C++基本语法

C++基本语法
- Hello World
- 注释
- 数学函数
  - 随机数
- 常量变量
- 数据类型
  - 强制类型转换
- 输入输出
  - cin & cout
  - scanf & printf
  - sscanf&sprintf
  - 读取一行
  - 连续输入
- 算术运算符
- 分支结构
  - 关系表达式与逻辑表达式
  - 分支语句
- 循环结构
  - for语句和while语句
- 数组
  - 一维数组
  - 多维数组
- 字符串与文件操作
  - 字符数组
    - 字符数组的一些工具
  - string类型字符串
  - string与字符数组的互相赋值
    - string到字符数组
    - 字符数组到string
  - 文件操作与重定向
- 函数与结构体
  - 函数的定义方式
  - 变量作用域与参数传递
    - 全局变量与局部变量
  - 结构体
    - 成员函数与构造函数
- 尾声

Hello World

```cpp
#include<iostream>
using namespace std;
int main(){
    cout << "Hello World!";
    return 0;
}
```

注释

1. 单行注释：`//`
2. 多行注释：`/*..*/`

数学函数

```
#include<cmath>
```

函数原型	样例	说明
`double sin(double x)`		x是弧度
`double cos(double x)`		x是弧度
`double exp(double x)`		
`double log(double x)`		log是自然对数
`double pow(double x, double y)`		x^y
`double sqrt(double x)`		$\sqrt{x}$
`double fabs(double x)`		$\|x\|$
`double ceil(double x)`		上取整
`double floor(double x)`		下取整

随机数

rand()函数：产生一个0到RAND_MAX的整数，其中RAND_MAX是一个常数，与编译器和操作系统有关（Windows下为32767，Linux下为int的最大值）

可以用rand()%a来生成0~a-1的随机数

rand()是伪随机，需要种子。可以用srand(time(0))，用当前时间作为种子。

上述函数要导入`cstdlib`和`ctime`头文件

常量变量

常量的两种定义方式

1. `const double PI = 3.14159`
2. `#define PI 3.14159`

其中，`#define`是宏定义，程序编译的时候会先把代码中的所有`PI`都替换为`3.14159`

变量和常量命名遵循以下规则：

1. 只能由英文字母、数字和下划线(_)组成
2. 不能以数字开头
3. 不能和其他关键字重复

数据类型

数据类型	占用空间	取值范围
char	1byte,8bit	-128~127
int	4byte,32bit	-2^{31}~2^{31}-1，约绝对值不超过2.1×10^9的整数
unsigned int	4byte,32bit	0~2^{32}-1，约不超过4.2×10^9的非负整数
long long	8byte,64bit	-2^{63}~2^{63}-1，约绝对值不超过9.2×10^{18}的整数
unsigned long long	8byte,64bit	0~$2^{64}-1$，约不超过1.8×10^{19}的非负整数
float	4byte,32bit	约指数绝对值不超过37，6位有效数字
double	8byte,64bit	约指数绝对值不超过307，15位有效数字

强制类型转换

```
(int)3.2
```

输入输出

cin & cout

```
#include<iostream>
using namespace std;
char a, b;
cin >> a >> b;
cout << a << b;
```

用`cin`输入的时候，多余的空格和回车会被忽略。

`cin`大多数时候返回`cin`本身（非0值），遇到EOF（输入结束），返回0

`cout`后使用三元运算符（a?b:c）注意加上括号，因为`<<`和`>>`的运算级比`&&`或`||`要高。

> 三目条件运算公式为 **x?y:z** 其中x的运算结果为boolean类型，先计算x的值，若为true，则整个三目运算的结果为表达式y的值，否则整个运算结果为表达式z的值

scanf & printf

可以不写`using namespace std;`

```
#include<cstdio>
int main(){
    char a, b, c, d;
    scanf("%c%c%c.%c", &a, &b, &c, &d);
    printf("%c.%c%c%c", d, c, b, a);
}
```

这里，`scanf`会返回成功匹配的变量个数。`&a`的`&`表示取内存地址

占位符	说明

占位符	说明
%d	一个十进制整数
%nd(n是正整数)	输出一个整数，不足n位用空格补齐
%I64d(Windows),%lld(Linux)	一个十进制整数，一般用于long long类型
%f	读入一个float类型的带小数的浮点数，或输出float或double类型的浮点数，默认6位小数
%lf	读入double类型的浮点数
%.nf(n是正整数)	输出固定n位小数的浮点数
%0nd(n是正整数)	输出一个整数，不足n位的用0补齐
%c	一个char类型的字符
%s	一个字符串

一般用C语言风格的scanf和printf较多，且scanf读取速度更快

scanf会自动忽略空格。

sscanf&sprintf

`s`是一个变量。

`sscanf(s,"",);`

`sprintf(s,"",);`

就是改成从s中读取或输出到s。

读取一行

1. 字符数组

   使用fgets()函数，`fgets(s,sizeof(s),stdin)`，这里的sizeof(s)限定了最大读入数量，故不会造成数组越界。

2. string

   使用getline()函数，`getline(cin,s)`，cin是输入流。

这两个函数有个区别，fgets可以读到换行符，而getline不能，但两者都能读到`'\0'`。也就是比如某行输入了'a'，那么fgets其实读到的是'a'、'\n'、'\0'，而getline读到的是'a'、'\0'

因此，如果要在某个输入后再读取一行(无论用哪个函数)，一定要先把那个输入之后的换行符读掉，在scanf里可以写`scanf("%d\n",&a)`，也可以用`getchar()`读掉。

此外，我还发现一点，这两个函数都是半覆盖读取，也就是读到哪覆盖到哪。比如`char a[100]="asdasdsadasdadad...."`，然后我用fgets实际上只读了一个't'进去，那么只有头三个字符会被更新，后面的字符还是不变。但如果用cout或printf来输出，却只会输出't'，也就是刚读进去的内容，原来的没被覆盖的就好像被屏蔽了一样，没有消失，但也不会被引用。string类型的也是这样。这应该是因为字符串到`'\0'`就被认为结束了吧。

连续输入

`while(cin>>a)`

`cin`会在读完之后返回0，读取的时候返回比较复杂，反正非0。

算术运算符

1. +-*：加减乘
2. /：如果除数与被除数都是整型，则为整除，如果有一个是浮点型，就是除
3. %：模，即取余数
4. ++、--：自增1，自减1。注意，i++与++i是不一样的，i++是先读取i，再把i+1，而++i是先把i+1再读取
5. `a<<b`、`a>>b`、`&`、`|`、`^`：位运算，分别是a左移b位，0补齐，a右移b位，按位与，按位或，按位异或

分支结构

关系表达式与逻辑表达式

关系表达式：<、>、<=、>=、==、!=

优先级（高到低）：

()	*/%	+-	<、>、>=、<=	==、!=

逻辑运算符：

1. &&：与
2. ||：或
3. ^：异或
4. !：非

完整优先级：

()	!、-(负号)、++、--	*/%	+-	<<、>>	<、>、>=、<=	==、!=	&&	||

分支语句

```
if(situation1){
    state
}else if(situation2){
    state
}else{
    state
}
```

```
switch (opt) {
    case a:
```

```
        state
        break;
    case b:
        state
        break;
    case c: case d:
        state
        break;
    default:
        state
        break;
}
```

`opt`变量可以是整数和字符，但不能是浮点数（浮点数比较最好不要直接用==）

如果一个case语句执行完了没有`break;`，就会接着执行后面的语句，而不管case符不符合；

如果只有一条语句，可以不用花括号

循环结构

for语句和while语句

```
for(int i = 1; i <= n; i++){
    state
}
```

先执行`int i = 1`，如果`i <= n`成立，执行`state`，执行完以后执行`i++`，然后再判断`i <= n`.....

```
while(situation){
    state
}
```

如果只有一条语句，可以不用花括号，逗号表达式算1条语句。

> 逗号表达式. c语言 提供一种特殊的 运算符 ， 逗号运算符 ，优先级别最低，它将两个及其以上的式子联接起来，从左往右逐个计算表达式，整个表达式的值为最后一个 表达式 的值。. 如：（3+5,6+8）称为逗号表达式，其求解过程先表达式1，后表达式2，整个表达式值是表达式2的值，如：（3+5，6+8）的值是14；a=(a=35,a4)的值是60，其中a=（35,a4）的值是60， a的值在逗号表达式里一直是15，最后被逗号表达式 赋值 为60，a的值最终为60。.

```
do{
    state
}while(situation);
```

数组

一维数组

定义方法：`数组类型 数组变量名[元素个数]`

在main函数内定义的数组内的值是未知的，可以用零初始器把所有值赋为0，`int a[100] = {0};`

但注意，如果写`int a[100] = {1};`，并不会全赋值为1，只有第一项赋值为1

在main函数之外（即全局变量），默认为0。

注意：超大数组要开在全局，不然会爆栈

数组的索引是从0开始的

多维数组

定义方法：`数组类型 数组变量名[行数][列数][...]`

比如`int a[3][n]`，我们已经知道，数组是内存中一段连续空间，具体来说是`a[0][0]、a[0][1]....a[0][n-1]、a[1][0].....`

所以遍历数组`a[i][j]`的时候，先遍历`i`和先遍历`j`，在效率上有区别，因为遍历`i`是跳跃的，非连续的

字符串与文件操作

字符数组

顾名思义，`char s[110]`

读入字符串：

1. `scanf("%s",s);`：注意，这里的`s`不用`&`取内存地址，是因为`s`是数组名，本身表示内存地址。
2. `cin>>s;`

注意，这两种方法读取的字符串都不带空格或换行， 读到空格或换行就会中断，然后末尾会读到一个特殊字符`'\0'`（ASCII码为0），标记字符串的结束，所以s要开大1个。

还可以用`getchar()`和`putchar()`来读入或输出单个字符

`EOF`常量：End of File，控制台中可以用Ctrl+Z（Windows下）或Ctrl+D（Linux下）来输入EOF标记

字符数组的一些工具

这些工具是`cstring`头文件里的

1. `strlen`
2. `strcpy`：不能直接把字符串赋值给字符数组（初始化时除外），需要借用这个函数，`strcpy(a,"abc");`。同样的，也不能直接在两个字符数组之间相互赋值，也得用`strcpy(a,b);`来把b赋值给a

string类型字符串

string类型的变量可以用来存储一个字符串，并且可以把这个字符串当成一个整体来处理——可以对string进行赋值、拼接、裁切、赋值等。

可以用`s.length()`来查询字符串s的长度，可以用s[0]来获取s的第一个字符是什么，也可以s=a+b进行字符串拼接。

字符串string需要使用头文件string，且包括以下常用方法

1. `string s`：定义一个string类型的变量s
2. `s+=str`或`s.append(str)`：在字符串s后面拼接字符串str
3. `s<str`：比较s的字典序是否在str的字典序之前
4. `s.size()`或`s.length()`：得到字符串s的长度
5. `s.substr(pos,len)`：截取字符串s，从第pos个位置开始len个字符，并返回这个字符串
6. `s.insert(pos,str)`：在字符串s的第pos个字符之前插入字符串str，并返回这个字符串
7. `s.find(str,[pos])`：在字符串s中从第pos给字符开始寻找str，并返回位置，如果找不到返回1（实际上是常数string::npos，用int强制转换为-1）。pos可以省略，默认值是0。

string与字符数组的互相赋值

string到字符数组

```
int n;
char s1[10];
string s2="LuoGu";
int main() {
    int len = s2.copy(s1,9);
    s1[len]='\0';
    cout << s1<<endl<<len;
    return 0;
}
```

`string.copy(s1,n,[pos])`，s1是字符数组，n是最多允许复制的字符数，`[pos]`是可选项，是要包含的第一个字符的位置。`copy`函数返回值为复制的字符数。

如前面的`fgets`和`getline`一样，`copy`也只是半覆盖，并且在末尾不会自动加上`'\0'`字符，如果我们不手动加上的话，会导致后面的字符也被输出。

或者还有一种方式

```
char s1[10]="aaaaaaa";
string s2="LuoGu";
strcpy(s1,s2.c_str()); // strncpy(s1,s2.c_str(),10);
```

这里的`s2.c_str()`就是把string转换为字符数组，再用上面提到的`strcpy`把这个字符数组拷贝给s1

字符数组到string

```
char s1[10]="LuoGu";
string s2 = s1;
```

文件操作与重定向

```
freopen("title.in","r",stdin);
freopen("title.out","w",stdout);
```

这两行就分别把输入输出重定向到`title.in`和`title.out`文件

注意，需要加上`cstdio`头文件

注意：在OJ上提交代码的时候，要把重定向删掉或注释掉，也可以像下面这样写

```
int main() {
    #ifndef ONLINE_JUDGE
        freopen("title.in","r",stdin);
        freopen("title.out","w",stdout);
    #endif
    return 0;
}
```

在很多OJ平台，编译时会定义`ONLINE_JUDGE`宏，如果检测到这个宏，就不会进行重定向操作

函数与结构体

函数的定义方式

```
返回类型 函数名 (参数类型1 参数名1, ..., 参数类型n 参数名n){
    函数体
    return 结果;
}
```

返回类型除了常见的变量类型外，还有一个特殊的返回类型，`void`，也即无返回值，不需要return语句

函数调用

`函数名 (参数1, ..., 参数n)`

函数定义里的参数是形式参数，函数调用时的传递的值是实际参数

例1

```
int fun(int a[], int b){
    pass;
    return b;
}
```

变量作用域与参数传递

如例1，参数`a`和参数`b`不同，前者是一个数组，后者是单个变量

单个变量是可以传递实际参数的，也就是在函数里修改形式参数的值，不会改变函数外部的参数的值。

但数组传递的是内存地址，也就是函数里的形式参数实际上只是函数外的参数的别名，修改它就相当于在外面修改。

全局变量与局部变量

在函数外定义的变量是全局变量，在函数内部也能访问。如果函数内也定义了一个和全局变量相同的名字的变量，那么全局变量在这个函数内会被屏蔽掉。

请看以下代码

```
#include<iostream>
using namespace std;
void swap(int a, int b){
    int t = b;
    b = a;
    a = t;
}
int main(){
    int a, b;
    cin >> a >> b;
    swap(a, b);
    cout << a << endl << b;
}
```

假设输入`1`、`2`，我们期望得到的输出是`2`、`1`，但实际上还是`1`、`2`，这是为什么呢。

诚如上面所说，`a`和`b`是单个变量，传递的是变量的值，也就相当于是把函数外的`a`的值拷贝了一份赋值给函数内的`a`，在内存里两个`a`虽然名字相同，但实际上是两个不同的变量，我们在函数内修改的是函数内的`a`，是不会影响到函数外的变量的。

那么怎样可以做到在函数内修改函数外的变量的值呢。我们在函数的参数列表里要这样写

```
#include<iostream>
using namespace std;
void swap(int &a, int &b){
    int t = b;
    b = a;
    a = t;
}
int main(){
    int a, b;
    cin >> a >> b;
    swap(a, b);
    cout << a << endl << b;
}
```

之前讲过`&`是取内存地址的意思，那么这里就是把`a`，`b`的内存地址传递给形式参数了。

注意，虽然string类型相当于加强版的字符数组，但它不会和字符数组一样传递内存地址，它做了封装，会自动拷贝一份传递过去，所以它和单个变量一样。

结构体

把一些不同类型的数据整合起来形成一个整体。

比如我们定义一个学生类型的结构体。

```
struct student {
    string name;
    int Chinese, Math, English;
} a, ans;
```

抽象来说，结构体的定义如下

```
struct 类型名 {
    数据类型1 成员变量1;
    数据类型2 成员变量2;
} {结构体变量名};

struct 已经定义过的类型名 结构体变量名 ;
//"struct"也可以不写
```

变量`a`作为一个学生，可以用`a.name`来表示`a同学`的姓名。这些成员变量可以像普通的变量那样读入、赋值或参与表达式运算。而相同类型的结构体变量之间也可以直接进行赋值运算。

结构体也可以作为数组的元素，返回类型和函数参数

成员函数与构造函数

在结构体内访问自己对象结构体的成员变量或成员函数时，在对应的变量名或函数明前加`this->`，但是如果不产生起义，不加也可以。在结构体外调用结构体成员函数时只需要`结构体变量名.成员变量名`即可

不仅如此，结构体对象还能像单个变量一样，作为参数直接传参金函数，而不是和数组一样只能传入一个地址。这样的话在函数中修改直接传入的结构体对象中的成员变量也不会影响到函数外面，除非加上`&`符号变成引用传参。

见如下代码：

```
struct student {
    int id;
    int academic, quality;
    double overall;
    student(int _id, int _ac, int _qu){ // 初始化构造函数，构造函数不需要任何返回类型，也不需要void
        this->id = _id;
        this->academic = _ac;
        this->quality = _qu;
        this-> overall = 0.7 * _ac + 0.3 * _qu;
        // 这里的this->也可以省略
    }
    student(){} // 没有传递参数的初始化构造函数，没有这个的话直接定义结构体对象时（student a;）会编译失败
    int sum() { // 就和普通的函数一样定义
        return academic + quality;
    }
};
int is_excellent(student s){
    return s.overall >= 80 && s.sum() > 140;
}
int main(){
    cin n;
    for (int i = 1; i <= n; i++){
        int tmp_id, tmp_ac, tmp_qu;
        cin >> tmp_id >> tmp_ac >> tmp_qu;
        student one_student(tmp_id, tmp_ac_ tmp_qu);
        if(is_excellent(one_student)) cout << "Excellent" << endl;
        else cout << "Not Excellent" << endl;
    }
}
```

结构体还有更多复杂的特性，如公开或私有访问权限，但算法竞赛中用的不多，这里就不做介绍了。

尾声

至此，已经介绍完了入门部分的全部内容。虽然只是介绍了C++的基本内容，但是已经足够应对大多数的算法竞赛所要求的语言知识了。C++是一种非常复杂的编程语言，可能在接下来的学习过程中会继续补充新的知识点。

编程绝不是靠读书或听课就能搞懂的，必须要亲自上机实践。如有条件，尽可能多地区完成洛谷中“入门难度”的题目。

本文档大部分内容整理自洛谷学术组编写的《深入浅出程序设计竞赛（基础篇）》，在此特别感谢。