## 基础部分

### 输出语句:

cout << “hello world” << endl

### 注释

与js相同

单行注释: //

多行注释: /\* \*/

### 变量:

变量存在的意义: 给一段内存空间起个名字,方便我们管理内存空间

变量创建的语法: 数据类型 变量名 = 变量的初始值

#include<iostream>

using namespace std;

int main()

{

int a = 10;

cout << “a = ” << a << endl;

system(“pause”);

return 0

}

### 常量

作用: 记录程序中不可改变的数据

c++ 定义常量有两种方式:

1. # define 宏常量: #define 常量名 常量值
   1. 通常定义在文件上方
2. const修饰的变量 const 数据类型 常量名 = 常量值
   1. 通常在变量定以前加关键字const,修饰该变量为常量不可以更改

### 关键字

详细看菜鸟教程cpp保留字

<https://www.runoob.com/w3cnote/cpp-keyword-intro.html>

### 变量命名规则

1. 标识符(就是变量名) 不可以是关键字
2. 标识符是由字母,数字,下划线组成
3. 标识符第一个字符只能是字母或者下划线
4. 标识符是区分大小写的

### 数据类型:

c++规定在创建一个变量或者常量的时候必须指定出一个数据类型,否则复发给变量分配内存

数据类型存在的意义就是给变量分配一个合适的内存空间

#### 整数型

作用: 整型变量表示 整数类型 的数据

c++中能够表示的整型的类型有以下几种方式,区别在于所占内存空间不同

|  |  |  |
| --- | --- | --- |
| 数据类型 | 占用空间 | 取值范围 |
| short(短整型) | 2字节 | (-2^15 ~ 2^15-1)  -32,768 ~ 32,767 |
| int(整型) | 4字节 | (-2^31 ~ 2^31-1)  2,147,483,648‬ -  2,147,483,647 |
| long(长整型) | window为4字节  Linux为4字节(32位)8字节(64位) | (-2^31 ~ 2^31-1)  2,147,483,648‬ -  2,147,483,647 |
| long long(长长整型) | 8字节 | (-2^63 ~ 2^63-1)  9,223,372,036,854,775,808 –  9,223,372,036,854,775,807 |

#### sizeof 关键字

short num1 = 10;

int num2 = 10;

long num3 = 10;

long long num4 = 10;

cout << "short占用的字节" << sizeof(num1) << endl; // 2

cout << "int占用的字节" << sizeof(num2) << endl; // 4

cout << "long占用的字节" << sizeof(num3) << endl; // 4

cout << "long long占用的字节" << sizeof(num4) <<endl; // 8

#### 实型(浮点型)

作用: 表示小数

浮点型变量有两种:

1. float 单精度
2. double 双精度

两者的区别在于表示的有效数字范围不同(有效数字不光指小数点后面的数,也包含小数点前面的数)

|  |  |  |
| --- | --- | --- |
| 数据类型 | 占用空间 | 有效数字范围 |
| float | 4字节 | 7位有效数字 |
| double | 8字节 | 15-16位有效数字 |

#include<iostream>

using namespace std;

int main() {

// 默认情况下在C++中会显示6位有效数字,如果需要表示更多位的数字需要特殊配置(还没学到暂时不讲)

// 单精度: float

// 双精度 double

// 后面加f是为了告诉编译器 这个变量是单精度的浮点型,否则的话编译器会将3.14先转换为double类型 再转为float

float f1 = 3.1415926f;

cout << "f1 = " << f1 << sizeof(f1) << endl; // 4字节

double b1 = 1232133.1415926;

cout << "b1 = " << b1 << sizeof(b1) << endl; // 8字节

// 科学计数法

float f2 = 3e2; // 3 \* 10^2

cout << "f2 = " << f2 << endl;

float f3 = 3e-2; // 3 \* 0.1^2

cout << "f3 = " << f3 << endl;

system("pause");

return 0;

}

#### 字符型

作用: 字符型变量用于标识单个字符

语法: char ch = ‘a’

注意1 : 在现实字符型变量时,用单引号将字符括起来,不要用双引号

注意2: 单引号只能由一个字符,不可以是字符串

在C和C++字符型变量只占用一个字节

字符型变量并不是把字符放到内存中存储,而是将对应的ASCII编码放到储存单元

#### 转义字符

\a 警报 \n 换行 \t tab \\ 转义一个反斜杠

一个 \t占用八个空间 也就是说 如果 \t前面有三个字符 就会输出5个空格,如果\t前面刚好8个字符 则会输出8个空格

cout << sizeof(ch) << "asdasdsa\td" << "\n"; 奇奇怪怪为什么不用加endl 了

#### 字符串型

作用: 用于标识一串字符

两种风格

#include<iostream>

using namespace std;

#include<string> //使用c++风格字符串要引用这个头文件

void main() {

// c语言风格字符串

// 注意事项 char 字符串名[]

// 等号后面要用双引号包含起来字符串

char cstr[] = "你好世界";

cout << cstr << endl;

// c++风格字符串

// #include<string> 需要添加这个头文件

string cppstr = "helloworld";

cout << cppstr << endl;

}

#### Boolean布尔型

作用: 布尔型代表真或者假的值

bool类型只有两个值:

true – 真 (本质为1)

false – 假 (本质为0)

在c++中 布尔型除了0剩下都代表真

示例

#include<iostream>

using namespace std;

void main5() {

//1.创建布尔类型

bool t = true;

bool f = false;

// 输出 1 0

cout << t << "\t" << f << endl;

//查看内存空间

// 1 1

cout << sizeof(t) << "\t" << sizeof(f) << endl;

system("pause");

}

#### 数据的输入

作用: 用于从键盘中获取数据

关键字 cin

语法: cin >> 变量

示例

#include<iostream>

using namespace std;

#include<string>

void main() {

// 整型

int a = 0;

cout << "输入整型" << endl;

cin >> a;

cout << "输入的是" << a << endl;

// 浮点型

float f = 0;

cout << "输入浮点型" << endl;

cin >> f;

cout << "输入的是" << f << endl;

// 字符型

char ch = 'a';

cout << "输入字符型" << endl;

cin >> ch;

cout << "输入的是" << ch << endl;

// 字符串

string str = "";

cout << "输入字符串" << endl;

cin >> str;

cout << "输入的是" << str << endl;

//布尔型

bool flag = false;

cout << "输入布尔型" << endl;

cin >> flag;

cout << "输入的是" << flag << endl;

system("pause");

}

#### 运算符

作用: 用于执行代码的运算

|  |  |
| --- | --- |
| 运算符类型 | 作用 |
| 算术运算符 | 用于处理四则运算 +-\*/ |
| 赋值运算符 | 用于将表达式的值赋给变量 |
| 比较运算符 | 用于表达式的比较 返回一个真或者假 |
| 逻辑运算符 | 用于根据表达式的值返回真值或者假值 |

##### 算术运算符

用于处理四则运算

|  |  |
| --- | --- |
| 运算符 | 术语 |
| + | 加 / 正号 |
| - | 减 / 负号 |
| \* | 乘 |
| / | 除 // 整型相除结果还是整型 向下取整 分母为0直接异常 |
| % | 取余 //只有整型变量可以进行取余运算 // 取余除数不可以为0 // 两个小数不可以做取余运算 |
| ++ | 递增 |
| -- | 递减 |

##### 赋值运算符

|  |  |
| --- | --- |
| 运算符 | 术语 |
| = | 赋值 |
| += | 加等于 |
| -= | 减等于 |
| \*= | 乘等于 |
| /= | 除等于 |
| %= | 余等于 |

##### 比较运算符

|  |  |
| --- | --- |
| 运算符 | 术语 |
| == | 等于 |
| != | 不等于 |
| > | 大于 |
| >= | 大于等于 |
| < | 小于 |
| <= | 小于等于 |

##### 逻辑运算符

|  |  |
| --- | --- |
| 运算符 | 术语 |
| || | 或 |
| && | 且 |
| ! | 非 |

### 程序流程结构

c/c++支持最基本的三种程序运行结构: 顺序结构 , 选择结构 , 循环结构

顺序结构: 程序按照顺序执行 , 不发生跳转

选择结构: 根据条件是否满足, 有选择的执行相应的功能代码

循环结构: 根据条件是否满腹, 循环多次执行某段代码

#### 选择结构

##### if语句

作用: 执行满足条件的语句

if语句的三种形式

单行格式if语句

多行格式if语句

多条件的if语句

// 单行if语句

if (1 < 2) {

cout << "1比2小" << endl;

}

// 多行if语句

if (1 < 2) {

cout << "1比2小" << endl;

}

else {

cout << "1比2大" << endl;

}

// 多条件if语句

int c = 0;

cin >> c;

if (c > 600) {

cout << c << "\t" << 1 << endl;

}

else if ( 500 < c && c < 600) {

cout << c << "\t" << 2 << endl;

}

else{

cout << c << "\t" << 3 << endl;

}

##### 三元运算符

c = a > b ? a : b;

注意在c++中三元运算符返回的是变量可以继续赋值(返回的直接是变量而不是变量的值)

int a = 10;

int b = 20;

(a > b ? a : b)= 100;

cout << a << endl; // 10

cout << b << endl; // 100

##### switch

switch 缺点: 判断的时候只能是整型或者字符型,不可以是一个区间

switch 有点: 结构清晰,执行效率高

语法:

switch(表达式){

case 结果: 执行语句;break;

case 结果: 执行语句;break;

......

default: 执行语句;break;

}

示例:

switch (b)

{

case 10:

case 9:

cout << b << 1 << endl;

break;

case 8:

case 7:

cout << b << 2 << endl;

break;

default:

cout << b << "莫得啦 一滴也莫得啦" << endl;

break;

}

#### 循环结构:

##### while循环

作用: 满足循环条件执行循环语句

用法: while(循环条件){循环语句}

解释: 只要循环条件的结果为真,就执行循环语句

示例

int a = 0;

while (a < 10) {

a++;

cout << a << endl;

if (a == 8) {

break;

}

}

do

{

a++;

cout << a << endl;

} while (a < 10);

##### for循环

for (int i = 0; i < 10; i++)

{

cout << i << endl;

}

#### 流程控制

##### goto

作用 跳转到标题

#include<iostream>

using namespace std;

int main() {

cout << "祖安钢琴家开始了" << endl;

goto FLAG;

cout << "我\*\*你个\*\*" << endl;

FLAG:

cout << "李大炮真帅" << endl;

system("pause");

return 0;

}

### 数组

概述:

所谓数组就是一个集合,里面存放了相同类型的数据元素

特点: 数组中的每个元素都是相同的数据类型, 数组是由连续的内存位置组成的

一维数组的定义方式:

1. 数据类型 数组名[数组长度];
2. 数据类型 数组名[数组长度] = {值1. 值2. 值3}
3. 数据类型 数组名[] = {值1. 值2. 值3}

#### 一维数组

数组名是个常量 不可以进行赋值操作

#include<iostream>

using namespace std;

int main12() {

const int len = 3;

string strArr[len] = { "李大炮", "李二炮", "李三炮" };

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

cout << strArr[i] << endl;

}

// 一位数组数组名 的用途

//可以获取数组占用空间大小

// 通过占用空间的总大小处以 数组中的第0个算出数组总长度

cout << "总长度" << sizeof(strArr) / sizeof(strArr[0]) << endl;

// 可以获取数组中的首地址

cout << "数组首地址为" << strArr << endl;

cout << "数组首地址为" << (int)strArr << endl;

//查看数组中元素的地址

cout << "数组第一个元素的地址为" << &strArr[0] << endl;

cout << "数组第一个元素的地址为" << (int)&strArr[0] << endl;

// 寻找最大值

int arr[] = {300,350,400,250,370};

int len1 = sizeof(arr) / sizeof(arr[0]);

int id = 0;

int tizhong = 0;

for (int i = 0; i < len1; i++) {

if (arr[i] > tizhong) {

id = i;

tizhong = arr[i];

}

}

cout << "第" << id+1 << "个小猪最重,体重为" << tizhong << endl;

// 数组倒排

int arr2[] = { 1,2,3,4,5,6 };

int len2 = sizeof(arr2) / sizeof(arr2[0]);

for (int i = 0; i < (len2 % 2 !=0 ? len2/2 + 1 : len2/2); i++)

{

int z = arr2[len2-i-1];

arr2[len2 - i - 1] = arr2[i];

arr2[i] = z;

}

for (int i = 0; i < len2 ; i++) {

cout << arr2[i] << endl;

}

// 冒泡排序

int arr3[] = { 5,6,28,9,2,0,3 };

int len3 = sizeof(arr3) / sizeof(arr3[0]);

for (int i = 0; i < len3 - 1; i++)

{

for (int j = 0; j < len3 - 1 - i; j++) {

if (arr3[j] < arr3[j + 1]) {

int tem = arr3[j];

arr3[j] = arr3[j + 1];

arr3[j + 1] = tem;

}

}

}

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

cout << arr3[i] << endl;

}

system("pause");

return 0;

}

#### 二维数组

二位数组定义方式:

1. int arr [行数] [列数]
2. int arr [行数] [列数] = {{数据1,数据2},{数据3,数据4}}
3. int arr [行数] [列数] = {数据1,数据2,数据3,数据4}
4. int arr [] [列数] = {数据1,数据2,数据3,数据4}

#include<iostream>

using namespace std;

int main() {

/\*

自我理解: cpp需要分配相同的内存空间 所以行数可以不输入但是列数必须输入

数据类型 数组名 [行数] [列数]

数据类型 数组名 [行数] [列数] = {{数据1,数据2},{数据3,数据4}} \*推荐使用

数据类型 数组名 [行数] [列数] = {数据1,数据2,数据3,数据4}

数据类型 数组名 [] [列数] = {数据1,数据2,数据3,数据4}

\*\*\*\*\*\*\*\*\*\*\*

我喜欢使用

int arr1[][2] = { {1,2},{2,3} };

\*/

int arr[2][2];

int arr1[2][2] = { {1,2},{2,3} };

int arr2[2][2] = { 1,2,2,3 };

int arr3[][2] = { 1,2,2,3 };

int arr4[][2] = { {1,2},{2,3} };

// 二维数组数组名

// 查看二维数组所占用的空间

// 获取二维数组首地址

cout << sizeof(arr1) << endl;

cout << arr1 << endl;

cout << (int)arr1 << endl;

cout << "----------------" << endl;

cout << (int)&arr1[0][1] << endl;

cout << (int)&arr1[1][0] << endl;

cout << "----------------" << endl;

// 计算二位数组的行和列

cout << "二维数组的行" << sizeof(arr1)/ sizeof(arr1[0]) << endl;

cout << "二维数组的列" << sizeof(arr1[0]) / sizeof(arr1[0][0]) << endl;

// 算出每个数组的总和

// 创建数组

int result[][3] = {

{100,100,100},

{70,50,100},

{60,40,30}

};

// 算出行

const int hang = sizeof(result) / sizeof(result[0]);

// 算出列

int lie = sizeof(result[0]) / sizeof(result[0][0]);

//创建一个空数组

int sum[hang];

cout << "计算每行的总和" << endl;

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

// 创建一个临时总和

int temsum = 0;

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

temsum += result[i][j];

}

// 将临时总和放入数组中

sum[i] = temsum;

}

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

cout << "第" << i << "行的总和为 " << sum[i] << endl;

}

system("pause");

return 0;

}