



# C++编程



# 目录

- 函数
- 递归
- 不定长输入
- 无穷大
- 排序和去重
- 埃氏筛
- 线性筛
- 二分查找
- 在线和离线
- 二维数组
- 字符数组
- 字符串
- 高精度运算
- 结构体
- 文件操作
- 快速读入
- 位运算

# 文件操作

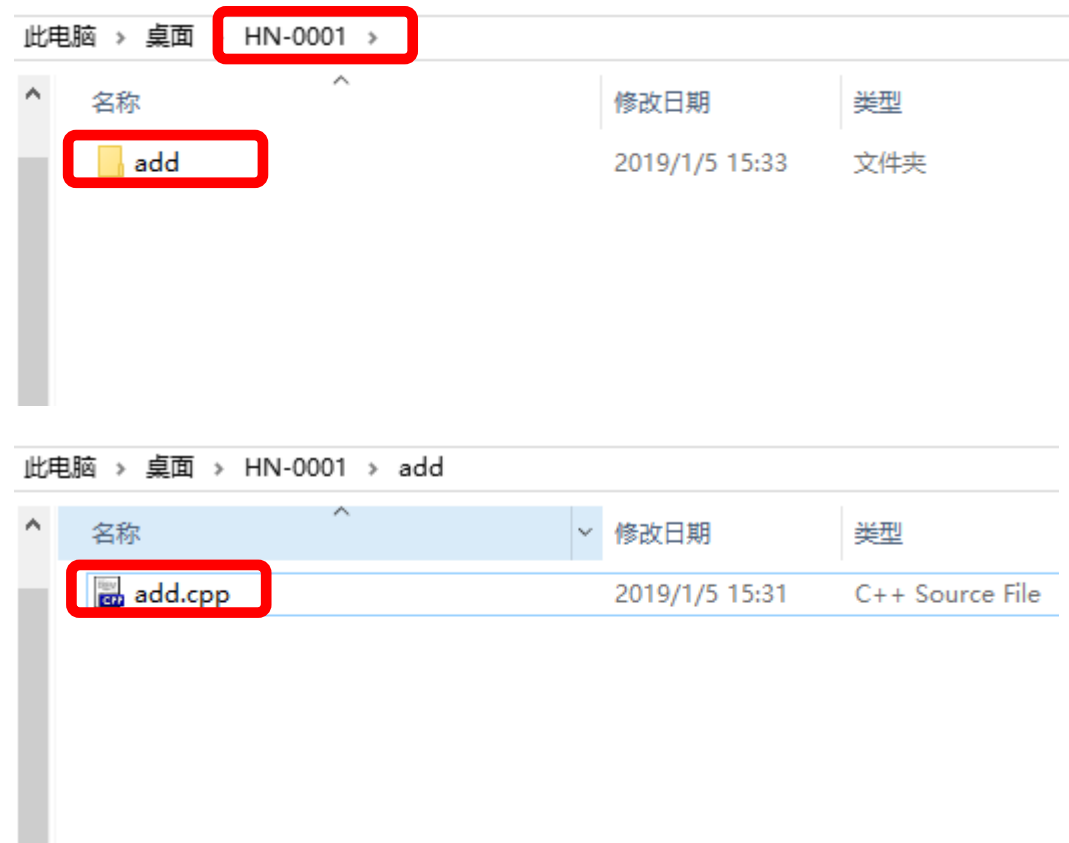
- 我们通过重定向操作实现文件读写

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    freopen("add.in", "r", stdin); //标准写入
    freopen("add.out", "w", stdout); //标准输出
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d", a + b);
    fclose(stdin); //关闭stdin
    fclose(stdout); //关闭stdout
    return 0;
}
```

# 文件操作

1. 考生个人目录，名字为考生考号
2. 为每道题建立对应的英文子目录
3. 把对应英文名的源程序cpp放入其中
4. 写好对应文件重定向语句

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    freopen("add.in", "r", stdin);
    freopen("add.out", "w", stdout);
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d", a + b);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```



# 真实案例

## 湖南广益实验中学信息学奥赛培训班期末测试

(七年级上学期)

题目名称	奇数偶数	珠心算	国王和骑士	游戏
题目类型	传统型	传统型	传统型	传统型
目录	odd	count	knight	game
可执行文件名	odd	count	knight	game
输入文件名	odd.in	count.in	knight.in	game.in
输出文件名	odd.out	count.out	knight.out	game.out
每个测试点时限	1.0 秒	1.0 秒	1.0 秒	1.0 秒
内存限制	128MB	128MB	128MB	128MB
测试点数目	10	10	10	10
每个测试点分值	10	10	10	10

名称	修改日期	类型
count	2019/1/5 15:44	文件夹
game	2019/1/5 15:44	文件夹
knight	2019/1/5 15:44	文件夹
odd	2019/1/5 15:44	文件夹

# getchar函数/putchar函数

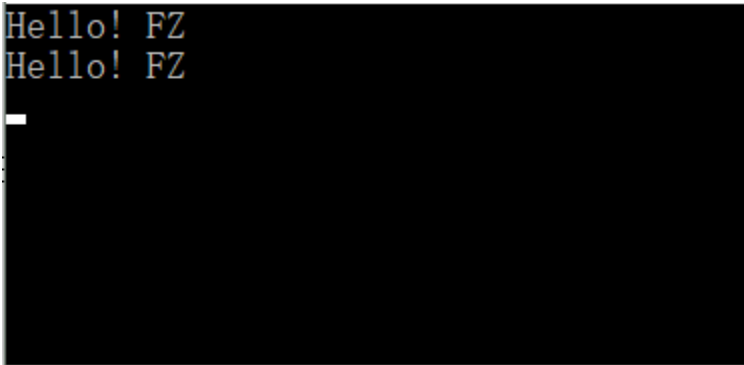
- 这是一对函数：**getchar()** / **putchar()**
- 用于读入/输出单个字符

```
Hello
H
-----
Process exited after 10.46 seconds with return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char c;
    c = getchar();
    putchar(c);
    return 0;
}
```

# getchar函数/putchar函数

- 这是一对函数: **getchar()** / **putchar()**
- 或者这样用:



```
Hello! FZ
Hello! FZ
_
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char c;
    while (c = getchar())
        putchar(c);
    return 0;
}
```

# 快速读入

- 本来到此为止了： `getchar/putchar`这都是很正经的函数
- 但是： `getchar()`函数的读入速度，比`scanf`还要快！
- 于是常常被TLE折磨的Oier，就想到要利用`getchar`的快速



# 快速读入

本页PPT为选学，  
不要求掌握



- 于是有了这段快速读入代码

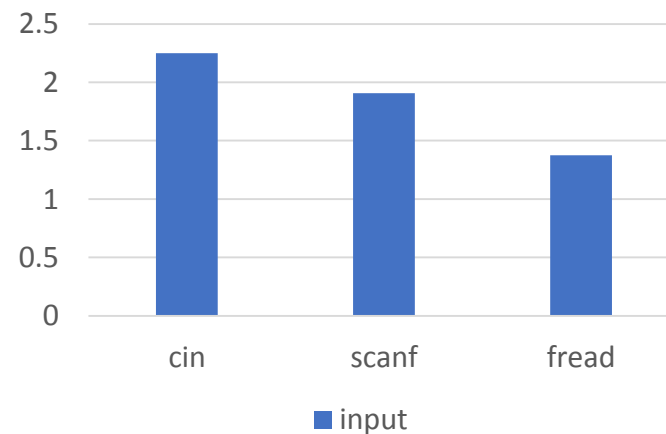
```
int read() //也可以根据题意定义 Long Long型
{
    char ch = getchar(); int num = 0, f = 1;
    while (ch < '0' || ch > '9') {if (ch == '-') f = -1; ch = getchar();} //处理负数
    while (ch >= '0' && ch <= '9') {num = num * 10 + (ch - '0'); ch = getchar();}
    return num * f;
}

n = read(); // main函数中调用 read()
```

# 快速读入

- 我们可以看到： $10^6$ 规模的数据输入，快速读入比scanf要快约40%，比cin要快约65%

试题		选手			
名称	排名	input	总分	总用时(s)	测试时间
fread	1	100	100	1.375	2019/1/8 17:33:47
scanf	1	100	100	1.906	2019/1/8 17:23:56
cin	1	100	100	2.25	2019/1/8 17:23:47



# 快速读入

- 但这并不是终极的**fread**版本，因为大多数情况下，它比**scanf**快得有限
- 终极**fread**适合卡常到极致的选手，有需要者可自行百度
- 搭配快速输出**fwrite**食用更美妙

# 二进制

- 我们都知道，计算机采用二进制存储数据
- 进一步的，二进制数在计算机中是以补码形式存储的

# 补码

- 计算机中的码制有原码、反码、补码三种
- 比如10，二进制表示为：00001010（一个字节8位）
- 注意前面那个最高位的0，是符号位，0表示正数
- 那么-10的二进制表示呢？不是写成-00001010
- 而是写成：10001010，也就是最高位的符号位1表示负数

# 补码

- 所以，一个字节能表示多大的数？  
 $-127 \sim 127$  ( $-2^7 \sim 2^7$ )
- 而int是四个字节存储，所以能表示  
 $-2^{31}-1 \sim 2^{31}-1$

# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 反码即是如果符号位为1，则除最高位符号位以外，按位取反
- 10的反码表示是00001010
- -10的反码表示是11110101

# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 这里的00001010 (10) 以及11110101 (-10) 都是反码
- 补码即是如果符号位为1，则除最高位符号位以外，反码+1
- 10的补码表示是00001010
- -10的补码表示是11110110



# 补码

- 计算机中的码制有原码、反码、补码三种
- 这里的00001010 (10) 以及10001010 (-10) 都是原码
- 这里的00001010 (10) 以及11110101 (-10) 都是反码
- 这里的00001010 (10) 以及11110110 (-10) 都是补码
- 计算机中的二进制存储都是采用的补码

# 补码

- 为什么都要采用补码呢？

- 我们试试看 $10-10$ ： $10+(-10)$

$00001010 + 11110110 = 00000000$ （注意符号位也是参与运算的！）

- 所以现在可以回答为什么计算机中的二进制存储要用补码：
- 为了把所有的加减法统一成加法

# 无符号数

- 这里就顺便介绍一个新概念：无符号数：**unsigned**
- 现在我们就可以理解了：无符号数就是字节的最高位也表示具体的数值而不再表示数的正负
- 它的好处是可以容纳的范围比有符号数大一倍，坏处是不能再表示负数了

# 无符号数

- 2147483648原本定义为int是要爆的
- 如果某些题明确是非负的，就可以定义为无符号数

```
2147483648
-----
Process exited after 0.1927 seconds w
ith return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    unsigned int n = 2147483648;
    printf("%u", n);
    return 0;
}
```

# 位运算：移位

- 这两个程序得到的结果都是10，但是位运算更快
- 原因是位运算是直接对二进制操作，省却了中间转换十进制的过程
- 二进制移位运算：<<、>>

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 20;
    printf("%d", n / 2);
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 20;
    printf("%d", n >> 1);
    return 0;
}
```

# 高低位交换



- 给一个正整数 $n$ 。这个数可以用一个32位的二进制数表示（不足32位补0）。我们称这个二进制数的前16位为“高位”，后16位为“低位”。将它的高低位交换，我们可以得到一个新的数。问这个新的数是多少（用十进制表示）

$n \in \text{int}$

Sample input	Sample output
1314520	249036820

# 分析



- 这个题暴力也能做，因为 $2^{16}=65536$ ，所以可以通过对65536取整、取模分离出前后16位
- 不过用位运算更快捷

# 参考代码

注意这里如果不用无符号数，可能会爆的，为什么？



```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    unsigned int n;
    scanf("%u", &n);
    printf("%u", (n >> 16) + (n << 16));
    return 0;
}
```



# 位运算：按位与/按位或

- 这两个程序得到的结果都是“odd”，同样位运算更快
- 二进制按位与运算：&，按位或运算：|

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 101;
    if (n % 2 == 1) printf("odd");
        else printf("even");
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 101;
    if (n & 1) printf("odd");
        else printf("even");
    return 0;
}
```

# 有多少个1

- 给一个正整数 $n$ ，问它的二进制表示中有多少个1

Sample input	Sample output
15	4

# 位运算

- $x \& x - 1$ 可以快速去掉 $x$ 的二进制位中最后一个1

- 顺便提一下：位运算的优先级相当低：比加减还低

```
4
-----
Process exited after 0.1192 seconds w
ith return value 0
请按任意键继续. . .
```



```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n = 15, cnt = 0;
    while (n > 0)
        cnt ++, n &= n - 1;
    printf("%d\n", cnt);
    return 0;
}
```

# 课外加练

- luogu 1100      高低位交换