

# 序列第k小

给定一个长度为 $n$ 的序列, 问第 $k$ 小的元素是多少.

- **Standard Input**

第1行, 两个整数 $n, k$ ; 接下来一行 $n$ 个数, 表示这个序列.

- **Standard Output**

输出仅1行, 表示第 $k$ 小的元素.

- **Samples**

| Input               | Output |
|---------------------|--------|
| 5 3<br>18 23 4 5 12 | 12     |

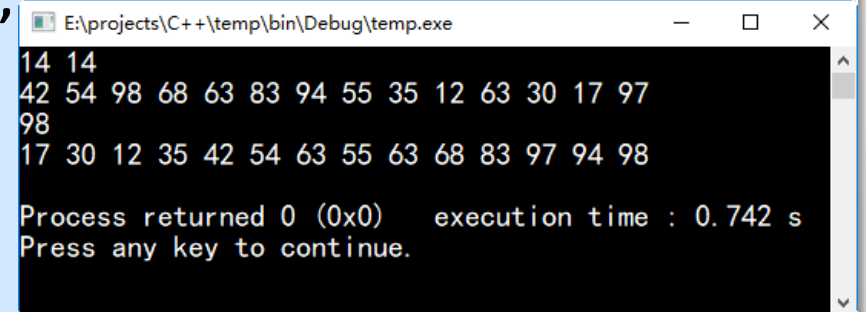
## 题目分析

- 利用快速排序过程, 经过一次划分后区间 $[L,R]$ 分成三部分: (1)左边 $[\text{left}, i-1]$ 小于等于基准数; (2)中间 $a[i]$ 等于基准数; (3)右边 $[i+1, \text{right}]$ 大于等于基准数.
- 如果:
  - ➔  $k < i$ , 答案在左边区间, 只需调用 $\text{Kth}(\text{left}, i-1, k)$ 即可;
  - ➔  $k == i$ , 答案 $a[i]$ ;
  - ➔  $k > i$ , 答案落在右边区间, 只需调用 $\text{Kth}(i+1, \text{right}, k)$ 即可.

# 参考程序

```
#include <bits/stdc++.h>
using namespace std;
int a[100005], k, n;
int Kth(int left, int right, int k)
{
    if(left == right) return a[left];
    if(left < right)
    {
        int i = left, j = right, x = a[left];
        while(i < j)
        {
            while(i < j && a[j] >= x) j--;
            if(i < j) a[i++] = a[j];
            while(i < j && a[i] <= x) i++;
            if(i < j) a[j--] = a[i];
        }
        a[i] = x;
    }
}
```

```
        if(k < i) return Kth(left, i-1, k);
        else if(k == i) return x;
        else return Kth(i+1, right, k);
    }
}
int main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    int x = Kth(1, n, k);
    cout << x << endl;
    return 0;
}
```



```
E:\projects\C++\temp\bin\Debug\temp.exe
14 14
42 54 98 68 63 83 94 55 35 12 63 30 17 97
98
17 30 12 35 42 54 63 55 63 68 83 97 94 98

Process returned 0 (0x0)   execution time : 0.742 s
Press any key to continue.
```

# 表达式求值问题

假设一个表达式仅由‘+’、‘-’、‘\*’、‘/’、‘(’、‘)’、‘#’和正整数构成, 从标准输入设备读入一个这样的表达式(格式合法, 合乎所述要求), 求出它的值并输出. 其中字符‘#’仅作表达式结束符用.

- **Standard Input**

只有一行, 即符合题目叙述的算术表达式。

- **Standard Output**

输出1行, 即输出表达式的值。

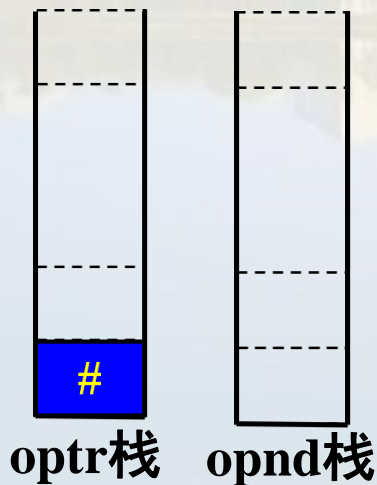
- **Samples**

| Input                        | Output   |
|------------------------------|----------|
| $(100+58)*(3+15000+1500)*10$ | 26074740 |

# 题目分析

$$(100+58)*(3+15000+1500)*10$$

- 该问题可以用栈结构完成. 为了实现算符优先算法, 使用两个工作栈, 一个操作符栈(optr), 一个操作数栈(opnd), 以寄存操作数或运算结果.
- 首先置操作数栈opnd为空栈, 表达式结束符' #'为操作符栈optr的栈底元素.
- 依次读入表达式中每个字符, 若是操作数, 则入操作数opnd栈; 若是运算符, 则和操作符栈optr中的栈顶运算符比较优先级(具体优先级定义如下表定义) 然后做相应操作, 直至整个表达式求值完毕.



| 运算符       | + | - | * | / | ( | ) | # |
|-----------|---|---|---|---|---|---|---|
| 对应的数字     | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 栈内操作符的优先级 | 3 | 3 | 5 | 5 | 1 | 6 | 0 |
| 栈外操作符的优先级 | 2 | 2 | 4 | 4 | 6 | 1 | 0 |

# 参考程序

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define SSIZE 500
typedef struct _stack {
    int element[SSIZE];
    int top;
}stack;
char ch[] = "+-*/()#"; //下标即"对应的数字"
int f1[] = {3, 3, 5, 5, 1, 6, 0}; //栈内元素优先级
int f2[] = {2, 2, 4, 4, 6, 1, 0}; //栈外元素优先级
void initstack(stack* s) { //初始化堆栈
    s->top = -1;
}
int isempty(stack* s) { //判空操作
    if(s->top < 0)    return 1;
    return 0;
}
```

| 运算符   | + | - | * | / | ( | ) | # |
|-------|---|---|---|---|---|---|---|
| 对应数字  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 栈内优先级 | 3 | 3 | 5 | 5 | 1 | 6 | 0 |
| 栈外优先级 | 2 | 2 | 4 | 4 | 6 | 1 | 0 |

```
int push(stack* s, int value) { //入栈操作
    if(s->top >= SSIZE - 1)    return 0;
    s->element[++s->top] = value;
    return 1;
}
int gettop(stack* s) { //取栈顶元素
    return s->element[s->top];
}
int pop(stack* s) { //出栈操作
    return s->element[s->top--];
}
int cton(char c) { //把运算符转换下标
    switch(c) {
        case '+': return 0;
        case '-': return 1;
        case '*': return 2;
        case '/': return 3;
        case '(': return 4;
        case ')': return 5;
        default: return 6;
    }
}
```

(100+58)\*(3+15000+1500)\*10



$$(100+58)*(3+15000+1500)*10$$

```
char compare(char c1, char c2) { // 优先级比较
    int i1 = cton(c1);
    int i2 = cton(c2);
    if(f1[i1] > f2[i2])        return '>';
    else if(f1[i1] < f2[i2])    return '<';
    else                        return '=';
}

int operate(int a, int t, int b){
    int sum;
    switch(t) {
        case 0: sum = a + b; break;
        case 1: sum = a - b; break;
        case 2: sum = a * b; break;
        default: sum = a / b;
    }
    return sum;
}
```

| 运算符   | + | - | * | / | ( | ) | # |
|-------|---|---|---|---|---|---|---|
| 对应数字  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 栈内优先级 | 3 | 3 | 5 | 5 | 1 | 6 | 0 |
| 栈外优先级 | 2 | 2 | 4 | 4 | 6 | 1 | 0 |

```
int evaluate() { // 调用前面函数, 完成运算
    char c;
    int i = 0, sum = 0;
    int x, t, a, b;
    stack optr, opnd;
    initstack(&optr);
    push(&optr, cton('#'));
    initstack(&opnd);
    c = getchar();
    while((c != '#') || (ch[gettop(&optr)] != '#')) {
        if(isdigit(c)) {
            sum = 0;
            while(isdigit(c)) {
                sum = sum * 10 + (c - '0');
                c = getchar();
            }
            push(&opnd, sum);
        } // end if
    }
}
```

$$(100+58)*(3+15000+1500)*10$$

```

else {
    switch(compare(ch[gettop(&optr)], c)) {
        case '<': push(&optr, cton(c));
                c = getchar();
                break;
        case '=': x = pop(&optr);
                c = getchar(); //只可能是括号
                break;
        case '>': t = pop(&optr); //开始运算
                b = pop(&opnd);
                a = pop(&opnd);
                push(&opnd, operate(a, t, b));
                break;
    } //end switch
} //end else
} //end while
return (gettop(&opnd));
}

```

```

int main() {
    int result = evaluate();
    printf("%d\n", result);
    return 0;
}

```

| 运算符   | + | - | * | / | ( | ) | # |
|-------|---|---|---|---|---|---|---|
| 对应数字  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 栈内优先级 | 3 | 3 | 5 | 5 | 1 | 6 | 0 |
| 栈外优先级 | 2 | 2 | 4 | 4 | 6 | 1 | 0 |



# 问题：逆波兰表达式

逆波兰表达式是一种把运算符前置的算术表达式，例如普通的表达式 $2+3$ 的逆波兰表示法为 $+ 2 3$ 。逆波兰表达式的优点是运算符之间不必有优先级关系，也不必用括号改变运算次序，例如 $(2+3)*4$ 的逆波兰表达法为 $* + 2 3 4$ 。本题求解逆波兰表达式的值，其中运算符包括 $+$ 、 $-$ 、 $*$ 、 $/$ 四个。

输入格式：1行，其中运算符和运算数之间都用空格分隔，运算数是浮点数。

输出格式：1行，表达式的值。可直接用`printf("%f\n",v)`输出表达式的值 $v$ 。

提示：可使用`atof(str)`把字符串转换为一个`double`类型的浮点数。`atof`定义在`math.h`中。

| 输入样例                        | 输出样例          |
|-----------------------------|---------------|
| $* + 11.0 12.0 + 24.0 35.0$ | $1357.000000$ |
| $+ 1 + 2 + 3 + 4 5$         | $15.000000$   |
| $+ + + 1 2 + 3 4 + 5 6$     | $21.000000$   |

样例1对应的表达式： $(11.0 + 12.0) * (24.0 + 35.0)$