

Обзор языка программирования C

Часть примеров позаимствована из
книги «Язык программирования C» (Б.
Керниган, Д. Ритчи)

Краткая история и особенности

- Разработан Dennis Ritchie в Bell Labs, 1972
- Первое применение – портирование ОС Unix на PDP-11
- Переносимость
- Минимализм
- Прямой доступ к памяти
- Эффективность сгенерированного кода
- Основная нынешняя область применения – системное программирование

Hello, world!

```
#include <stdio.h> /* Заголовочный файл */  
  
main() /* main() - место начала исполнения программы */  
{  
    printf("hello, world\n"); /* отформатировать и напечатать  
строку */  
}
```

Компиляция программы, состоящей из одного файла

- `gcc hello.c`
- GCC = GNU Compiler Collection
- `hello.c` – имя программы
- Выходной файл – `a.out` в рабочей директории
- Опция `-o` позволяет задать имя (`gcc hello.c -o hello`)
- Больше информации – `man gcc`

Переменные.

- Имена состоят из букв, цифр и знака подчёркивания(_), первый символ не может быть цифрой
- Переменные регистрозависимые, т.е. А и а – разные переменные
- Обычно нижний регистр используется для переменных, верхний – для констант (о них позже)
- размер переменной можно определить с помощью `sizeof(variable)`

Переменные – 2. Основные ТИПЫ.

- **char** – 1 байт. Часто используется при работе с текстовыми строками и памятью.
- **int** – целое число. В данный момент обычно 32 или 64 бита. Конкретный размер зависит от реализации.
- **float** – число с плавающей точкой с единичной точностью.
- **double** – число с плавающей точкой с двойной точностью.
- **void**

Переменные – 3. Квалификаторы.

- **signed/unsigned** – квалификатор char и int.
Показывает, является ли переменная знаковой ($-N..N-1$) или беззнаковой ($0..2*N-1$)
 - По умолчанию int является signed, для char наличие знака **зависит от реализации**
- **short/long/long long** – модификатор размера переменной.
 - **`sizeof(short int) <= sizeof(int) <= sizeof(long int) <= sizeof(long long int)`**

Переменные – 4. Примеры.

```
unsigned char c;  
int i; /* синоним signed int i */  
double d;  
short int s;  
unsigned long l; /* при наличии long/short тип int может быть  
пропущен */  
int one = 1;
```


Функция printf ()

- `#include <stdio.h>`
- `int printf(const char *format, СПИСОК аргументов для подстановки);`
- строка с форматом
 - `%s` – строка
 - `%` – целое число
 - `%f` – float
 - `%g` – double

Примеры

```
#define STR "абырвалг"
```

```
printf ("i = %d\n", 10);  
/* i = 10 */
```

```
printf ("My name is %s\n", STR);  
/* My name is абырвалг */
```

```
printf ("Длина строки \"%s\" равна %d.\n", STR,  
strlen(STR));  
/* Длина строки "абырвалг" равна 8. */
```

```
printf ("2/3=%10.5f", (float)2/3); /* 10 задаёт минимальную  
длину поля, 5 - число цифр после запятой */  
/* 2/3= 0.66667 */
```

Введение в массивы

- Набор однотипных элементов, последовательно располагающихся в памяти
- Нумерация идёт с нуля
- Возможно задание многомерных массивов
- Количество элементов массива – $\text{sizeof(arr)}/\text{sizeof(arr[0])}$

```
int i[100];
```

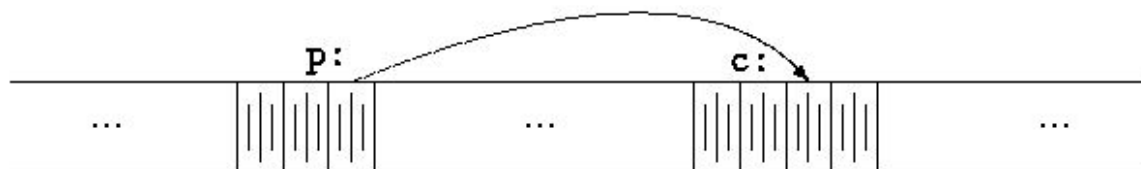
```
double map[100][100]; /* Двумерный */
```

```
int sequence[] = {10, 20, 30}; /* С инициализацией */
```

```
char string[] = "This is madness!\n"; /* Строка - тоже массив */
```

Введение в указатели

- Указатель – переменная, содержащая адрес другой переменной
- & - взятие адреса переменной (взятие адреса объекта)
- * - разыменовывание указателя (доступ к объекту, адрес которого содержит указатель)
- Void * - «бестиповой» указатель
- NULL – нулевой указатель



Введение в указатели - 2



```
int *p = NULL;
```

```
int i = 10;
```

```
p = &i; /* теперь p содержит адрес в памяти, по которому  
находится переменная i */
```

```
*p = *p + 1; /* запись в адрес, находящийся в p его  
содержимого, увеличенного на 1 */
```

```
/* теперь i содержит 11 */
```

```
*p = 0;
```

```
/* теперь i содержит 0 */
```

Строковые константы

- Последовательность (возможно, нулевая) ненулевых символов, заканчивающаяся на 0
- Задание строки возможно помещением текста в двойные кавычки (одинарные кавычки – взятие кода символа)
- Спецсимволы: `\n`, `\r`, `\ooo`, `\xhh`, `\"`, `\'`, `\0`, `\\` и т.д.
- Функция C standard library `strlen(str)` – длина строки без завершающего `\0`



```
char *s1 = "Я строка";  
char *s2 = "Я" " тоже" " строка";  
printf ("Первая строчка\nВторая \"строчка\"");
```

Использование #define для объявления констант.

```
#define MESSAGE "Some frequently used string."  
#define DEBUG 0  
#define BUFSIZE 1024  
#define NEWBUF BUFSIZE*2
```

/ теперь мы можем использовать определённые выше константы*/*

«Истина» и «ложь»

- Если численное значение 0, оно ложно
- Иначе оно истинно
 - Например, 1, 1000, -1234 – все истинны

Операторы

- Арифметические: +, -, *, /, %(остаток от деления)
- Логические: >, >=, <, <=, ==, !=, &&(логическое И), ||(логическое ИЛИ), !(отрицание, $x \neq 0 \Rightarrow 0$, $x == 0 \Rightarrow 1$)
- Побитовые: &(побитовое И), |(побитовое ИЛИ), ^(исключающее ИЛИ), <<(сдвиг влево), >>(сдвиг вправо), ~(дополнение)

Примеры

```
int a, b;  
double x;  
a = 2*2; /* a = 4 */  
b = (a + 8) / 2; /* b = 6 */  
b = 3 / 2; /* b = 1 */  
x = 3 / 2; /* x = 1.5 */  
a = 1234 || 0; /* a != 0 */  
a = 1234 && 0; /* a == 0 */  
a = (1 == 0) || (10 > 1) /* a != 0 */
```

++ И --

- Прибавление/вычитание единицы с последующей записью значения в переменную
- Могут находиться перед переменной (префиксная запись, «сперва прибавить/вычесть, потом использовать») и после (постфиксная запись, наоборот)

Примеры

```
int x = 0, n = 0;
```

```
n++; /* n = 1 */
```

```
x = ++n; /* x = 2, n = 2 */
```

```
x = n++; /* x = 2, n = 3 */
```

+=, -=, *= И Т.Д.

- **A += B эквивалентно A = A + B**
- **+ - * / % << >> & ^ |**

/* Пример */

x = 2;

x *= 5; /* x = 10 */

x -= 3; /* x = 7 */

A ? B : C

- Если A истинна(т.е. не равна 0), то B, иначе C

/ Пример: */*

/ если x кратно 2, то y = 1, иначе 0 */*

y = x % 2 ? 0 : 1;

Преобразование типа.

- Возможное преобразование:
 - автоматическое
 - (type)variable
- “Общее правило” – автоматическое преобразование происходит при преобразовании в более «ёмкий» тип данных (например, char -> long int), при котором преобразование имеет смысл (целочисленный тип -> целочисленный тип; указатель -> указатель)

Правила конверсии для беззнаковых чисел

- If either operand is long double, convert the other to long double.
- Otherwise, if either operand is double, convert the other to double.
- Otherwise, if either operand is float, convert the other to float.
- Otherwise, convert char and short to int.
- Then, if either operand is long, convert the other to long.

Приоритет операторов

Операторы

Наивысший

() [] -> .

! ~ ++ -- + - * (type) sizeof

* / %

+ -

<< >>

< <= > >=

== !=

&

^

|

&&

||

?:

= += -= *= /= %= &= ^= |= <<= >>=

,

Самый низкий

if-else

```
if (expression)
```

```
    statement1
```

```
else
```

```
    statement2
```

- Если expression ИСТИННО, то statement1, иначе – statement2

Примеры

/ 1. else идёт с соседним if-ом! */*

```
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

/ 2. */*

```
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

switch

- Условное выполнение кода при равенстве выражения одной из ЧИСЛЕННЫХ КОНСТАНТ

```
switch (expression) {  
    case const-expr: statements  
    case const-expr: statements  
    default: statements  
}
```

Пример

```
switch (c) {  
    case '0': case '1': case '2':  
case '3': case '4':  
    case '5': case '6': case '7':  
case '8': case '9':  
    ndigit[c-'0']++;  
    break;  
case ' ':  
case '\n':  
case '\t':  
    nwhite++;  
    break;  
default:  
    nother++;  
    break;  
}
```

while - ЦИКЛ

while (expression)
statement

- Выполнение statement пока expression не равен 0

```
/* Пример */  
int i = 0;  
while (i < 10){  
    printf("%d\n", i);  
    i ++;  
}
```

for - ЦИКЛ

```
for (expr1; expr2; expr3)  
    body
```

- содержит 3 выражения, разделённые запятой

Аналогичен следующей конструкции с `while`:

```
while (expr2) {  
    statement  
    expr3;  
}
```

/ Пример */*

```
printf ("напечатать все степени двойки, меньше 1000\n");  
for (i = 1; i < 1000; i *= 2)  
    printf ("%d\n", i);
```

Цикл do-while

do

statement

while (expression);

- Выполнять statement пока expression не равно нулю. statement выполнятся хотя бы раз

Break и continue

- Break: ранний выход из циклов for, while, do
 - Выход осуществляется из самого вложенного цикла
- Continue: начало следующей итерации цикла

Примеры

```
/* trim: remove trailing blanks, tabs, newlines */
int trim(char s[])
{
    int n;

    for (n = strlen(s)-1; n >= 0; n--)
        if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
            break;
    s[n+1] = '\0';
    return n;
}

    for (i = 0; i < n; i++)
        if (a[i] < 0) /* skip negative
elements */
            continue;
    ... /* do positive elements */
```

Goto

- Goto – переход по метке
- Использование оправданно очень редко!
- Есть возможность перехода между функциями – `man longjmp`

```
for (i = 0; i < n; i++)  
    for (j = 0; j < m; j++)  
        if (a[i] == b[j])  
            goto found;  
    /* didn't find any common element */  
    ...  
found:  
    /* got one: a[i] == b[j] */  
    ...
```

ФУНКЦИИ

```
тип_возвращаемого_значения имя_функции (аргумент1,  
аргумент2, ...)  
{  
    ... /* тело функции */  
}
```

- Перед использованием функция должна быть объявлена.
- Тип возвращаемого значения может быть **void**(ничего не возвращает)
- Количество аргументов может быть как целым, так и переменным(**man stdarg**). Кроме того, оно тоже может быть равным нулю (**void**).
- Пустые скобки – количество аргументов не определено!
- Функции не могут быть объявлены внутри других функций

Пример

```
#include <stdio.h>
```

```
int power(int m, int n);
```

```
/* test power function */
```

```
main()
```

```
{
```

```
    int i;
```

```
    for (i = 0; i < 10; ++i)
```

```
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
```

```
    return 0;
```

```
}
```

```
/* power: raise base to n-th power; n >= 0 */
```

```
int power(int base, int n)
```

```
{
```

```
    int i, p;
```

```
    p = 1;
```

```
    for (i = 1; i <= n; ++i)
```

```
        p = p * base;
```

```
    return p;
```

```
}
```

Заголовочные файлы

calc.h

```
#define NUMBER '0'  
void push(double);  
double pop(void);  
int getop(char []);  
int getch(void);  
void ungetch(int);
```

main.c

```
#include <stdio.h>  
#include <stdlib.h>  
#include "calc.h"  
#define MAXOP 100  
main() {  
    ...  
}
```

getop.c

```
#include <stdio.h>  
#include <ctype.h>  
#include "calc.h"  
getop() {  
    ...  
}
```

getch.c

```
#include <stdio.h>  
#define BUFSIZE 100  
char buf[BUFSIZE];  
int bufp = 0;  
int getch(void) {  
    ...  
}  
void ungetch(int) {  
    ...  
}
```

stack.c

```
#include <stdio.h>  
#include "calc.h"  
#define MAXVAL 100  
int sp = 0;  
double val[MAXVAL];  
void push(double) {  
    ...  
}  
double pop(void) {  
    ...  
}
```

external, static, register, volatile переменные

- External – задание внешнего связывания (external linking). Все обращения к external объектам, даже из разных, независимо откомпилированных файлов, происходят к одному и тому же объекту в памяти
- Static - вне функций - ограничение области видимости до файла, внутри – сохранение значения между вызовами для локальных переменных
- Register - рекомендация компилятору держать значение в переменной в регистре процессора
- Volatile - Значение переменной может быть изменено вне программы

Директивы препроцессора

- `#include "name"`, `#include <name>`
- `#define ЧТО_ЗАМЕНИТЬ НА_ЧТО_ЗАМЕНИТЬ`
- `#undef`
- `#if`, `#ifdef`, `#ifndef`, `#elif`, `#endif`

Примеры

```
#define BUFSIZE 1024
#define forever for (;;)
#define max(A, B) ((A) > (B) ? (A) : (B)) /* обратите внимание на
сторонние эффекты! */
```

```
#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
```

Ещё об указателях и массивах

```
swap(a, b);
```

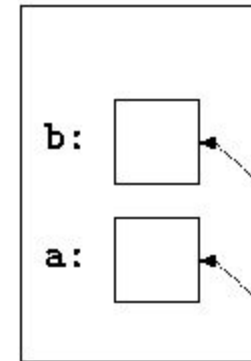
```
void swap(int x, int y) /* НЕПРАВИЛЬНО */  
{  
    int temp;  
  
    temp = x;  
    x = y;  
    y = temp;  
}
```

Ещё об указателях и массивах

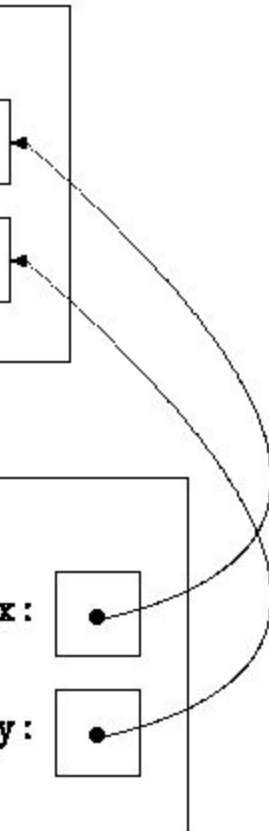
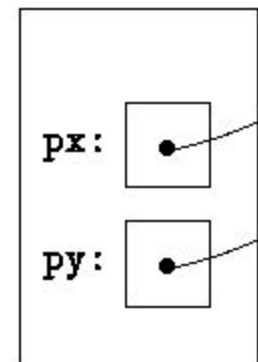
```
swap(&a, &b);
```

```
void swap(int *px, int *py) /*  
поменять местами *px и *py */  
{  
    int temp;  
  
    temp = *px;  
    *px = *py;  
    *py = temp;  
}
```

in caller:

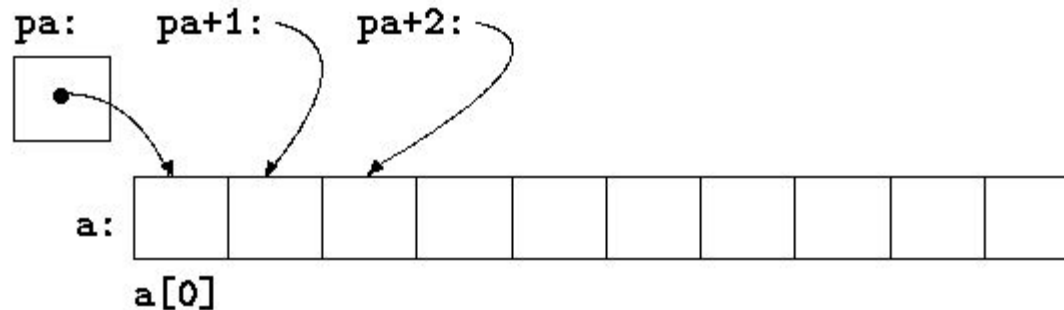


in swap:



Ещё об указателях и массивах

```
int a[10];  
int *pa;  
pa = &a[0];  
x = *pa;  
y = *(pa + 1) /* над указателями возможны арифметические действия */
```



Передача параметров программе. Аргументы main.

- 1 аргумент – количество аргументов
- 2 аргумент – массив строк с аргументами

```
#include <stdio.h>
```

```
/* echo command-line arguments; 2nd version */
```

```
main(int argc, char *argv[])
```

```
{
```

```
    while (--argc > 0)
```

```
        printf("%s%s", *++argv, (argc > 1) ? " " : "");
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

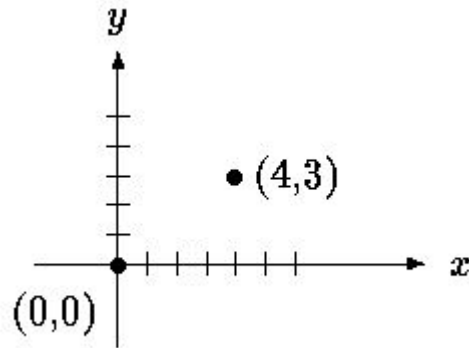
Указатели на функцию

- Возможно задание и передача указателей на функции

```
int (*pf)(); /* указатель на функцию, возвращающую int */  
int (*comp)(int, void *); /* указатель на функцию с целым  
числом в роли первого аргумента и void * указателем в роли  
второго */
```

Структуры

```
struct point {  
    int x;  
    int y;  
};
```



- Набор переменных (возможно, различных типов) объединённых вместе для простоты работы с ними
- `struct struct_name { ... } x, y, z;`
- `p->x` то же самое, что `(*p).x`

typedef

- Задание пользовательского типа данных

```
typedef struct tnode { /* the tree node: */  
    char *word;        /* points to the text */  
    int count;         /* number of occurrences */  
    struct tnode *left; /* left child */  
    struct tnode *right; /* right child */  
} Treenode;
```


Полезные функции C standard library

- printf/scanf, sprintf/sscanf, fprintf/fscanf
- putchar/getchar, putc/getc
- fopen/fclose – работа с файлами
- memchr
- strchr
- malloc/realloc/free
- strlen
- is(alpha/upper/digit/...)
- sin,cos,sqrt,... (-lm в строке компиляции для линковки мат. библиотеки)

Литература

- Б. Керниган, Д. Ритчи – «Язык программирования Си» (Kernigan, Ritchie – “The C Programming Book”) aka “K&R”
- The C standard library: Linux manpages / MSDN
- Б. Керниган, Р. Пайк – «Практика программирования» (Kernigan, Pike – “The Practice of Programming”)

Спасибо за внимание!

Вопросы?

Следующая лекция – Введение в
C++

Backup

Перечисления (enumerations)

/ Предоставляют простой способ задания нескольких констант с различиющимися значениями */*

/ По умолчанию значение первого элемента - 0 */*

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,  
JUL, AUG, SEP, OCT, NOV, DEC };
```

/ Переменной может и не быть: */*

```
enum { UP, DOWN, RIGHT, LEFT };
```

unions

- Структура, члены которой располагаются по одному и тому же адресу памяти

```
union u_tag {  
    int ival;  
    float fval;  
    char *sval;  
} u;
```

БИТОВЫЕ ПОЛЯ

- Задание размеров элементов структур в битах

```
struct {  
    unsigned int is_keyword : 1;  
    unsigned int is_extern : 1;  
    unsigned int is_static : 1;  
} flags;
```