C++入门

> 关于输入输出：
>
> 1、cin、cout：可以不指定输入输出格式，使用比较方便，但是耗时更长；
>
> 2、scanf、printf：使用规则更加复杂，但是耗时更短；
>
> 3、不要在同一程序中同时使用cout、printf；

一、基本数据类型：

类型	取值范围	大致范围
int（整型）	-2^{31}~$2^{31}-1$	**绝对值**在10^9范围内的整数
long long （整型）	-2^{63}~$2^{63}-1$	**绝对值**超过10^9的数就用long long
float（浮点型）	-2^{128}~$2^{128}-1$	有效精度6~7位
double（浮点型）	-2^{1024}~$2^{1024}-1$	有效精度15~16位
char（字符型）	-128~127	/
bool（布尔型）	0 or 1	/

注意：

- 碰到**浮点型**数据，都使用double存储即可；
- 字符常量必须用**单引号**括起来（%c）；字符串常量用**双引号**括起来（%s）；不能将字符串常量赋给字符变量，而是赋给字符串（**string类型**）；
- 小写字母比大写字母的ASCII码值**大32**；
- 转义字符：用 \ 加一些特定字母来表示；
- `“%.1f”`：浮点类型的数据，保留1位小数输出；
- 定义符号常量（又称宏定义）：`#define 标识符 常量`；**尽量不要使用宏定义来完成除定义常量之外的任何事情**；
- 定义 `const` 常量：`const 数据类型 变量名=常量;`

二、顺序结构：

1、赋值表达式：

2、输入输出：

- 在PAT考试/考研机试/算法比赛中，我们尽量使用 `scanf/printf` 来完成输入输出工作；
- 在 `scanf` 中，除了 **char 数组**（%s）整个输入的情况外，其他所有变量类型都要加&；
- `scanf` 是以**空白符**（空格、换行等）作为输入结束标志的；
- 对应double类型的数据，输入时使用“`%lf`”，输出时使用“`%f`”；
- 一些实用的输出格式：
  - `%md`：使不足m位的int型变量以m位进行右对齐输出，高位以空格补齐；
  - `%0md`：同 `%md` ，高位以0补齐；
  - `%.mf`：让浮点数保留m位小数输出；

3、使用 `getchar` 和 `putchar` 输入/输出单个字符：

4、注释：

5、typedef：用于给复杂的数据类型起一个别名；

6、常用的 `math` 函数：

- `fabs(double x)`：对 double 类型变量**取绝对值**，返回 double 类型；
- `floor(double x)` / `ceil(double x)`：对 double 类型变量**向下取整/向上取整**，返回 double 类型；
- `pow(double r, double p)`：用于执行**幂指数**运算，返回 double 类型；
- `sqrt(double x)`：用于执行**算术平方根**运算，返回 double 类型；
- `log(double x)`：用于执行**自然对数为底的对数**运算，返回 double 类型；
- `sin(double x)`、`cos(double x)`、`tan(double x)`：**三角函数**运算，参数要求是弧度制，返回 double 类型；
- `asin(double x)`、`acos(double x)`、`atan(double x)`：**反三角函数**运算，参数要求是弧度制，返回 double 类型；
- `round(double x)`：**四舍五入**运算，返回 double 类型；

三、选择结构：

if 语句、switch 语句

四、循环结构：

while语句、do...while语句、for语句、break、continue；

五、数组：

1、一维数组：

（1）数组大小必须是常量，**不能是变量**；

（2）只能访问下标为：0 ~ size-1的元素；

（3）冒泡排序：通过交换完成排序任务

2、二维数组：

（1）初始化：按第一维的顺序依次用大括号给出第二维初始化的情况，然后用逗号隔开，并用大括号全部括住。**未赋初值的元素，初始化为0**；

（2）如何数组大小较大（10^6 **级别**），则需要将该数组**定义在主函数外**；（函数栈帧的大小不足）

3、memset：对所有元素赋相同的值：

（1）函数格式：`memset(数组名，值，sizeof(数组名))`

（2）头文件：`string.h`

（3）只建议使用 `memset` 对数组元素**赋值 0 或 -1**；若要对数组赋其他数字，则使用 **fill 函数**；

4、字符数组：

（1）初始化：两种方式，一个元素一个元素地赋值，或直接输入字符串赋值（**仅限初始化**）；

（2）字符数组的输入输出：

- scanf 输入、printf 输出：%c、%s两种格式；
- getchar 输入、putchar 输出：每次输入、输出单个字符；
- gets 输入、puts 输出；（C++11已废除，推荐使用 `cin.getline` ）
  - gets 输入：用于输入一行字符串，**以换行符 \n 作为结束**；因此，**scanf 完一个整数后，要先用 getchar 接受整数后的换行符，然后再使用 gets**；
  - puts 输出：用于输出一行字符串，将一维数组在界面上输出，并**紧跟一个换行符。**

```
#include<iostream>
#include<string>
using namespace std;
int main(){                         // 字符串的输入输出（整行）
    char a[51];
    string b;
    // cin.getline(a, 51);
    // printf("%s", a);
    getline(cin, b);
    //cout<<b<<endl;
    printf("%s", b.c_str());
    return 0;
}
```

（3）字符数组的存放：

- 字符数组的末尾有一个空字符 **\0**，以表示存放的字符串的末尾，并占用一个字符位；
- 定义字符数组时，一定要比实际存储字符串的**长度至少多1**；
- 注意 \0 与空格不是一个东西；
- 如果使用 **getchar** 输入字符数组，一定要在每个字符串末尾加入 “\0”；

5、`string.h` 头文件：

- `strlen()`：获取字符数组中第一个 \0 前的字符的个数；

```
#include<iostream>
#include<string.h>
using namespace std;
int main(){
    string a;
    getline(cin, a);
    //cout<<a.length()<<endl;
    cout<<a.size()<<endl;
    return 0;
}
```

- `strcmp()`：返回字符串大小比较的结果，比较原则是：**按字典序**；

```
#include<iostream>
#include<string.h>
using namespace std;
int main(){
    char str1[51], str2[51];
    cin.getline(str1, 51);
    cin.getline(str2, 51);
    //printf("%s %s", str1, str2);
    if(strcmp(str1, str2) < 0) cout<<'<'<<endl;
    else if(strcmp(str1, str2) > 0) cout<<'>'<<endl;
    else cout<<'='<<endl;
}
```

- `strcpy(a1, a2)`：将字符串 a2 复制给另一个字符串 a1；
- `strcat(a1, a2)`：将一个字符串 a2 **接到字符串 a1 后面**；

```
#include<iostream>
#include<string.h>
using namespace std;
int main(){
    char str1[101], str2[51];
    cin.getline(str1, 51);
    cin.getline(str2, 51);
    // scanf("%s%s", s1, s2);
    strcat(str1, str2);
    printf("%s", str1);
    return 0;
}
```

```
// int main() {
//     string s1, s2;
//     cin >> s1;
//     cin >> s2;
//     cout << s1 + s2;
// }
```

6、sscanf 与 sprintf:

- 头文件：stdio.h
- sscanf：`sscanf(str, "%d", &n);`
    - 将字符数组 str 中的内容**以 “%d” 的格式**写到 n 中，顺序从左至右;

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

int main() {
    int a, b, c;
    char str[100];
    cin.getline(str, 100);

    if (sscanf(str, "%d is greater than %d", &a, &b) == 2) {
        cout << (a > b ? "Yes" : "No")<<endl;
    } else if (sscanf(str, "%d is equal to %d plus %d", &a, &b, &c) == 3) {
        cout << (a == b + c ? "Yes" : "No")<<endl;
    } else {
        cout << "???"<<endl;
    }
    return 0;
}
```

- sprintf：`sprintf(str, "%d", n);`
    - 将 n **以“%d”的格式**写到str数组中，顺序从右至左;

```
#include<iostream>
using namespace std;
int main(){
    int year, month, day, hour, minute, second;
    char str[30];
    cin>>year>>month>>day>>hour>>minute>>second;
    sprintf(str,"%04d-%02d-%02d %02d:%02d:%02d",  year, month, day, hour,
minute, second);
    cout<<str;
    return 0;
}
```

具体可以参考《算法笔记》P53、P54;

六、函数：

1、函数的定义：

（1）全局变量：

（2）局部变量：

（3）实参与形参：

2、数组作为函数参数：

- 数组作为函数参数时，只需填写数组名即可。
- 函数形参中：一维数组不需要填写长度，二维数组的**第二维需要填写长度**；
- 数组作为参数时，**在函数中对数组元素的修改等同于对原数组元素的修改**（区别于普通局部变量）；
- 数组可以作为函数参数，但是不能作为返回类型；

3、函数的嵌套、递归调用：

七、指针：

1、什么是指针：

（1）指针：指针是一种**特殊的变量**（一个 unsigned 类型的整数），它**存储了另一个变量的地址**，从而可以访问该变量所在的内存地址上存储的数据。

（2）变量地址：如何获取变量地址？使用**取地址运算符&。**在变量名前面加上&，就表示该变量的地址；

2、指针变量：

（1）定义指针变量：

- 在数据类型后紧跟一个**星号*，**用于表示该变量是一个指针变量；星号只结合**第一个**变量名；
- 对于 `int* p=&a;`，指针 p 的数据类型是 `int*` ，变量名是 p；因此，地址 &a 是赋值给 p 而不是 *p 的；即：**星号是类型的一部分**；

（2）如何获得 a 的值呢？

- p 存放了变量 a 的地址，* 就相当于一把钥匙，将其加在 p 的前面，这样 *p 就可以打开房门，取出变量 a 的值；
- **p 保存的是地址，*p 是该地址存放的元素**（直接对 *p 赋值，也能够修改变量的值）；

（3）指针变量的加减法：

- 参考程序计数器 PC 的加减法（指针的减法结果反映的是两个指针之间相差了几个数据存储基本单位，而不是地址的值之差）；
- 指针变量支持自增、自减操作；

（4）指针与数组：

- 数组名 a 可以作为数组的首地址 &a[0] 使用；
- **两个关键的等价关系：** `a + i = &a[i]` , `*(a + i) = a[i]`

（5）指针变量作为函数参数：将变量的地址传入函数（此时在函数中对该地址中的元素进行修改，原来的值也会随之修改）；

3、引用：

（1）引用：引用不产生副本，而是给原变量起了个**别名**；对引用变量的操作就是对原变量的操作。

（2）引用运算符：**&**，通常写在变量名的前面；通常在函数**定义形参**时使用；无论是否使用引用，函数的形参名与变量名都可以**不同**；

（3）要将引用的 & 与取地址运算符 & 区分开，引用并不是取地址的意思；

（4）针对指针的引用：

（5）引用产生的是变量的别名，因此**不可以对常量进行引用**；

八、结构体：

1、结构体的定义：

2、访问结构体内的元素：

- 有两种访问方式：“ . ”操作和“ -> ”操作；
- 对普通结构体变量的访问：`stu.id`；
- 对**结构体指针变量**的访问：`(*p).id`；

3、结构体的初始化：构造函数

（1）当结构体中变量很多不方便一一初始化时，我们引入**构造函数**：

- 一种初始化结构体的函数，**定义在结构体内**；
- 无需返回类型，且**函数名与结构体名相同**；
- 一般来说，对于一个普通的结构体而言，其内部会生成一个默认的构造函数（不可见）；

（2）构造函数+函数重载：

如果自己重新定义了构造函数，则不能不经初始化就定义结构体变量；这就意味着，原有的默认生成的构造函数已经被覆盖了。为了能够既不初始化就定义结构体变量，又能享受初始化带来的便捷，我们引入**函数重载**：只要构造函数的参数个数、类型**不完全相同**，就可以定义任意多个相同函数名的构造函数，以适应不同的初始化场合。

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

struct Point {
```

```
    int x, y;
    Point(){}                   // 不初始化就定义结构体变量
    Point(int _x, int _y){      // 同时初始化 x 和 y
        x = _x; y = _y;
    }
};

int main(){
    Point point;
    int x,y;
    cin>>x>>y;
    point = Point(x,y);
    cout<<point.x<<' '<<point.y<<endl;
    return 0;
}
```

```
struct Node {
    int id;
    Node* left;
    Node* right;
    Node(int _id, Node* _left, Node* _right){
        id = _id; left = _left; right = _right;
    }
};

int main(){
    int id1, id2, id3;
    cin>>id1>>id2>>id3;
    Node node1 = Node(id1, NULL, NULL);
    Node node2 = Node(id2, NULL, NULL);
    Node node3 = Node(id3, &node1, &node2);     // & 为取地址符
    cout<<node3.left->id<<' '<<node3.right->id<<endl;
    return 0;
}
```

九、补充：

1、cin：

（1）cin 支持同时读入多个变量：`cin>> score >> n >> student_name;` 其中，score 是 double 型、n 是 int 型、student_name 是 char 型。

（2）getline：支持一次性读入一整行。

```
char str[100];
cin.getline(str, 100);
```

如果使用了 string 容器，则修改读入方式如下：

```
string str;
getline(cin, str);
```

2、cout：

（1）cout 的使用方法与 cin 基本一致；**同时输出多个变量时，中间没有空格。**

（2）两种换行方式：使用 \n 来完成换行 或 使用 endl 完成换行；

3、浮点数的比较：

（1）C++中“ == ”执行的原理：要求左右两侧的表达式的值**完全相同**，才返回 True；这对于浮点数而言，显然很难做到，因此引入**极小数 eps** 来对误差进行修正。

（2）事实上，当我们对浮点数进行相等比较时，通常要求等号左右两侧的表达式的值**在同一个区间内，即可判定相等**，并定义极小数 eps 如下：

```
const double eps = 1e-8;            // eps 等于 10^(-8)
```

（3）浮点数如何判断相等：

- 首先定义极小数 eps（可以将极小数简单理解为浮点运算中的 0 ）；
- 对要判定相等的两个浮点型变量a、b**做减法，并求其绝对值**；
- 如果绝对值 **小于** 极小数eps，则说明浮点数 a = b；

（4）那么在上面的基础上，我们可以很容易地得到**判断 > 、 < 、>= 、<= 的方法**：

- 若 $a-b>eps$，则说明：$a>b$；
- 若 $a-b<(-eps)$，则说明：$a<b$；
- 若 $a-b>(-eps)$，则说明：$a>=b$；
- 若 $a-b<eps$，则说明：$a<=b$；

```
#include<iostream>
#include<cmath>
using namespace std;
const double eps = 1e-8;

bool Greater(double a, double b){
    return a -b > eps;
}

bool Less(double a, double b){
    return a - b < -eps;
}

int main(){
    double a, b, c, d;
    double re1, re2;

    cin>>a>>b>>c>>d;
    re1 = a*asin(sqrt(b)/2);
```

```
    re2 = c*asin(sqrt(d)/2);

    if(Greater(re1, re2)) cout<<"1"<<endl;
    else if(Less(re1, re2)) cout<<"2"<<endl;
    else cout<<"0"<<endl;
    return 0;
}
```