4. Функции и структура программы

Владимир Верстов

Б. Керниган, Д. Ритчи. Язык программирования C

Поиск шаблона в строке (аналог `grep`)

```
while (на выход поступает строка)
    if (строка содержит шаблон)
        Вывести строку
```

```c
#include <stdio.h>

#define MAXLINE 1000

int getline(char line[], int max);
int strindex(char source[], char searchfor[]);

char pattern[] = "text";

int main()
{
    char line[MAXLINE];
    int found = 0;
    while (getline(line, MAXLINE) > 0)
        if (strindex(line, pattern) >= 0) {
            printf ("%s", line);
            found++;
        }
    return found;
}
```

```c
int getline(char s[], int lim)
{
   int c, i;
   i = 0;
   while (--lim > 0 && (c = getchar()) != EOF && c != '\n' )
      s[i++] = c;
   if (c == '\n')
      s[i++] = c;
   s[i] = '\0';
   return i;
}
```

```c
/* вычисляет место t в s или выдает -1, если t нет в s */
int strindex (char s[], char t[])
{
   int i, j, k;
   for (i = 0; s[i] != '\0'; i++) {
      for (j = i, k = 0; t[k] != '\0' && s[j] == t[k]; j++, k++) {
         ;
      }
      if (k > 0 && t[k] == '\0')
         return i;
   }
   return -1;
}
```

Определение функции

```
тип-возвращаемого-значения имя-функции (объявления аргументов)
{
    объявления и операторы
}
```

Минимальная функция заглушка

```
void dummy()  {}
```

Оператор `return`

- — … это механизм возвращения значений из вызываемой функции в вызывающую.

- После `return` может идти любое выражение

```
return выражение;
```

Компиляция программы из нескольких файлов

- Каждая функция может быть выделена в отдельный файл с исходным кодом

- Команда для компиляции в таком случае:

```
cc main.c getline.c strindex.c
```

Внешние переменные

- Внутренние переменные - аргументы функций и переменные, определенные внутри функций
- Внешние переменные определяются вне каких-либо функций и могут использоваться несколькими функциями
- Функции — всегда внешние, в языке C нельзя определить функцию внутри функции
- Внешние переменные — способ обмена данными между функциями

Программа калькулятор

Формы записи формул

Нотация	Пример	Особенности
Прекфиксная или польская	* + 3 4 8	• Не нужны скобки • Удобно писать программы
Инфиксная	(3 + 4) * 8	• Нужны скобки • Важен приоритет операций • Легко воспринимается человеком
Постфиксная или обратная польская	3 4 + 8 *	• Не нужны скобки • Удобно писать программы

Алгоритм калькулятора

```
while (операция или операнд - не конец файла)
   if (число)
      поместить его в стек
else if (операция)
   извлечь операнды из стека
   выполнить операцию
   поместить результат в стек
else if (конец строки)
   извлечь и вывести вершину стека
else
   ошибка
```

Стек

- — это структура данных или коллекция по принципу LIFO, Last In - First Out

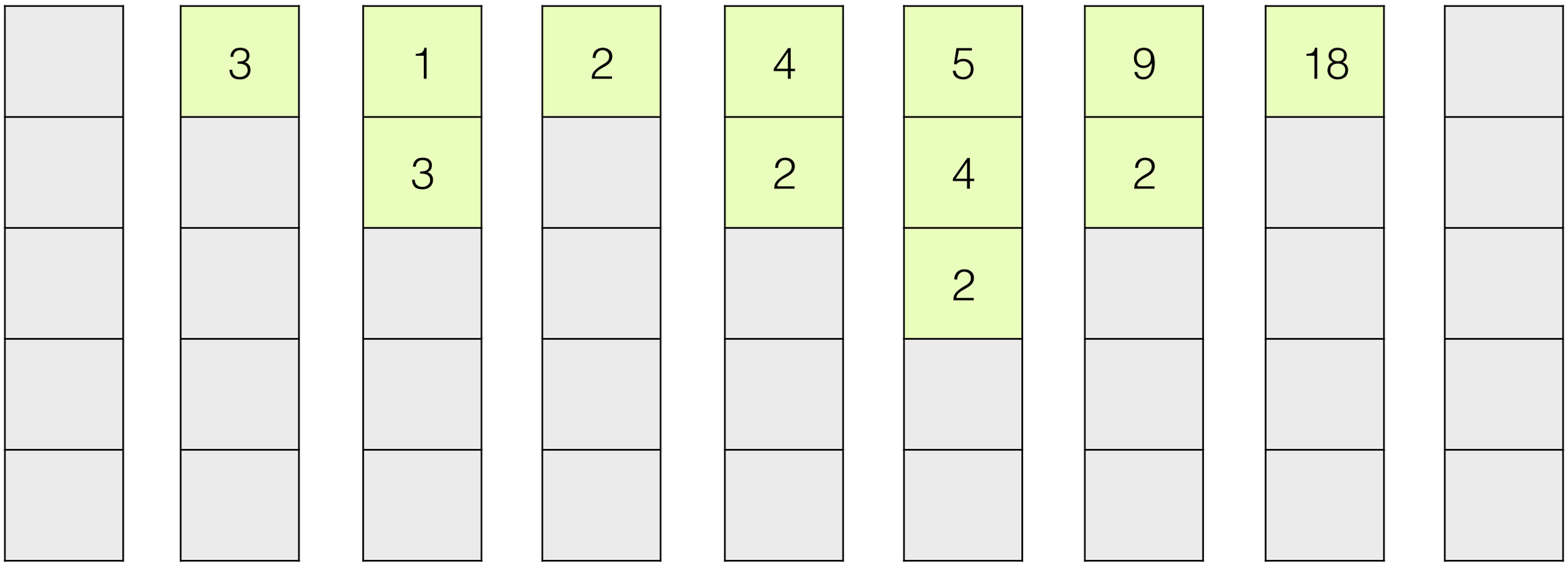
- Доступные операции:

  ➡ push — поместить данные в стек

  ➡ pop — извлечь данные из вершины стека

Пример

```
(3 - 1) * (4 + 5) -> 3 1 - 4 5 + *
```



Структура программы

```
#include-директивы
#define-директивы

объявления функций для main
int main () {...}

внешние переменные для функций push и pop

void push (double f) {...}
double pop () {...}

int getop(char s[]) {…}

функции, вызываемые функцией getop
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "calc.h"

#define MAXOP 100
#define NUMBER '0'

int getop (char []);
```

```c
int main() {
    int type;
    double op2;
    char s[MAXOP];
    while ((type = getop(s)) != EOF) {
        switch (type) {
            case NUMBER:
              push (atof (s));
                break;
            case '+':
                push (pop() + pop());
                break;
            case '*':
                push (pop() * pop());
                break;
            case '-':
                op2 = pop();
                push (pop() - op2);
                break;
            case '/' :
                op2 = pop();
                if (op2 != 0.0)
                    push (pop() / op2);
                else
                    printf("error: zero divisor\n”);
                    return -1;
                break;
            case '\n' :
                printf("\t%.8g\n", pop());
                break;
            default:
              printf("error: unknown command %s\n", s);
              return -1;
        }
    }
    return 0;
}
```

```c
int getop(char s[]) {
    int i, c;
    while ((s[0] = c = getch()) == ' ' || c == '\t' )
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.')
        return c;
    i = 0;
    if (isdigit(c))
        while (isdigit(s[++i] = c = getch()))
            ;
    if (c == '.')
        while (isdigit(s[++i] = c = getch()))
            ;
    s[i] = '\0';
    if (c != EOF)
        ungetch(c);
    return NUMBER;
}
```

calc.h

```
void push (double);
double pop ();

int getch();
void ungetch(int);
```

```c
// buffer.c

#include <stdio.h>
#include <stdlib.h>

#define BUFSIZE 100

static char buf[BUFSIZE];
static int bufp = 0;

int getch(void) {
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) {
    if (bufp >= BUFSIZE) {
        printf ("ungetch: too many characters\n");
     exit(333);
    } else {
        buf[bufp++] = c;
    }
}
```

```c
// stack.c

#include <stdio.h>
#include <stdlib.h>

#define STACK_SIZE 100

static double stack[STACK_SIZE];
static int top = -1;

void push(double n) {
   if (top + 1 < STACK_SIZE) {
      stack[++top] = n;
   } else {
      printf("error: stack is full\n");
      exit(111);
   }
}

double pop() {
   if (top != -1) {
      return stack[top--];
   } else {
      printf("error: stack is empty\n");
      exit(222);
   }
}
```

Компиляция

- Заголовочные файлы (header files, .h) нет необходимости компилировать

- Команда для компиляции программы калькулятора:

```
cc main.c stack.c buffer.c -o calc
```

Область видимости

- **Областью видимости** имени (переменной или функции) является часть программы, в пределах которой можно использовать имя (переменной или функции)

- Область видимости **локальной переменной** распространяется от точки, в которой она объявлена, до конца блока

- Область видимости **внешней переменной** распространяется от точки, в которой она объявлена, до конца файла

extern

- Ключевое слово `extern` нужно использовать при необходимости использовать внешнюю переменную до ее определения или если она определена в другом файле

- Объявление — сообщение о том, какими свойствами обладает переменная (чаще всего какого она типа)

- Определение — выделение памяти под хранение переменной

Внешние статические переменные

- Область видимости статичных внешних переменных или функций ограничена файлом, в котором они объявлены: от точки объявления до конца файла

```
static double stack[STACK_SIZE];
static int top = -1;
```

Локальные статические переменные

- Локальные статические переменные существуют непрерывно, а не создаются и уничтожаются во время вызова функции

- Локальные статические переменные применяются для хранения состояния внутри функции

- Пример, подсчет количества вызовов функции f

```
void f() {
   static int i = 0;
   i++;
   printf("i = %d\n", i);
}
```

Регистровые переменные

- Ключевое слово `register` сообщает компилятору, что переменная будет интенсивно использоваться и ее имеет смысл разместить в регистре процессора

- Ключевое слово `register` может быть проигнорировано компилятором

- Пример:

```
register int i;
```

Инициализация переменных

```
int x = 1;
char squote = '\'';
long day = 1000L * 60L * 60L * 24L;
```

Инициализация локальных переменных

```
int binsearch(int x, int v[], int n) {
    int low = 0,
    int high = n - 1
    int mid;
    ...
```

Инициализация массивов

```
// arrays
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// character arrays or strings
char pattern[] = "text";
char pattern[] = {'t', 'e', 'x', 't', '\0'};
```

Рекурсия

- Функции в C могут вызываться рекурсивно, т.е. функция может вызывать сама себя прямо или косвенно

“Чтобы понять рекурсию, нужно понять рекурсию.”

Числа Фибонначи

```
unsigned int fib(unsigned int n){
  if (n < 2)
    return n;
  else
    return fib(n - 1) + fib(n - 2);
}
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,…

Препроцессор C

- Первый этап компиляции
- Типовое использование: директивы `#include` и `#define`
- А так же: макроподстановки и условная компиляция

Включение файлов

- При обработке директива заменяется содержимым файла

- Если имя файла в “”, то его поиск обычно начинается в директории файла с исходным кодом

- Если имя файла в <>, то его поиск обычно начинается в стандартной директории с библиотеками C

```
#include "имя-файла"
#include <имя-файла>
```

Макроподстановки

- Везде, где встречается `имя`, оно заменяется на `замещающий-текст`
- Имя задается так же, как имена переменных
- Замещающий текст может быть любым
- Область видимости имени от директивы до конца файла

```
#define имя замещающий-текст
```

Пример макроподстановки

```
#define max(A, B) ((A) > (B) ? (A) : (B))

x = max(p+q, r+s);

x = ((p+q) > (r+s) ? (p+q) : (r+s));
```

Условное включение

- Включение файла `hdr.h` в текст программы не более одного раза

```
#if !defined(HDR)
#define HDR

/* здесь содержимое hdr.h */

#endif
```

```
#ifndef HDR
#define HDR

/* здесь содержимое hdr.h */

#endif
```

Условное включение

```
#if SYSTEM == SYSV
    #define HDR "sysv.h"
#elif SYSTEM == BSD
    #define HDR "bsd.h"
#elif SYSTEM == MSDOS
    #define HDR "msdos.h"
#else
    #define HDR "default.h"
#endif
#include HDR
```

Сортировка массивов

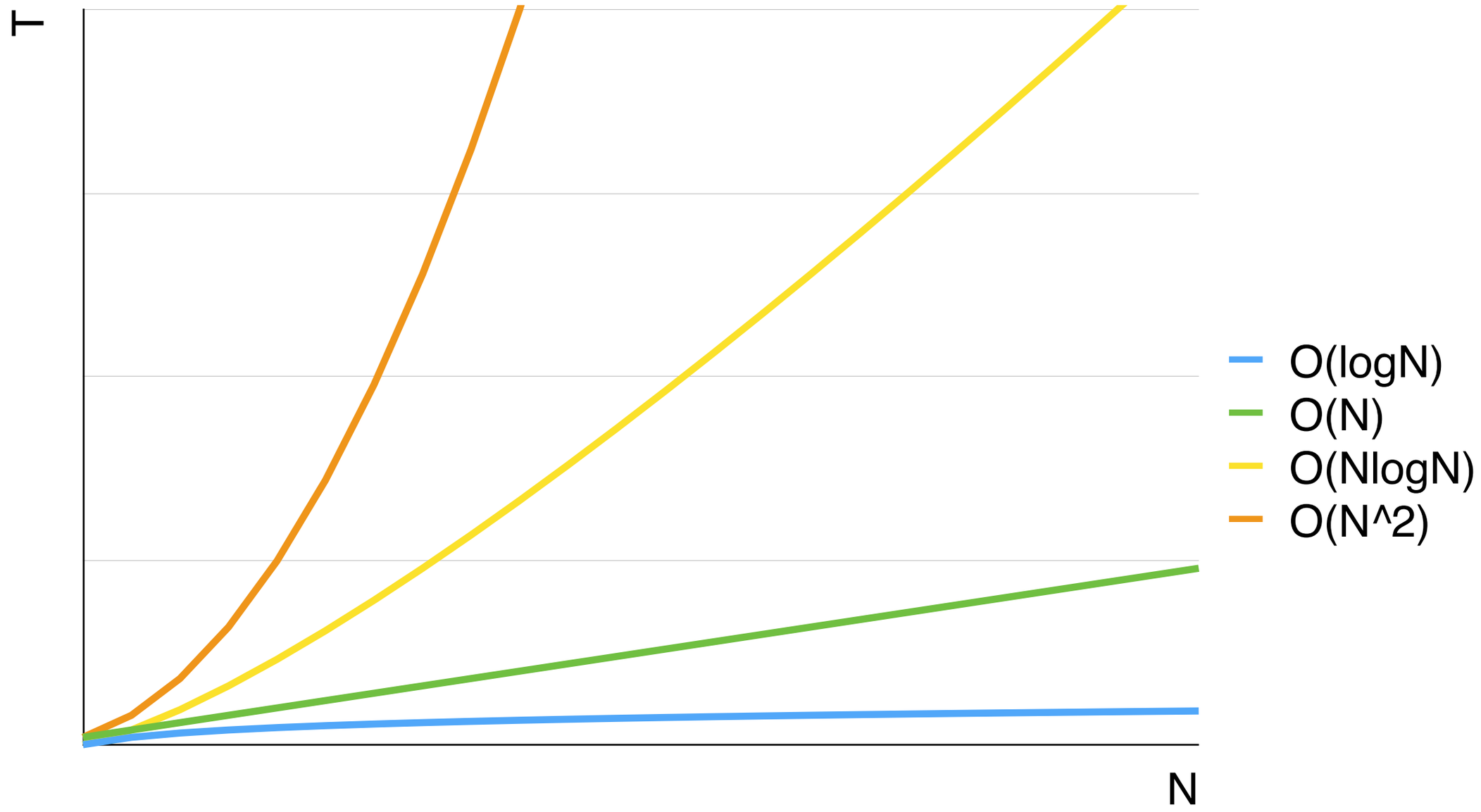
Перестановка элементов массива

```
void swap(int v[], int i, int j) {
    int temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

Сортировка пузырьком

```
void bubble_sort(int v[], int n) {
    int i, j;
    for (i = 0 ; i < n - 1; i++) {
        for (j = i+1 ; j < n; j++) {
            if (v[i] > v[j]) {
                swap(v, i, j);
            }
        }
    }
}
```

Сложность алгоритма



Сортировка Шелла

```
void shell_sort(int v[], int n) {
    int gap, i, j, temp;
    for (gap = n/2; gap > 0; gap /= 2) {
        for (i = gap; i < n; i++) {
            for (j = i-gap; j >= 0 && v[j] > v[j+gap]; j-=gap) {
                swap(v, j, j+gap);
            }
        }
    }
}
```

Быстрая сортировка

```
void quick_sort(int v[], int left, int right) {
    int i, mid;
    if (left >= right)
        return;
    swap(v, left, (left + right)/2);
    mid = left;
    for (i = left+1; i <= right; i++)
        if (v[i] < v[left])
            swap(v, ++mid, i);
    swap(v, left, mid);
    quick_sort(v, left, mid-1);
    quick_sort(v, mid+1, right);
}
```