Символы. Структуры. Команды препроцессора

Основы языка C, лекция 3

Повторяем: функции

- Функции

```
float foo (float x, int n);   // прототип
float foo (float x, int n) {  // реализация
    float res = x * n;
    return res;
}
int main ( ) {
    float z = foo (3.5, -10);     // вызов
    return 0;
}
```

Переменные

Переменные	Область видимости	Время жизни	Инициа-лизация
Локальные	до конца блока	до конца блока	undifined
Аргументы функций	до конца функции	1 вызов функции	foo(3)
Глобальные	до конца файла в другом файле: **extern** int x;	выполнение программы	**0**
static	до конца файла до конца блока	выполнение программы	**0**

- Аргументы в функцию передаются **по значению** (копии)

Символы. ASCII таблица

1 ☺	33 !	65 A	97 a	129 ü	161 ¡	193 Á	225 á
2 ☻	34 "	66 B	98 b	130 ,	162 ¢	194 Â	226 â
3 ♥	35 #	67 C	99 c	131 ƒ	163 £	195 Ã	227 ã
4 ♦	36 $	68 D	100 d	132 „	164 ¤	196 Ä	228 ä
5 ♣	37 %	69 E	101 e	133 …	165 ¥	197 Å	229 å
6 ♠	38 &	70 F	102 f	134 †	166 ¦	198 Æ	230 æ
7 •	39 '	71 G	103 g	135 ‡	167 §	199 Ç	231 ç
8 ◘	40 (	72 H	104 h	136 ˆ	168 ¨	200 È	232 è
9 ○	41)	73 I	105 i	137 ‰	169 ©	201 É	233 é
10 ◙	42 *	74 J	106 j	138 Š	170 ª	202 Ê	234 ê
11 ♂	43 +	75 K	107 k	139 ‹	171 «	203 Ë	235 ë
12 ♀	44 ,	76 L	108 l	140 Œ	172 ¬	204 Ì	236 ì
13 ♪	45 -	77 M	109 m	141 ï	173 -	205 Í	237 í
14 ♫	46 .	78 N	110 n	142 Ž	174 ®	206 Î	238 î
15 ☼	47 /	79 O	111 o	143 ï	175 ¯	207 Ï	239 ï
16 ►	48 0	80 P	112 p	144 ï	176 °	208 Ð	240 ð
17 ◄	49 1	81 Q	113 q	145 '	177 ±	209 Ñ	241 ñ
18 ↕	50 2	82 R	114 r	146 '	178 ²	210 Ò	242 ò
19 ‼	51 3	83 S	115 s	147 "	179 ³	211 Ó	243 ó
20 ¶	52 4	84 T	116 t	148 "	180 ´	212 Ô	244 ô
21 §	53 5	85 U	117 u	149 •	181 µ	213 Õ	245 õ
22 ▬	54 6	86 V	118 v	150 –	182 ¶	214 Ö	246 ö
23 ↨	55 7	87 W	119 w	151 —	183 ·	215 ×	247 ÷
24 ↑	56 8	88 X	120 x	152 ˜	184 ¸	216 Ø	248 ø
25 ↓	57 9	89 Y	121 y	153 ™	185 ¹	217 Ù	249 ù
26 →	58 :	90 Z	122 z	154 š	186 º	218 Ú	250 ú
27 ←	59 ;	91 [	123 {	155 ›	187 »	219 Û	251 û
28 ∟	60 <	92 \	124 |	156 œ	188 ¼	220 Ü	252 ü
29 ↔	61 =	93]	125 }	157 ï	189 ½	221 Ý	253 ý
30 ▲	62 >	94 ^	126 ~	158 ž	190 ¾	222 Þ	254 þ
31 ▼	63 ?	95 _	127 ⌂	159 Ÿ	191 ¿	223 ß	255 ÿ
32	64 @	96 `	128 €	160	192 À	224 à	

Ввод и вывод

```
#include <stdio.h>
int main ( )
{
    char c;
    scanf ("%c", & c);
    printf ("Symbol %c has code %d \n", c, c);
    return 0;
}
```

ввели
Q*Enter*
напечатала программа
Symbol Q has code 81

ввели
3*Enter*
напечатала программа
Symbol 3 has code 51

scanf начнет работать после нажатия Enter

Ввод и вывод - 2

```
#include <stdio.h>
int main ( )
{
    char c1, c2;
    scanf ("%c%c",   &c1, &c2);
    printf ("%c%c",  c1, c2);
    return 0;
}
```

ввели
Qa*Enter*
напечатала программа
Qa

ввели
1z2x3c*Enter*
напечатала программа
1z

ввели
Q*Enter*
напечатала программа
Q*перевод строки*

Символьная константа

- 'a', '7', '\n' — код символа (целое число)
- из символа число
- char c = '7'; // 55
  int x; // хотим получить 7
- x = '7' — 48; // математически верно, но не понятно
- x = '7' — '0'; // ok
- Из большой буквы 'W' в маленькую 'w'
  char c1 = 'W'; // c1 + 32 - плохо
  char c2 = c1 — 'A' + 'a'; // toupper(c1);
- Не путайте идентификатор x и константу 'x'

Локализация

- Если у вас национальный алфавит, то он обычно идет не по порядку и — 'A' + 'a' не будет работать
- tolower(c1) — работать будет (стандарная)
  toupper(c1)

typedef — псевдоним типа

- **typedef тип *псевдоним* ;**
  typedef type *alias* ;
- **typedef unsigned int *Length* ;**
- **unsigned int** width;
  Length height;
- Короче
  typedef **unsigned long long *llu***; // uint32
- Изменения в меньшем количестве мест, если вместо unsigned int надо использовать unsigned long

Новый составной тип данных

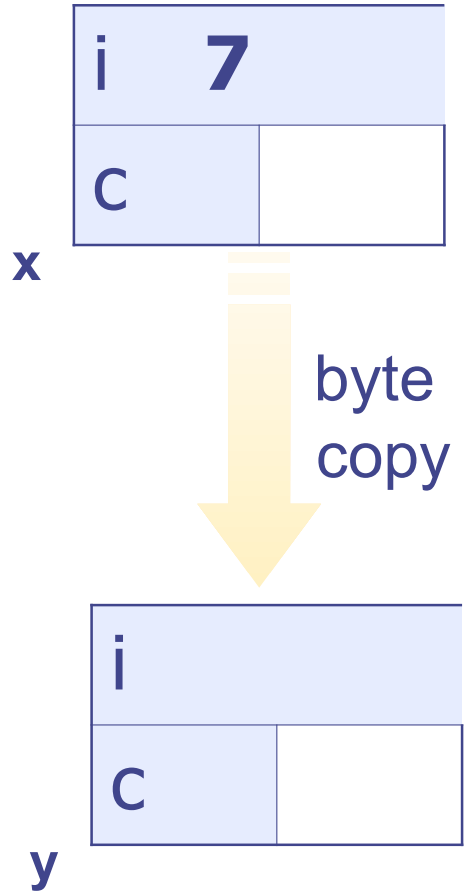
- Объявляется перечислением (можно разнотипных) данных
- опишем студента для зачета по физкультуре
-
```
struct Student {
    long id;
    int birth_year;        // год рождения
    float weight;          // вес, кг
    float height;          // рост, м
    float run100;          // время забега на 100м, сек
    unsigned char pushup;  // сколько отжиманий
    unsigned char pullup;  // подтягиваний
};
```

Объявление типа данных

- struct A {
  int i;
  char c;
  };
- новый тип **struct A**
  - **struct A** является типом в языке C
  - **A** является типом в языке C++ (без struct)
- **A** — имя типа структуры
- **i, c** — поля (fields) структуры
- **;** в конце

Обращение к полю структуры

```
struct A {
    int i;
    char c;
};
struct A  x, y;
x.c = 'z';
x.i = 7;
y.i = x.i + 3;
printf ("%d  %c \n", x.i, x.c);
```



y = x; // побитовое копирование

типы x и y должны совпадать

typedef

средство дать типу псевдоним

```
typedef тип псевдоним;
typedef unsigned int Length;
typedef struct Node Node;
typedef struct {
    int n;
    char c;
} Pair  ;
Pair   x;
```

Явная инициализация

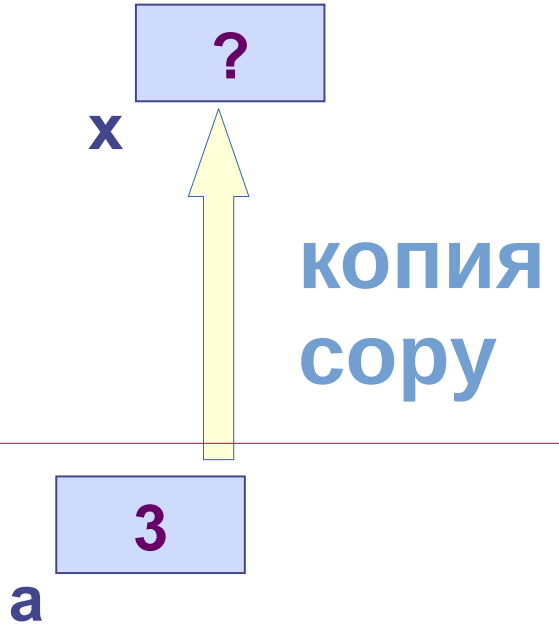
- struct Point {
  int x;
  int y;
  } p1, p2; // объявили переменные p1 и p2
- struct Point maxp = {640, 480};
- struct Rectangle {
  struct Point lt; // left top
  struct Point rb; // right bottom
  } rect = { {0, 0}, {640, 480} } ;
- **rect.rb.x** = 0;

```
struct Point p = {
    .y = 6,
    .x = 11
};
```

Передача аргументов

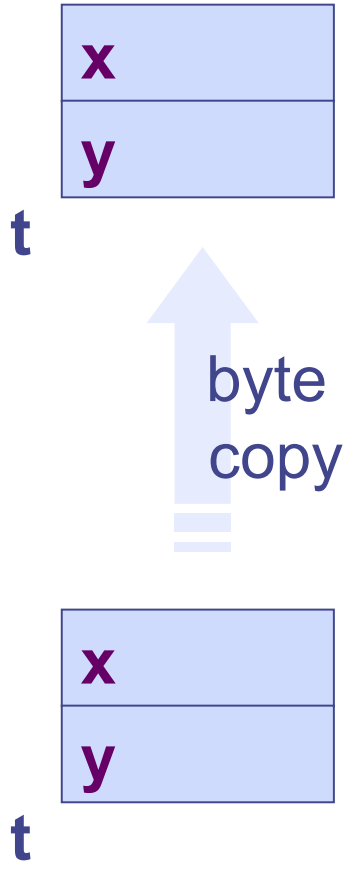
- **Аргументы передаются только по значению (копии)**

```
void inc (int x) {
    x = x + 1;          // x++;
    // a = a + 1;   // нельзя, не видно
}
int main ( ) {
    int a = 3;
    inc (a);
    inc (a);
    printf("%d\n", a);              // ??
}
```



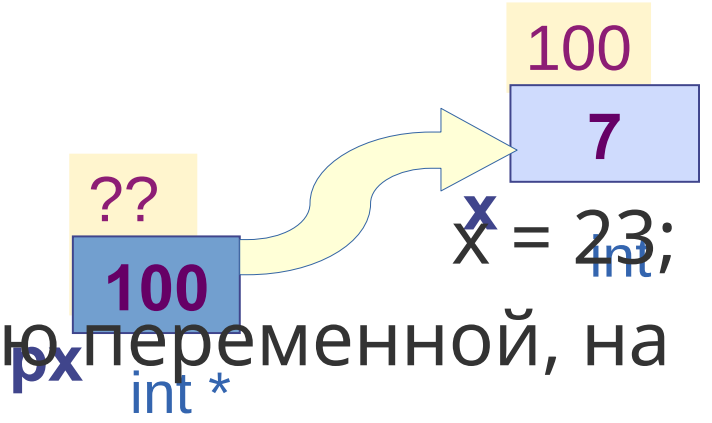
Передается по значению

```
void inc (struct Point t) {
    t.x ++;
}
int main ( ) {
    struct Point t;
    t.x = 2;
    inc ( t );
    inc ( t );
    printf ("%d\n", t.x );
    return 0;
```



Тип "указатель на ..."

- 2 * 3 // умножение
- **int** x = 7;
  int * px; // переменная px типа указатель на int
  // pointer to int, *указывает в никуда*
  px = & x; // & - вычислить адрес переменной
  // связывает px с x



- * px = 23; // * - разыменование x = 23;
  // перейти от указателя к значению переменной, на которую он указывает
  // записать **int** по адресу, который хранится в px
  // потому что переменная px типа **int ***
  * px = * px + 4; x = x + 4;

адрес (pointer), значение (value)

&операция взятия указателя

100
int *

7
int

px = & x;

int * тип данных

int x;

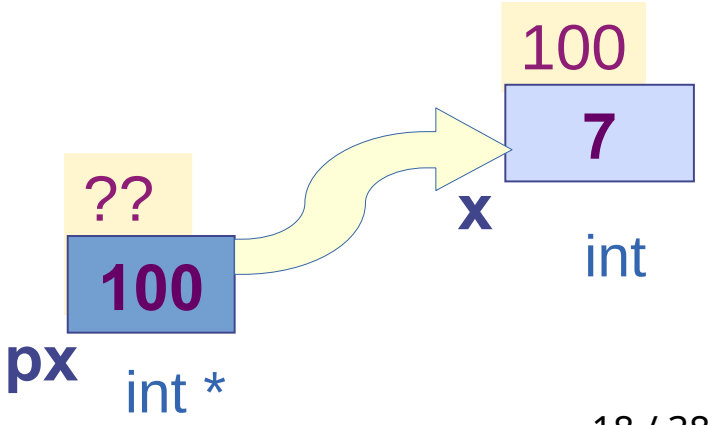
int * px;

x = * px;

7
int

100
int *

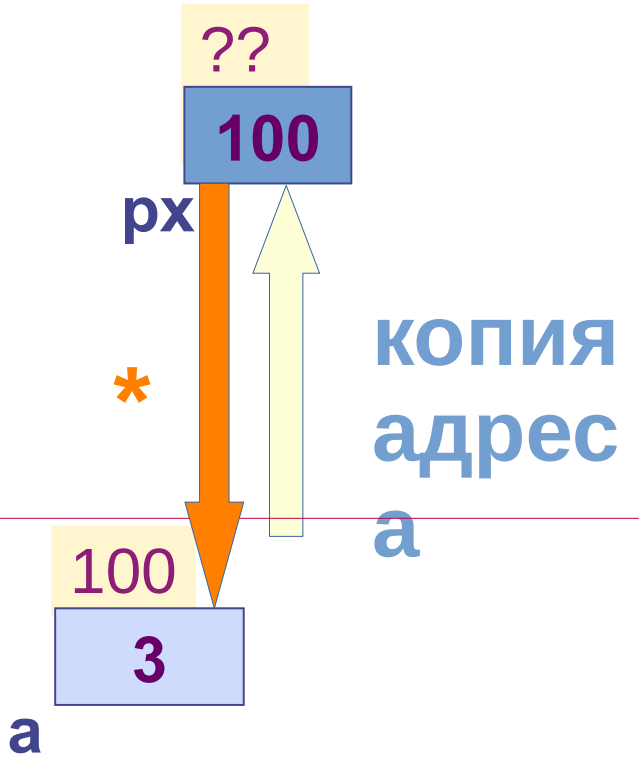
*** операция чтения/записи в память по указателю**



Передаем копию адреса

- **Аргументы передаются только по значению**

```
void inc (int * px) {
    *px = *px + 1;          // (*px)++;
}
```

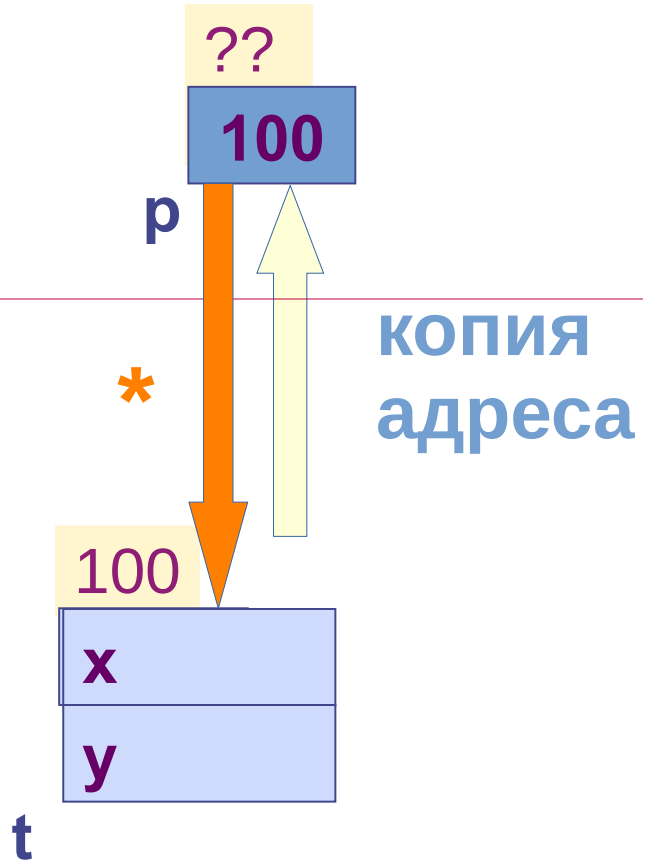



```
int main ( ) {
    int a = 3;
    inc (&a);
    inc (&a);
    printf("%d\n", a);          // ??
}
```

Передаем копию адреса

- Некрасиво (*p).x = (*p).x + 1; или ((*p).x)++;

```
void inc (struct Point * p) {
    p -> x = p -> x + 1;          // p->x ++;
}
```




```
int main ( ) {
    struct Point t;
    t.x = 2;
    inc ( &t );
    inc ( &t );
    printf ("%d\n", t.x );
    return 0;
}
```

Точка и стрелка

- Обращение к полю структуры, если выражение типа
  - Структура — точка **.**
  - Указатель на структуру — стрелка **->**
- **struct Rectangle** rect = { {0, 0}, {640, 480} };
  struct Rectangle ***** r = & rect;
- rect . lt . x = 7; // rect — структура
  // rect . lt - структура
- r -> lt . x = 7; // r — указатель на структуру
  // r -> lt - структура

Копия или указатель?

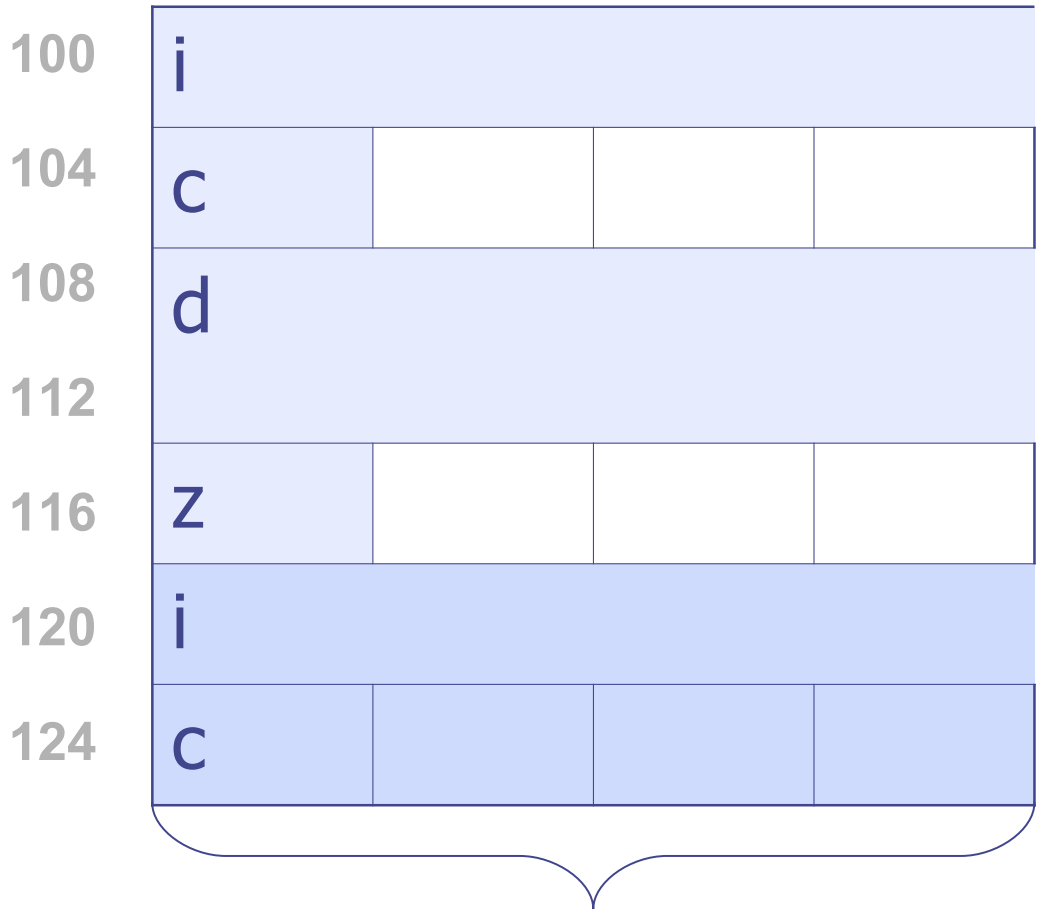
- Если можно обойтись копией:
  void print_point (**struct Point** p);
  float distance (**struct Point** p1, **struct Point** p2);
- Нужно изменить значение :
  - new_point = move1 (old_point, dx);
    struct Point move1(**struct Point** p, int dx);
  - move2 (& point, dx);
    void move2 (**struct Point *** p, int dx);
  - move3 (&new_point, &old_point, dx);
    void move3 (struct Point * dst,
    const struct Point * src, int dx);

sizeof (struct A)

sizeof (struct A) ≠ sizeof (int) + sizeof (char)

```
struct B {
   int  i;
   char       c;
   double     d;
   char       z;
   struct A   a;
   } p;

& p.z
```



Структура в структуре

упоминаемый тип должен быть определен выше

содержит структуру

```
struct Rect {
  struct Point lt;
  struct Point br;
};
```

ссылается на структуру

```
struct Window {
  struct Rect *      display;
  struct Font *      font;
   struct Rect       frame;
};
```

Структура со ссылками на себя

ссылается на себя (Ok)

```
struct Node {
    char * word;
    int n;
    struct Node * left;
    struct Node * right;
};
```

содержит себя (почему ошибка?)

```
struct Node {
  char * word;
  int n;
  struct Node left;
  struct Node right;
};
```

Декларация

A содержит указатель на B,
B содержит указатель на A
Какую структуру надо писать выше?

обещание компилятору
определить структуру B

```
struct B;
struct A {
    struct B * pb;
    int n;
};
struct B {
    struct A * left;
    struct A * right;
};
```

Анонимные структуры

Имя структуры можно опустить

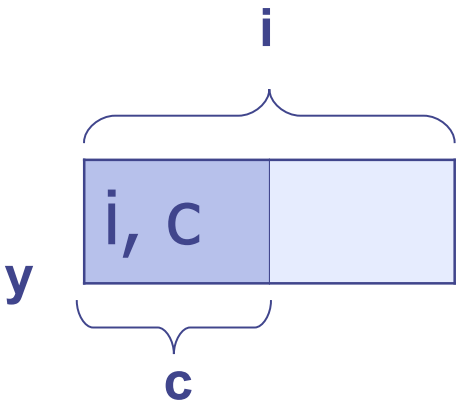
```
struct {
    int n;
    char c;
} p1, p2;
```

мы не сможем определить еще одну переменную данного типа, ибо у него нет имени.

union

```
struct A {
    int i;
    char c;
};
union B {
    int i;
    char c;
};
```

все поля начинаются с одного места в памяти



Битовые поля

не будет на зачете

```
struct {
      unsigned int is_keyword    : 1;
      unsigned int is_extern     : 1;
      unsigned int count         : 5;   // 0 .. 31
} flags;
```

```
flags.is_extern = 1;
flags.count = 7;
if (flags.is_extern && flags.is_keyword)
```

зависит от реализации (переносимость?)

справа-налево или слева-направо

анонимное поле : ширина (для пропуска бит)

выравнивание по границе слова : 0

не могут быть массивами,
нельзя на битовое поле получить указатель

Препроцессор #include

- Работает до компилятора, команды начинаются #
- текстовая замена и условная компиляция (replace)
- **#include** <stdio.h> // стандартные библиотеки
  Заменить эту строку содержимым файла stdio.h
  - системных директориях
  - указанных явно в командной строке -I
- **#include** "my.h" // свой проект и библиотеки
  - сначала в текущей директории запуска gcc
  - указанных явно в командной строке -I
  - системных директориях

#define - макроопределение

- **#define идентификатор** текстовая_замена
  идентификатор — правила C [_a-zA-Z0-9]
  текстовая замена — все до конца строки (поможет \)
  #define A 3
  int x;
  x = A + x; // x = 3 + x;
- Не ставьте в конце ; получите ошибку
  #define A 3 **;**

  x = A + x; // x = 3 **;** + x;

Макросы с аргументами

- sin, sinf, sinl — синусы для разных типов
- #define cube(x) x * x * x
  y = cube(2);
  Подстановка кода, не имеет значение тип
- Время компиляции и время выполнения
- отладка (debug)

Больше скобок

- #define sum(a, b) a + b // будут проблемы!
  y = sum (2, 3) * 4;
  y = 2 + 3 * 4
  #define sub(a, b) (a + b) // так лучше
- #define square(x) (x * x)
  int a = 2, b;
  b = square (a + 3);
  после замены получим:
  b = (a + 3 * a + 3);
- #define square(x) ((x) * (x))

x++ и ++x

- int x = 2;
  printf("%d\n", x++); // 2
  printf("%d\n", x); // 3
  printf("%d\n", ++x); // 4
  printf("%d\n", x); // 4
- int y, x = 2;
  y = ++x + ++x;
  Sequence point
  Undefined behavior
- https://en.cppreference.com/w/cpp/language/eval_order

Как сломать макрос?

```
#define square(x)  ( (x) * (x) )
```

- int x = 2, y;
  y = square(x++);
  y = ((x++) * (x++)) ;
- Не используйте вызовы функций в аргументах макроса
  int c = getchar( );
  isdigit(c = getchar())

  #isdigit(x) ('0' <= (x) && (x) <= '9')

Условная компиляция

- #define DEBUG
  #define _WIN64
- #ifdef __linux__ // #if defined (__linux__)
  много кода, который будет работать в Linux
  #elif defined (_WIN64)
  много кода для 64-битной Windows
  #else
  printf ("Какая у вас операционная система?\n");
  #endif

Не посылать main

- #define AAA // безумный идентификатор

```
#ifdef AAA
#include <stdio.h>
float a3 (float x);
int main ( ) {
    float z;
    scanf("%f", &z);
    return 0;
}
#endif
float a3 (float x) {
    return x*x*x;
}
```

Закомментируем define
// #define AAA
перед посылкой на сервер

или (еще лучше):
gcc **-D**AAA ...
#define AAA писать не нужно

- __LINE__
- __FUNCTION__
- __FILE__
- void foo ( ) {
  printf("%d: %s\n", __LINE__, __FUNCTION__);
  }
- "бревно" "бревно"