i++;

}while(i <= 5);

### for语句

for(int i=1; i<=5; i++){

cout << “i = ” << i << endl; //1 2 3 4 5

}

int arr[5] = {1, 2, 3, 4, 5};

for(int i: arr){

cout << “i = ” << i << endl; //1 2 3 4 5

}

### break:跳出循环

for(int i = 1; i <= 5; i++){

if(i == 3) break;

cout << “i = ” << i << endl; //1 2

}

### continue:跳过本次循坏

for(int i = 1; i <= 5; i++){

if(i == 3) continue;

cout << “i = ” << i << endl;

}

### 数组概念

数组是一组相同数据类型数据的有序集合。它使用一块连续的内存空间，每个元素都可以通过下标访问。

——下标（索引）从0开始。

——数组元素必须一致。

### 数组定义

数据类型 数组名[数组长度]；

int arr[3];

### 数组初始化

int arr[3] = {1, 2, 3}; // 显式初始化

int arr2[] = {4, 5, 6}; // 自动推断长度为3

int arr3[5] = {7, 8}; // 未赋值元素自动为0: {7, 8, 0, 0, 0}

char str[6] = "hello"; // 字符串初始化，自动加\0

### 数组访问

通过下标访问和修改数组元素：  
 int a[3] = {10, 20, 30};

cout << a[0] << endl; // 输出10

a[1] = 50; // 修改第二个元素为50

### 数组遍历

通常用for循环。

int nums[5] = {1, 2, 3, 4, 5};

for (int i = 0; i < 5; ++i) {

cout << nums[i] << " ";} //1 2 3 4 5

for (int num : nums) {

cout << num << " "; //1 2 3 4 5

}

### 数组套数组（多维数组）

二维数组定义： int arr[2][3];

初始化: int arr[2][3] = {

{1, 2, 3}

{4, 5, 6}

};

访问: cout << matrix[1][2]; // 输出6

遍历二维数组：

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 3; ++j) {

cout << matrix[i][j] << " ";

}

}

### 函数概念

函数是执行特定任务的独立代码块，可以多次调用。通过函数，可以将程序模块化，提高代码的重用性与可维护性。

### 函数定义

返回值类型 函数名(参数列表) {

// 函数体

// ...

return 返回值; // 如果返回值类型为void，可以省略

}

示例：

int add(int a, int b) {

return a + b;

}

void printHello() {

cout << "Hello!" << endl;

}

### 函数的声明

函数声明用于在使用函数前，告诉编译器函数的返回类型、名称和参数类型。通常放在 文件开头或头文件中。

### 函数调用

int result = add(2, 3); // 调用add函数，返回5

printHello(); // 调用无参函数

### 函数传参

参数用于和外部数据交互。C++参数传递分为：

- \*\*值传递\*\*（默认）：传递参数的副本，不影响原变量

- \*\*引用传递\*\*（加&）：传递变量本身，可以修改原变量

示例：

void foo(int x) { x = 10; } // 值传递，不影响外部变量

void bar(int &y) { y = 10; } // 引用传递，会修改外部变量

int a = 5, b = 5;

foo(a); // a仍为5

bar(b); // b变为10

### 指针

int a = 20; //&a是变量a的地址

int \*p = &a; //变量a的地址赋值给p

cout << \*p; //Out: 20

### 指针常量：不能改变地址，但可以修改指针指向的数值

int a = 10, b = 20;

int \*const p = &a; // p是指针常量

\*p = 30; // 正确，能修改a的值

//p = &b; // 错误，不能改变p的指向

### 常量指针：指针指向的数据是常量，不能通过该指针修改数据内容但指针指向的地址可以变

int a = 10, b = 20;

const int \*p = &a; // 或 int const \*p = &a;

//\*p = 30; // 错误，不能通过p修改a

p = &b; // 正确，可以改变p的指向

### 常量指针常量：指针本身不能改变指向的地址，指向的数据内容也不能通过该指针修改

int a = 10, b = 20;

const int \*const p = &a;

//\*p = 30; // 错误，不能通过p修改a

//p = &b; // 错误，不能改变p的指向

### 指针函数：返回值为指针的函数

类型名\* 函数名(参数列表) { ... }

示例：

int\* getPtr(int x) {

static int y = x;//用静态变量是

return &y;

}

int \*p = getPtr(5); //Out: 5

### 函数指针:指向函数的指针，通过它可以间接调用函数

返回值类型 (\*指针变量名)(参数类型列表);

示例：

int add(int a, int b) { return a + b; }

int (\*funcPtr)(int, int) = add; // 声明并初始化函数指针

int result = funcPtr(3, 4); // 通过函数指针调用add

cout << result; //Out: 7

### 指针函数指针：一个指针指向一个返回值为指针的函数

返回值类型\* (\*指针变量名)(参数类型列表);

示例：

int\* foo(int x) {

static int y = x;

return &y;

}

int\* (\*pf)(int) = foo; // pf是指向返回int\*的函数的指针

int\* p = pf(5); // 调用foo函数，返回int\*指针

cout << \*p; //Out: 5

### 指针数组：元素为指针的数组，每个元素都是一个指针变量

示例：  
 int a = 10, b = 20, c = 30;

int\* arr[3] = {&a, &b, &c}; // arr[0]指向a，arr[1]指向b，arr[2]指向c

cout << \*arr[1]; // 输出20

### 数组指针：指向整个数组的指针

int arr[3] = {1, 2, 3};

int (\*p)[3] = &arr; // p指向整个arr数组，[3]为了避免非法访问不属于该 数组的内存地址

cout << (\*p)[2]; // 输出3

### 指针数组指针：指向元素为指针的数组的指针

int a = 10, b = 20, c = 30;

int\* arr[3] = {&a, &b, &c};

int\* (\*p)[3] = &arr; // p指向arr这个指针数组

cout << \*(\*p)[2]; // 输出30

### 结构体：struct

结构体是用户自定义的数据类型，可以包含不同类型的成员变量。

struct Person {

char name[20];

int age;

};

### 结构体指针：指向结构体变量的指针

Student stu = {1, "Tom", 98.5};

Student\* pStu = &stu; // 结构体指针

printf("%d %s %.1f\n", pStu->id, pStu->name, pStu->score);

// 用.是直接访问结构体成员，用->是通过结构体指针访问

### 结构体指针数组：元素为结构体指针的数组

Student stu1 = {1, "Tom", 98.5};

Student stu2 = {2, "Jerry", 95.0};

Student\* pArr[2]; // 结构体指针数组

pArr[0] = &stu1;

pArr[1] = &stu2;

// 访问printf("%s\n", pArr[1]->name); // Jerry

### 结构体数组指针：指向结构体数组首元素的指针

Student stus[3] = {

{1, "Tom", 98.5},

{2, "Jerry", 95.0},

{3, "Mike", 92.0}

};

Student\* pStus = stus; // 结构体数组指针

// 访问第二个元素

printf("%s\n", (pStus+1)->name); // Jerry