

# Программирование микроконтроллеров

## Основы языка Си (продолжение)

# Краткое содержание предыдущих серий

```
1
2 // Объявление функции
3 int summ(int a, int b);
4
5 // Объявление глобальной переменной
6 extern int global_var;
7
8 // определение глобальной переменной
9 int global_var;
10
11 // Определение функции
12 int summ(int a, int b)
13 {
14     return a + b;
15 }
```

```
18 int main()
19 {
20     // Вызов функций выглядит так
21     summ(100, 200);
22     int result = summ(10, 20);
23     printf("result = %d\n", result);
24
25     // Переменные можно инициализировать
26     // при определении
27     int var = 100;
28     return 0;
29 }
```

# Краткое содержание предыдущих серий

```
31
32 // Глобальная переменная
33 int global_var;
34
35 int main()
36 {
37     // Локальная переменная
38     int local_var;
39     // Локальная статическая переменная
40     static int sl_var;
41
42     return 0;
43 }
44
```

```
32 int main()
33 {
34     // Константы
35     const int const_var = 100;
36
37     // Арифметика
38     int a = 10, b = 30, c = 40;
39     int d = (a + b) / c * 500 - 24;
40
41     // Логические операции
42     int t = 1, f = 0, res;
43     res = t || f; // 1
44     res = t && f; // 0
45
46     // Побитовые логические операции
47     res = 0b010 | 0b110; // 0b110
48     res = 0b010 & 0b110; // 0b010
49     res = 0b010 ^ 0b110; // 0b100 // XOR
50
51     // Сравнения
52     res = t == f; // 0
53     res = 100 < 10; // 0
54     res = a >= 10; // 1
55
56     return 0;
57 }
```



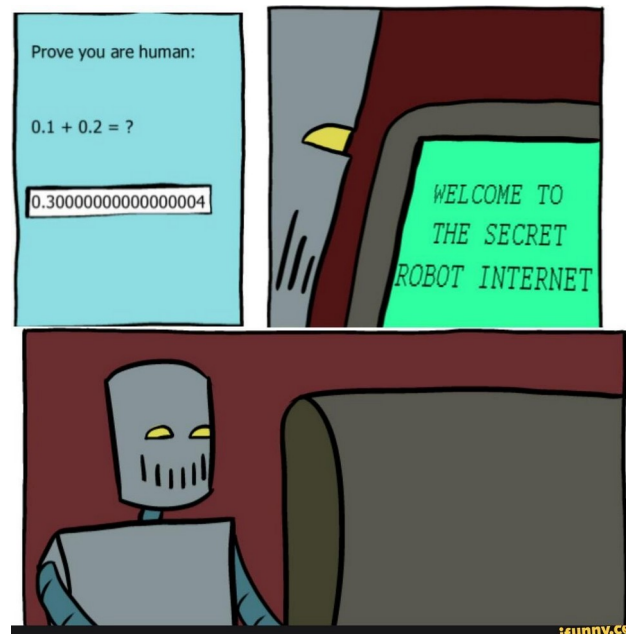
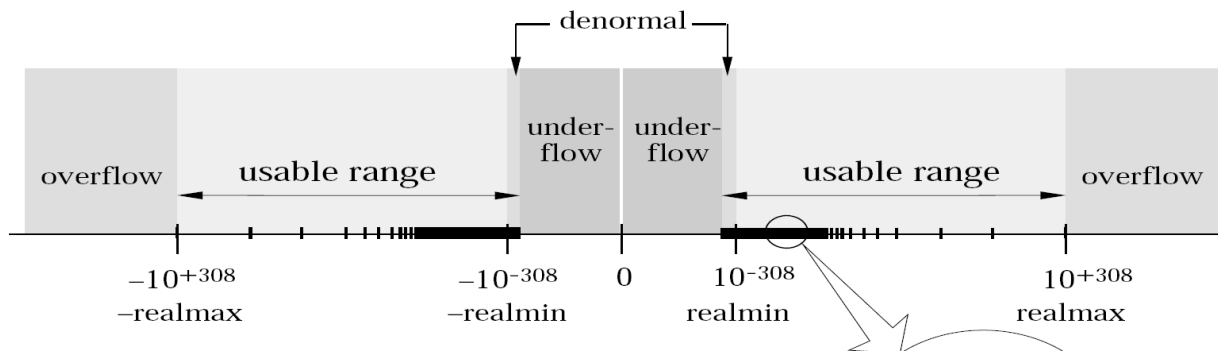
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

# Краткое содержание предыдущих серий

```
32 int main()
33 {
34     // 1 байт [-128..+128]
35     int8_t i8;
36     // 2 байта [-32768..+32767]
37     int16_t i16;
38     // 4 байта [-2147483648..+2147483647]
39     int32_t i32;
40     // 8 байт [-9223372036854775808..+9223372036854775807]
41     int64_t i64;
42
43     // 1 байт [0..255]
44     uint8_t ui8;
45     // 2 байта [0..65535]
46     uint16_t ui16;
47     // 4 байта [0..4294967295]
48     uint32_t ui32;
49     // 8 байт [0..18446744073709551615]
50     uint64_t ui64;
51
52     return 0;
53 }
```

# Краткое содержание предыдущих серий

```
31
32 int main()
33 {
34     // [1.175494351e-38 ... 3.402823466e+38]
35     // ~7 знаков точности
36     float float_var;
37     // [2.2250738585072014e-308 ... 1.7976931348623158e+308]
38     // ~15 знаков точности
39     double double_var;
40
41     return 0;
42 }
```



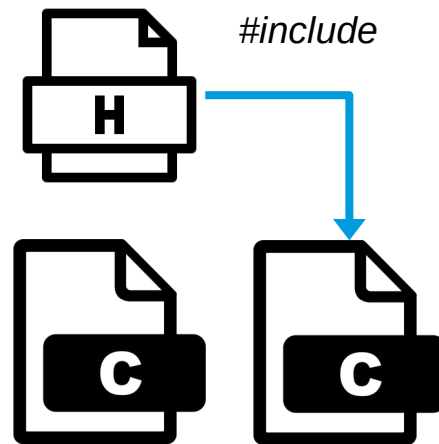
# Краткое содержание предыдущих серий

```
4 int function()  
5 {  
6     int x;  
7     int y = 10;  
8     // x не инициализированна  
9     int a = x + y; // <- UB!  
10    printf("%d\n", a);  
11  
12    // Переполнение знакового целого  
13    int8_t value = 10000000000000000000; // <- UB!  
14  
15    // Здесь нет return  
16    // <- UB!  
17 }  
18  
19  
20 int main()  
21 {  
22     int var = function(); // x = ?  
23 }  
24
```



# Краткое содержание предыдущих серий

```
1
2 // Включение заголовочных файлов
3 #include <stdint.h>
4 #include <stdbool.h>
5 #include <stdio.h>
6
7 // Определение макроса
8 #define MACRO 42
9
10
11 int main()
12 {
13     int x = MACRO;
14     printf("x = %d\n", x);
15
16     return 0;
17 }
18 |
```



# Управляющие конструкции - if

```
6 // Высота выброса парашюта (метры)
7 #define CHUTE_DEPLOY_ALT 100
8
9
10 int main()
11 {
12     // Получаем высоту с датчика
13     int alt = baro_alt();
14     // Пора ли выпускать парашют?
15     if (alt < CHUTE_DEPLOY_ALT)
16     {
17         // Пора, выпускаем!
18         deploy_chute();
19     }
20     else
21     {
22         // Нет, еще не пора
23         // сделаем что-то другое
24         int speed = measure_airspeed();
25     }
26
27     return 0;
28 }
```

Ветвление программы делается через управляющую конструкцию «if»

**If (условие) { если условие верно }  
else { если не верно }**

Скобочки вокруг условия обязательны.  
else можно не писать



# Управляющие конструкции - if

```
6 // Высота выброса парашюта (метры)
7 #define CHUTE_DEPLOY_ALT 100
8
9
10 int main()
11 {
12     // Получаем высоту с датчика
13     int alt = baro_alt();
14     // Пора ли выпускать парашют?
15     if (alt < CHUTE_DEPLOY_ALT)
16     {
17         // Пора, выпускаем!
18         deploy_chute();
19     }
20     else
21     {
22         // Нет, еще не пора
23         // сделаем что-то другое
24         int speed = measure_airspeed();
25     }
26
27     return 0;
28 }
```

Ветвление программы делается через управляющую конструкцию «if»

**If (условие) { если условие верно }  
else { если не верно }**

Скобочки вокруг условия обязательны.  
else можно не писать

# Управляющие конструкции - if

```
10 int main()
11 {
12     // Определяемся с тем, сколько зонд будет спать
13     // между замерами
14     bool sleep_time_ms;
15
16     // Мы уже приземлились?
17     if (is_landed())
18     »     sleep_time_ms = 1000; // Спим целую секунду
19     else
20     »     sleep_time_ms = 100; // Спим 0.1 секунды
21
22     // Спим, только решили спать
23     sleep(sleep_time_ms);
24
25     return 0;
26 }
27
```

{ } можно не писать, если в «then» или «else» секциях находится один оператор

# Управляющие конструкции - if

```
10 int main()  
11 {  
12     // Определяемся с тем, сколько зонд будет спать  
13     // между замерами  
14     bool sleep_time_ms;  
15  
16     // Мы уже приземлились?  
17     if (is_landed())  
18     »     sleep_time_ms = 1000; // Спим целую секунду  
19     else  
20     »     sleep_time_ms = 100; // Спим 0.1 секунды  
21  
22     // Спим только решили спать  
23     sleep(sleep_time_ms);  
24  
25     return 0;  
26 }  
27
```

{ } можно не писать, если в «then» или «else» секциях находится один оператор

# Управляющие конструкции - if

Но нужно быть осторожнее

```
10 int main()
11 {
12     // Определяемся с тем сколько зонд будет спать
13     // между замерами
14     bool sleep_time_ms;
15
16     // Мы уже приземлились?
17     if (is_landed())
18     »     sleep_time_ms = 1000; // Спим целую секунду
19     »     enable_beeper(); // Включаем пищалку
20
21     // Спим только решили спать
22     sleep(sleep_time_ms); // А вот тут кстати UB
23
24     return 0;
25 }
26 |
```

# Управляющие конструкции - if

```
10 int main()
11 {
12     // Проверяем выдвинулась ли антенна
13     int ant_ok = is_antenna_deployed();
14     if (ant_ok == 1) // вместо ==
15     {
16         // Передаем телеметрию
17         transmit_telemetry_frame();
18     }
19
20     // YODA нотация
21     if (1 == ant_ok)
22     {
23         // ...
24     }
25
26     // Правильная работа с булевыми типами
27     if (ant_ok)
28     {
29         // ...
30     }
31
32     return 0;
33 }
```

Со строгим равенством тоже  
нужна осторожность

Операция присваивания тоже  
имеет результат

$a = (b = c)$

Работает похожим образом с

$a = (b + c)$

# Управляющие конструкции - switch

```
6 int main()
7 {
8     int day_no = get_day_of_week();
9
10    if (1 == day_no)
11        printf("Понедельник\n");
12    else if (2 == day_no)
13        printf("Вторник\n");
14    else if (3 == day_no)
15        printf("Среда");
16    else if (4 == day_no)
17        printf("Четверг");
18    else if (5 == day_no)
19        printf("Пятница");
20    else if (6 == day_no)
21        printf("Суббота");
22    else if (7 == day_no)
23        printf("Воскресенье");
24    else
25        printf("Вы там с ума сошли?");
26
27    return 0;
28 }
```

Задача — напечатать  
названия дня недели по его  
номеру

Через if много писанины

# Управляющие конструкции - switch

```
5
6 int main()
7 {
8     const int day_no = get_day_of_week();
9
10    switch (day_no)
11    {
12        case 1: printf("Понедельник\n"); break;
13        case 2: printf("Вторник\n"); break;
14        case 3: printf("Среда\n"); break;
15        case 4: printf("Чертверг\n"); break;
16        case 5: printf("Пятница\n"); break;
17        case 6: printf("Суббота\n"); break;
18        case 7: printf("Воскресенье\n"); break;
19        default:
20            printf("Не бывает такого дня");
21            /* break */
22    }
23
24    return 0;
25 }
26 |
```

switch (<что сравниваем>)

{

case <значение>:

<выполняемый код>

default:

<если все мимо>

}

# Управляющие конструкции - switch

```
6 int main()
7 {
8     const int day_no = get_day_of_week();
9
10    switch (day_no)
11    {
12        case 1:
13        case 2:
14        case 3:
15            printf("It`s wednesday!");
16        case 4:
17        case 5:
18            printf("Будний день");
19            break;
20
21        default:
22        case 6:
23        case 7:
24            printf("Выходной день");
25            break;
26    }
27
28    return 0;
29 }
```

**break** — выход из **switch**. Без него выполнение пойдёт насквозь до самого конца блока **switch**



# Перечисления - ENUM

```

5
6 // Определяем собственный тип данных!
7 enum day_of_the_week_t
8 {
9     // Понедельник и так далее
10    DOTW_MONDAY = -1,
11    DOTW_TUESDAY,
12    DOTW_WEDNESDAY,
13    DOTW_THURSDAY,
14    DOTW_SATURDAY,
15    DOTW_SUNDAY = 100,
16 };
17

```

## Что-то среднее между макросами и интами

```

18
19 // Функция может возвращать такой тип данных
20 enum day_of_the_week_t get_day();
21
22 int main()
23 {
24     // Переменные могут его хранить
25     const enum day_of_the_week_t day = get_day();
26
27     // Свич писать по нему значительно приятнее
28     switch(day)
29     {
30     case DOTW_MONDAY:
31     case DOTW_TUESDAY:
32     case DOTW_WEDNESDAY:
33     case DOTW_WEDNESDAY:
34     case DOTW_THURSDAY:
35         printf("Будний день");
36         break;
37
38     case DOTW_SATURDAY:
39     case DOTW_SUNDAY:
40         printf("Выходной день");
41         break;
42     }
43
44     return 0;
45 }
46

```

# Перечисления - ENUM

```
22 int main()  
23 {  
24     // Неявно преобразуются к интам  
25     function_of_int(DOTW_MONDAY); // Никаких проблем  
26     int day = DOTW_MONDAY; // Тоже никаких проблем  
27     day = DOTW_MONDAY + 1000; // Опять никаких проблем  
28  
29     enum day_of_the_week_t enum_day = 100; // СНОВА НИКАКИХ ПРОБЛЕМ  
30  
31     return 0;  
32 }  
33
```

# typedef

```
7 int main()
8 {
9     typedef float temperature;
10    temperature t = 10.f;
11
12    return 0;
13 }
```

```
5
6 enum demo_enum_type
7 {
8     demo_value = 10,
9 };
10
11 typedef enum demo_enum_type demo_enum;
12
```

**typedef <кого назвать> <как>;**

```
5
6 typedef enum day_of_the_week_t
7 {
8     // Понедельник и так далее
9     DOTW_MONDAY = -1,
10    DOTW_TUESDAY,
11    DOTW_WEDNESDAY,
12    DOTW_THURSDAY,
13    DOTW_SATURDAY,
14    DOTW_SUNDAY = 100,
15 } day_of_the_week_t;
16
```

# Енумы и макросы в дикой природе

```
255
256 typedef enum sx1280_mod_lora_sf_t
257 {
258     SX1280_MOD_LORA_SF_5   = 0x50,
259     SX1280_MOD_LORA_SF_6   = 0x60,
260     SX1280_MOD_LORA_SF_7   = 0x70,
261     SX1280_MOD_LORA_SF_8   = 0x80,
262     SX1280_MOD_LORA_SF_9   = 0x90,
263     SX1280_MOD_LORA_SF_10  = 0xA0,
264     SX1280_MOD_LORA_SF_11  = 0xB0,
265     SX1280_MOD_LORA_SF_12  = 0xC0,
266 } sx1280_mod_lora_sf_t;
267
268
269 typedef enum sx1280_mod_lora_bw_t
270 {
271     SX1280_MOD_LORA_BW_1600 = 0x0A,
272     SX1280_MOD_LORA_BW_800  = 0x18,
273     SX1280_MOD_LORA_BW_400  = 0x26,
274     SX1280_MOD_LORA_BW_200  = 0x34,
275 } sx1280_mod_lora_bw_t;
276
```

```
3
4 // =====
5 // Регистры
6 // =====
7 #define NRF24_REGADDR_CONFIG      0x00
8 #define NRF24_REGADDR_EN_AA      0x01
9 #define NRF24_REGADDR_EN_RXADDR  0x02
10 #define NRF24_REGADDR_SETUP_AW    0x03
11 #define NRF24_REGADDR_SETUP_RETR  0x04
12 #define NRF24_REGADDR_RF_CH       0x05
13 #define NRF24_REGADDR_RF_SETUP    0x06
14 #define NRF24_REGADDR_STATUS      0x07
15 #define NRF24_REGADDR_OBSERVE_TX  0x08
16 #define NRF24_REGADDR_RPD         0x09
17 #define NRF24_REGADDR_RX_ADDR_P0  0x0A
18 #define NRF24_REGADDR_RX_ADDR_P1  0x0B
19 #define NRF24_REGADDR_RX_ADDR_P2  0x0C
20 #define NRF24_REGADDR_RX_ADDR_P3  0x0D
21 #define NRF24_REGADDR_RX_ADDR_P4  0x0E
22 #define NRF24_REGADDR_RX_ADDR_P5  0x0F
23 #define NRF24_REGADDR_TX_ADDR     0x10
24 #define NRF24_REGADDR_RX_PW_P0    0x11
25 #define NRF24_REGADDR_RX_PW_P1    0x12
26 #define NRF24_REGADDR_RX_PW_P2    0x13
27 #define NRF24_REGADDR_RX_PW_P3    0x14
28 #define NRF24_REGADDR_RX_PW_P4    0x15
29 #define NRF24_REGADDR_RX_PW_P5    0x16
30 #define NRF24_REGADDR_FIFO_STATUS  0x17
31 #define NRF24_REGADDR_DYNPD        0x1C
32 #define NRF24_REGADDR_FEATURE      0x1D
33
```

# switch и enum в дикой природе

```
84     uint8_t value;
85     switch (sf)
86     {
87     case SX1280_MOD_LORA_SF_5:
88     case SX1280_MOD_LORA_SF_6:
89         value = 0x1E;
90         break;
91
92     case SX1280_MOD_LORA_SF_7:
93     case SX1280_MOD_LORA_SF_8:
94         value = 0x37;
95         break;
96
97     case SX1280_MOD_LORA_SF_9:
98     case SX1280_MOD_LORA_SF_10:
99     case SX1280_MOD_LORA_SF_11:
100    case SX1280_MOD_LORA_SF_12:
101        value = 0x32;
102        break;
103
104    default:
105        // Сверху перебраны все возможные правильные значения
106        return SX1280_ERROR_BAD_ARG;
107    };
108
109    const uint16_t reg_addr1 = 0x0925;
110    int rc = sx1280_brd_reg_write(drv->api.brd, reg_addr1, &value, 1);
111    SX1280_RETURN_IF_ERROR(rc);
```

# Массивы и циклы

```
6 // Высота выброса парашюта (м)
7 #define CHUTE_DEPLOY_ALT (100)
8
9
10 // Функция возвращает текущую высоту по барометру
11 // Внимание, очень сильно шумит!
12 int get_baro_alt(void);
13
14
15 int main()
16 {
17     int alt = get_baro_alt();
18     if (alt <= CHUTE_DEPLOY_ALT)
19     {
20         // Пока выпускать!
21     }
22
23     return 0;
24 }
25
```

Известно что датчик давления очень сильно шумит. Но при этом очень быстро работает. Необходимо отфильтровать его показания

# Массивы и циклы

```
~
6 // Высота выброса парашюта (м)
7 #define CHUTE_DEPLOY_ALT (100)
8
9
10 // Функция возвращает текущую высоту по барометру
11 // Внимание, очень сильно шумит!
12 int get_baro_alt(void);
13
14
15 int main()
16 {
17     int alt = get_baro_alt();
18     if (alt <= CHUTE_DEPLOY_ALT)
19     {
20         // Пора выпускать!
21     }
22
23     return 0;
24 }
25
```

Известно что датчик давления очень сильно шумит. Но при этом очень быстро работает. Необходимо отфильтровать его показания.

Самое простое что можно придумать — запросить несколько значений и посчитать среднее

# Массивы и циклы

```
5
6 // Высота выброса парашюта (м)
7 #define CHUTE_DEPLOY_ALT (100)
8
9
10 // Функция возвращает текущую высоту по барометру
11 // Внимание, очень сильно шумит!
12 int get_baro_alt(void);
13
14
15 int main()
16 {
17     int alt0 = get_baro_alt();
18     int alt1 = get_baro_alt();
19     int alt2 = get_baro_alt();
20     int alt3 = get_baro_alt();
21     int alt4 = get_baro_alt();
22     int alt5 = get_baro_alt();
23     int alt = (alt0 + alt1 + alt2 + alt3 + alt4 + alt5) / 6;
24     if (alt <= CHUTE_DEPLOY_ALT)
25     {
26         // Пора выпускать!
27     }
28
29     return 0;
30 }
31
```

Можно сделать так. Но если этого окажется не достаточно и нужно будет сделать 100 замеров? Или 1000?



# Массивы и циклы

```
6 // Высота выброса парашюта (м)
7 #define CHUTE_DEPLOY_ALT (100)
8
9
10 // Функция возвращает текущую высоту по барометру
11 // Внимание, очень сильно шумит!
12 int get_baro_alt(void);
13
14
15 int main()
16 {
17     int arr[6];
18     arr[0] = get_baro_alt();
19     arr[1] = get_baro_alt();
20     arr[2] = get_baro_alt();
21     arr[3] = get_baro_alt();
22     arr[4] = get_baro_alt();
23     arr[5] = get_baro_alt();
24
25     int alt = (arr[0] + arr[1] + arr[2]
26             + arr[3] + arr[4] + arr[5]) / 6;
27     if (alt <= CHUTE_DEPLOY_ALT)
28     {
29         // Пора выпускать!
30     }
31
32     return 0;
33 }
```

Массив это упорядоченный набор переменных одного и того же типа. Определяется как одиночная переменная, но в конце дописывается размер массива в `[]`.

Для доступа к элементу массива пишется его имя и в `[]` номер элемента.

## Нумерация с нуля

# Цикл for

```
6 // Высота выброса парашюта (м)
7 #define CHUTE_DEPLOY_ALT (100)
8
9
10 // Функция возвращает текущую высоту по барометру
11 // Внимание, очень сильно шумит!
12 int get_baro_alt(void);
13
14
15 int main()
16 {
17     int arr[6];
18     int i;
19     for (i = 0; i < 6; i++)
20     »   arr[i] = get_baro_alt();
21
22     int mean = 0;
23     for (int i = 0; i < 6; i++)
24     »   mean += arr[i];
25
26     int alt = mean / 6;
27     if (alt <= CHUTE_DEPLOY_ALT)
28     {
29     »   // Пора выпускать!
30     »   }
31
32     return 0;
33 }
```

## Цикл for

for (

<то, что происходит при входе>;

<условие выхода>;

<то, что происходит после каждой итерации>

{

<то что повторяется>

}

Например for ( ; ; ) { ... } - бесконечный цикл

# Цикл while

```
6 int main()  
7 {  
8     // Ждем пока сенсор будет готов  
9     while(!sensor_is_ready())  
10    {  
11        sleep(100);  
12    }  
13  
14    // Проснулся, теперь меряем  
15    int data = measure();  
16    return 0;  
17 }  
18 |
```

## Цикл while

while (<условие>) {

<то что повторяется>

}

В программах для МК функция main очень часто включает цикл while(1) { }

# Цикл do-while

## Цикл do-while

```
6 int main()  
7 {  
8     bool sensor_is_ready = false;  
9  
10    do {  
11        send_telemetry();  
12        check_landing();  
13        sensor_is_ready = check_sensor();  
14    } while (!sensor_is_ready);  
15  
16    return 0;  
17 }
```

do {

<то, что повторяется>

} while(<условие>);

Используется довольно редко. Всегда выполняет тело хотя бы один раз

# break и continue

```
9  >> while(1)
10 >> {
11 >>     bool should_stop = check_stop();
12 >>     if (should_stop)
13 >>     {
14 >>         shut_down_beeper();
15 >>         break; // Немедленный выход из цикла
16 >>     }
17 >> }
18
19 >> while(1)
20 >> {
21 >>     int got_message = check_message();
22 >>     if (!got_message)
23 >>         continue;
24
25 >>     // какая-то обработка сообщения
26 >>     // но только если получили
27 >> }
```

**break** — немедленно останавливает цикл. Тело прекращает работу тут же.

**continue** — немедленный переход к следующей итерации.

Работает во всех видах циклов.

Кстати { } можно не писать, если зацикливается один оператор. Точно так же как с if.

# Многомерные массивы и инициализация

```
6 int main()
7 {
8     // Многомерные массивы
9     int two_dim[2][20];
10    int three_dim[2][4][100];
11    two_dim[0][0] = 10;
12    three_dim[1][2][3] = 42;
13
14    // Инициализация массивов
15    int non_intialized[10]; // Не инициализирован
16    int explicit_init[4] = {1, 2, 3, 4}; // Явно указаны все члены
17    int implicit_init[4] = {1, 2}; // Что не указано то 0
18    int auto_size[] = {1, 2, 3, 4, 5}; // Автоматический размер
19
20    int two_dim_init[10][10] = {
21        >> {0, 1, 2, 50},
22        >> {0},
23        >> {42, 42, 42}
24    }; // Все что явно не указано - то 0
25
26    return 0;
27 }
28 |
```

# sizeof и UB

```
6 int main()
7 {
8     >> const int size = get_array_size();
9     >> int dynamic_array[size]; // Грешновато. Работает очень не везде
10
11     >> // Вот так делать вообще нельзя
12     >> int array[10] = {0};
13     >> array[10] = 10; // UB!
14     >> array[100500] = 100; // UB!
15     >> array[-1] = 1000; // VERY UB!
16
17     >> // Арифметика не определена для массивов
18     >> int x = array0 + array1; //?
19     >> // Размер массива изменять нельзя
20     >> int array0[10] = 100; // Неопределенненько
21
22     >> // Размер массива удобно "посчитать" через sizeof
23     >> int array_size = sizeof(array0) / sizeof(array0[0]);
24
25     >> // При использовании в качестве аргументов функций
26     >> // массивы упрощаются до указателей.
27     >> // Об этом чуть позже
28 }
```

# Строки

```
18 int main()
19 {
20     // Эти записи эквивалентны
21     // const обязательно!
22     const char string0[] = "Hello!";
23     const char string1[] = {
24         'H', 'e', 'l', 'l', 'o', '!', 0
25     };
26     const char string2[] = {
27         0x48, 0x65, 0x6C, 0x6C, 0x30, 0x21, 0x00
28     };
29     print_string_bytes(string0, sizeof(string0));
30     print_string_bytes(string1, sizeof(string1));
31     print_string_bytes(string2, sizeof(string1));
32
33     // С русскими буквами сложнее
34     const char string3[] = "Привет!";
35     print_string_bytes(string3, sizeof(string2));
36 }
37
```

```
6 void print_string_bytes(const char string[], int size)
7 {
8     printf("bytes of \"%s\":\n", string);
9     for (int i = 0; i < size; i++)
10     {
11         unsigned char byte = string[i];
12         printf("0x%02X", (int)byte);
13     }
14     printf("\n");
15 }
```

```
<terminated> (exit value: 0) cansat-lection Debug [C/C++ Application] /home/_crypto/snork/prog/eclips
bytes of "Hello!":
0x48 0x65 0x6C 0x6C 0x6F 0x21 0x00
bytes of "Hello!":
0x48 0x65 0x6C 0x6C 0x6F 0x21 0x00
bytes of "Hello!":
0x48 0x65 0x6C 0x6C 0x6F 0x21 0x00
bytes of "Привет!":
0xD0 0x9F 0xD1 0x80 0xD0 0xB8 0xD0 0xB2 0xD0 0xB5 0xD1 0x82 0x21 0x00
```



# СИМВОЛЫ

```
5
6 int main()
7 {
8     int var = 'A';
9     int delta = 'A' - 'a';
10    int utf = '0';
11
12    printf("%d, %d, %d\n", var, delta, utf);
13 }
14
```

Problems Tasks Con  
<terminated> (exit value: 0  
65, -32, 53412

```
22:48:51 **** Incremental Build of configuration Debug for project cansat-lection ****
make all
Building file: ../src/project/main.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/project/main.d" -MT"src/project/main.o"
../src/project/main.c: В функции «main»:
../src/project/main.c:10:19: предупреждение: многознаковая символьная константа [-Wmultichar]
   10 |         int utf = '0';
       |                   ^~~
Finished building: ../src/project/main.c
```

# Таблицы ASCII и Unicode

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL	8 BS	9 HT	A LF	B VT	C FF	D CR	E SO	F SI
1	16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB	24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
2	32 SPACE	33 EXCLAM. MARK	34 QUOT. MARK	35 NUM SIGN	36 DOLLAR SIGN	37 PERCENT SIGN	38 AMPERSAND	39 APOSTROPHE	40 LEFT PAREN.	41 RIGHT PAREN.	42 ASTERISK	43 PLUS SIGN	44 COMMA	45 HYPHEN- MINUS	46 FULL STOP	47 SOLIDUS
3	48 DIGIT ZERO	49 DIGIT ONE	50 DIGIT TWO	51 DIGIT THREE	52 DIGIT FOUR	53 DIGIT FIVE	54 DIGIT SIX	55 DIGIT SEVEN	56 DIGIT EIGHT	57 DIGIT NINE	58 COLON	59 SEMI- COLON	60 LESS- THAN SIGN	61 EQUALS SIGN	62 GREATER- THAN SIGN	63 QUESTION MARK
4	64 COMMERCIAL AT	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 LEFT SQ. BRACKET	92 REVERSE SOLIDUS	93 RIGHT SQ. BRACKET	94 CIRCUM- FLEX ACCENT	95 LOW LINE
6	96 grave ACCENT	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 L. CURLY BRACKET	124 VERTICAL LINE	125 R. CURLY BRACKET	126 TILDE	127 DEL

ASCII code table including entity references, control codes and Unicode names (1.1)

© Tom Gibara July 2014

# Таблицы ASCII и Unicode

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

- Latin script
- Non-Latin European scripts
- African scripts
- Middle Eastern and Southwest Asian scripts
- South and Central Asian scripts
- Southeast Asian scripts
- East Asian scripts
- CJK characters
- Indonesian and Oceanic scripts
- American scripts
- Notational systems
- Symbols
- Private use
- UTF-16 surrogates

As of Unicode 15.0

100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
110	111	112	113	114	115	116	117	118	119	11A	11B	11C	11D	11E	11F
120	121	122	123	124	125	126	127	128	129	12A	12B	12C	12D	12E	12F
130	131	132	133	134	135	136	137	138	139	13A	13B	13C	13D	13E	13F
140	141	142	143	144	145	146	147	148	149	14A	14B	14C	14D	14E	14F
150	151	152	153	154	155	156	157	158	159	15A	15B	15C	15D	15E	15F
160	161	162	163	164	165	166	167	168	169	16A	16B	16C	16D	16E	16F
170	171	172	173	174	175	176	177	178	179	17A	17B	17C	17D	17E	17F
180	181	182	183	184	185	186	187	188	189	18A	18B	18C	18D	18E	18F
190	191	192	193	194	195	196	197	198	199	19A	19B	19C	19D	19E	19F
1A0	1A1	1A2	1A3	1A4	1A5	1A6	1A7	1A8	1A9	1AA	1AB	1AC	1AD	1AE	1AF
1B0	1B1	1B2	1B3	1B4	1B5	1B6	1B7	1B8	1B9	1BA	1BB	1BC	1BD	1BE	1BF
1C0	1C1	1C2	1C3	1C4	1C5	1C6	1C7	1C8	1C9	1CA	1CB	1CC	1CD	1CE	1CF
1D0	1D1	1D2	1D3	1D4	1D5	1D6	1D7	1D8	1D9	1DA	1DB	1DC	1DD	1DE	1DF
1E0	1E1	1E2	1E3	1E4	1E5	1E6	1E7	1E8	1E9	1EA	1EB	1EC	1ED	1EE	1EF
1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9	1FA	1FB	1FC	1FD	1FE	1FF

- Latin script
- Non-Latin European scripts
- African scripts
- Middle Eastern and Southwest Asian scripts
- South and Central Asian scripts
- Southeast Asian scripts
- East Asian scripts
- Indonesian and Oceanic scripts
- American scripts
- Cuneiform
- Hieroglyphs
- Notational systems
- Symbols
- Unallocated code points

As of Unicode 15.0

# Таблицы ASCII и Unicode

```
= String::from("السلام عليكم");  
= String::from("Dobrý den");  
= String::from("Hello");  
= String::from("Dobryj den");  
= String::from("नमस्ते");  
= String::from("こんにちは");  
= String::from("안녕하세요");  
= String::from("你好");  
= String::from("Olá");  
= String::from("Здравствуй");  
= String::from("Hola");
```

If we look at the Hindi word “नमस्ते” written in the Devanagari script, it is stored as a vector of `u8` values that looks like this:

```
[224, 164, 168, 224, 164, 174, 224, 174,  
224, 165, 135]
```

That's 18 bytes and is how computers ultimately store this data. If we look at them as Unicode scalar values, which are what Rust's `char` type is, those bytes look like this:

```
['न', 'म', 'स्', 'ते']
```

# Пользовательские типы

```
6 typedef float coord_t;
7 //typedef double coord_t;
8 //typedef int coord_t;
9
10
11 // Подсчёт расстояния между двумя точками
12 float distance(
13     coord_t x0, coord_t y0, coord_t z0,
14     coord_t x1, coord_t y1, coord_t z1)
15 {
16     float dx = x1 - x0;
17     float dy = y1 - y0;
18     float dz = z1 - z0;
19     return sqrt(dx*dx + dy*dy + dz*dz);
20 }
21
22
23 int main()
24 {
25     coord_t x0, y0, z0;
26     coord_t x1, y1, z1;
27
28     x0 = y0 = z0 = 0;
29     x1 = y1 = z1 = 1;
30
31     coord_t d = distance(x0, y0, z0, x1, y1, z1);
32 }
```

# Структуры

```
6 struct point_t
7 {
8     float x;
9     float y;
10    float z;
11 };
12
13
14 // Подсчёт расстояния между двумя точками
15 float distance(struct point_t p0, struct point_t p1)
16 {
17     float dx = p1.x - p0.x;
18     float dy = p1.y - p0.y;
19     float dz = p1.z - p0.z;
20     return sqrt(dx*dx + dy*dy + dz*dz);
21 }
22
23
24 int main()
25 {
26     struct point_t p0;
27     p0.x = p0.y = p0.z = 0;
28
29     struct point_t p1 = {1, 1, 1};
30
31     struct point_t res = {.x = 0, .y = 0};
32     res = distance(p0, p1);
33 }
34
```

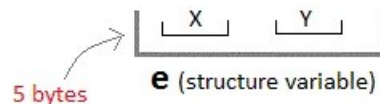
```
5 struct time_point
6 {
7     int64_t seconds;
8     int64_t microseconds;
9 };
10
11
12 typedef struct gps_point
13 {
14     struct time_point time;
15     float lat;
16     float lon;
17     int alt;
18 } gps_point;
19
20
21 // Так тоже можно
22 // typedef struct gps_point gps_point;
23
24
25 int main()
26 {
27     // struct нужно писать
28     struct time_point tp;
29     // struct можно не писать
30     gps_point gp;
31 }
```

# Объединения

```
197
198 //! Событие завершения передачи пакета
199 typedef struct sx1280_event_params_tx_done_t
200 {
201     //! Успешно или нет согласно статусу
202     bool succeed;
203     //! Если случился таймаут, то какой именно
204     sx1280_event_timeout_t timeout;
205 } sx1280_event_params_tx_done_t;
206
207
208 typedef struct sx1280_event_params_rx_done_t
209 {
210     //! Случился или нет
211     bool succeed;
212     //! Если случился таймаут, то какой именно
213     sx1280_event_timeout_t timeout;
214     //! Нормально ли у этого пакета все с CRC (только для лоры)
215     bool lora_crc_good;
216 } sx1280_event_params_rx_done_t;
217
218
219 typedef struct sx1280_event_t
220 {
221     sx1280_event_class_t evt_class;
222     union event_params_t
223     {
224         sx1280_event_params_tx_done_t tx_done;
225         sx1280_event_params_rx_done_t rx_done;
226     } evt_params;
227 } sx1280_event_t;
228
```

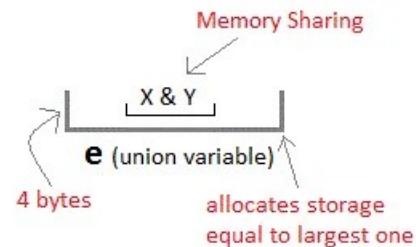
## Structure

```
struct Emp
{
    char X;    // size 1 byte
    float Y;   // size 4 byte
} e;
```



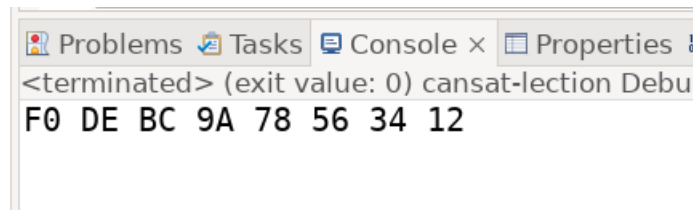
## Unions

```
union Emp
{
    char X;
    float Y;
} e;
```



# Объединения

```
5
6 typedef union caster
7 {
8     int64_t as_integer;
9     uint8_t as_bytes[8];
10 } caster_t;
11
12
13 int main()
14 {
15     union caster c; // можно было caster_t c;
16     c.as_integer = 0x0123456789ABCDEF0;
17
18     for (int i = 0; i < sizeof(c.as_bytes); i++)
19         printf("%02X ", c.as_bytes[i]);
20
21     printf("\n");
22     return 0;
23 }
24
```





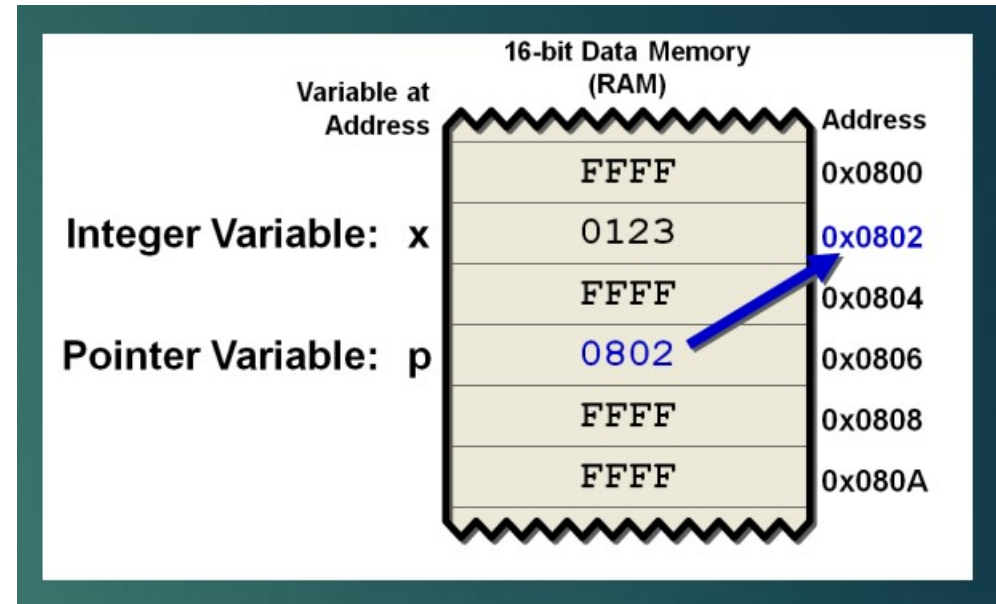
# Модель памяти языка Си

```
5
6 int summ(int arg0, int arg1)
7 {
8     int rv = arg0 + arg1;
9     return rv;
10 }
11
12
13 int main()
14 {
15     int left = 10, right = 42;
16     {
17         int in_block0 = 104;
18         float in_block1 = 52;
19     }
20
21     int res = summ(left, right);
22 }
23
```



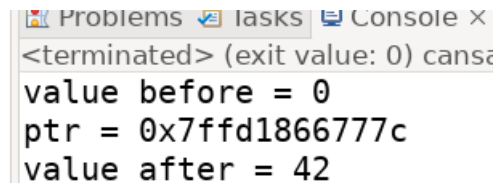
# УКАЗАТЕЛИ

```
12
13 int main()
14 {
15     // Контейнер для целого числа
16     int value = 0;
17
18     // Указатель на целое число
19     int *ptr = &value;
20 }
21
```



# УКАЗАТЕЛИ

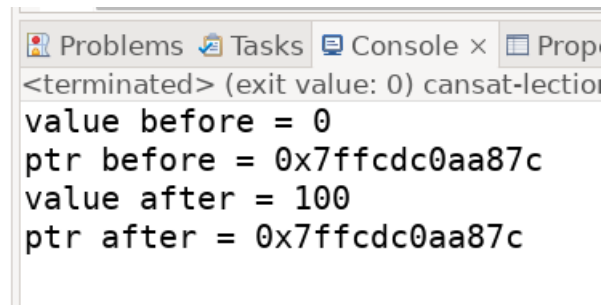
```
13 int main()  
14 {  
15     int value = 0;  
16     printf("value before = %d\n", value);  
17  
18     int *ptr = &value; // & - взятие адреса  
19     printf("ptr = %p\n", ptr);  
20  
21     *ptr = 42; // * - разыменование. Работа по адресу  
22     printf("value after = %d\n", value);  
23 }  
24
```



The screenshot shows a console window with the following output:  
<terminated> (exit value: 0) consa  
value before = 0  
ptr = 0x7ffd1866777c  
value after = 42

# Финт с аргументами функций

```
5
6 void return_by_arg(int *arg)
7 {
8     // Имеет эффект на внешний мир
9     *arg = 100;
10
11     // По-прежнему мучаем локальную копию
12     arg = arg + 10000;
13 }
14
15
16 int main()
17 {
18     int value = 0;
19     int *value_ptr = &value;
20     printf("value before = %d\n", value);
21     printf("ptr before = %p\n", value_ptr);
22
23     // Указатель передается по значению
24     // но его значение это адрес value.
25     // И мы можем влиять на value как захотим
26     return_by_arg(value_ptr); // Можно было сразу &value давать
27
28     printf("value after = %d\n", value);
29     printf("ptr after = %p\n", value_ptr);
30 }
31
```

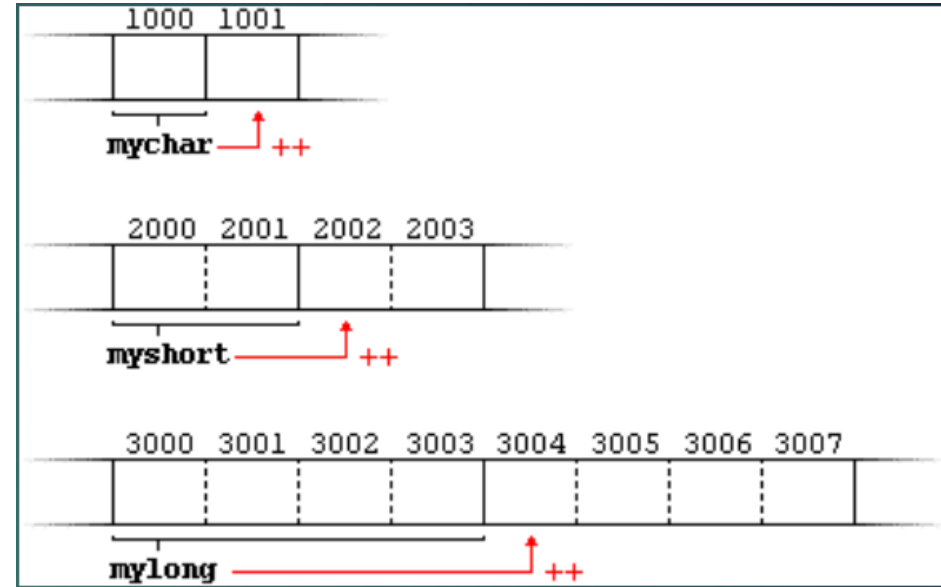


```
Problems Tasks Console x Prop
<terminated> (exit value: 0) cansat-lectio
value before = 0
ptr before = 0x7ffcdc0aa87c
value after = 100
ptr after = 0x7ffcdc0aa87c
```

# Указательная арифметика

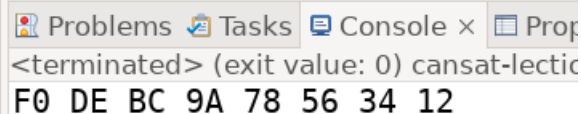
```
6 int main()
7 {
8     char *cptr = 0;
9     printf("ptr = %p; ptr+1 = %p\n", cptr, cptr+1);
10
11     int *iptr = 0;
12     printf("ptr = %p; ptr+1 = %p\n", iptr, iptr+1);
13
14     int64_t *lptr = 0x100;
15     printf("ptr = %p; ptr+1 = %p\n", lptr, lptr+1);
16 }
17
```

Problems Tasks Console x Prop  
<terminated> (exit value: 0) cansat-lectio  
ptr = (nil); ptr+1 = 0x1  
ptr = (nil); ptr+1 = 0x4  
ptr = 0x100; ptr+1 = 0x108



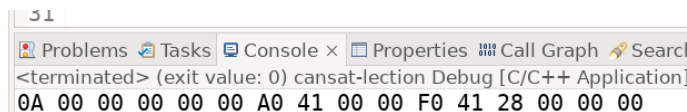
# Преобразования типов указателей

```
5
6= int main()
7 {
8     int64_t large_int = 0x123456789ABCDEF0;
9     uint8_t* bytes_ptr = (uint8_t*)&large_int;
10
11     for (int i = 0; i < 8; i++)
12     {
13         uint8_t value = *(bytes_ptr + i);
14         printf("%02X", value);
15     }
16 }
17
```



Problems Tasks Console x Prop  
<terminated> (exit value: 0) cansat-lection Debug [C/C++ Application]  
F0 DE BC 9A 78 56 34 12

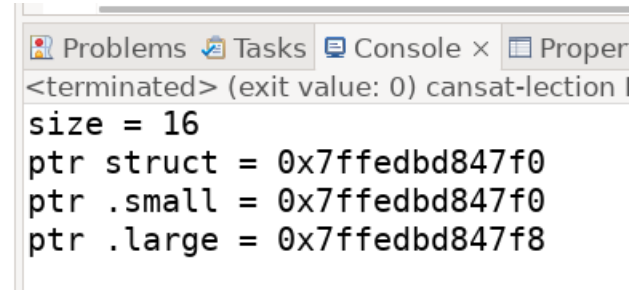
```
5
6= struct gps_point
7 {
8     int time;
9     float lat;
10    float lon;
11    int alt;
12 };
13
14
15= int main()
16 {
17     struct gps_point point = {
18         .time = 10,
19         .lat = 20,
20         .lon = 30,
21         .alt = 40
22     };
23     uint8_t* bytes_ptr = (uint8_t*)&point;
24
25     for (int i = 0; i < sizeof(point); i++)
26     {
27         uint8_t value = *(bytes_ptr + i);
28         printf("%02X", value);
29     }
30 }
31
```



Problems Tasks Console x Properties Call Graph Search  
<terminated> (exit value: 0) cansat-lection Debug [C/C++ Application]  
0A 00 00 00 00 00 A0 41 00 00 F0 41 28 00 00 00

# Выравнивание

```
5
6 typedef struct aligned
7 {
8     char small;
9     int64_t large;
10 } aligned;
11
12
13 int main()
14 {
15     aligned al = {0};
16
17     printf("size = %d\n", sizeof(al));
18     printf("ptr struct = %p\n", &al);
19     printf("ptr .small = %p\n", &al.small);
20     printf("ptr .large = %p\n", &al.large);
21 }
22
```



The screenshot shows a debugger's console window with tabs for Problems, Tasks, Console, and Properties. The console output displays the results of the program's execution, including the size of the struct and the memory addresses of the struct and its members.

```
<terminated> (exit value: 0) cansat-lection I
size = 16
ptr struct = 0x7ffedbd847f0
ptr .small = 0x7ffedbd847f0
ptr .large = 0x7ffedbd847f8
```

# Выравнивание

```
5
6 #pragma pack(push, 1)
7
8 typedef struct aligned
9 {
10     >> char small;
11     >> int64_t large;
12 } aligned;
13
14 #pragma pack(pop)
15
16
17 int main()
18 {
19     >> aligned al = {0};
20
21     >> printf("size = %d\n", sizeof(al));
22     >> printf("ptr struct = %p\n", &al);
23     >> printf("ptr .small = %p\n", &al.small);
24     >> printf("ptr .large = %p\n", &al.large);
25 }
26
```

```
<terminated> (exit value: 0) cansat-lection D
size = 9
ptr struct = 0x7ffd44777f1f
ptr .small = 0x7ffd44777f1f
ptr .large = 0x7ffd44777f20
```

```
10
17 int main()
18 {
19     >> aligned al = {0};
20     >> aligned *ptr = &al;
21
22     >> // Доступ к полям структур/юнионов по указателю
23     >> // через ->
24     >> // Вот так еще можно
25     >> ptr->large = 10;
26
27     >> // Вот так - лучше не стоит
28     >> int64_t *free_ptr = &ptr->large;
29     >> *free_ptr = 10;
30 }
31
```



# Указатели и массивы

```
5
6 // Если нужно передать в функцию массив
7 // Лучше использовать указатель и размер
8 void func_ptr(int *pointer, int size);
9 // Так тоже можно
10 void func_arr(int array[], int size);
11 // Так тоже можно, но никто не гарантирует,
12 // что вам дадут именно такой массив
13 void func_arr2(int array[100500]);
14 // Так тоже можно
15 void func_arr3(int array[10][20]);
16
17
18 int main()
19 {
20     int array[100] = {0};
21     int *point = array; // Так можно! Без явных кастов
22
23     // Вот это - одинаковые записи
24     *(array + 10) = 100;
25     point[10] = 100;
26 }
27
```

# Значение NULL

```
5
6 void func_with_opt_arg(int arg0, int *opt_arg)
7 {
8     // ...
9
10    if (opt_arg != NULL)
11    {
12        // Что-то делаем с опциональным аргументом
13    }
14 }
15
16
17 int main()
18 {
19     int arg0 = 0;
20     func_with_opt_arg(arg0, NULL);
21
22     int arg1 = 10;
23     func_with_opt_arg(arg0, arg1);
24 }
25 |
```

NULL — специальный макрос из `<stdlib.h>` который можно писать как значение указателям, которые указывают в никуда.

99.999%, что `NULL == 0`, но это не точно

# Указатель на void

```
5
6 // Функция принимает совершенно любой указатель ptr
7 // И копирует по нему data_size байт в радиосистему
8 void func_with_any_ptr(void *ptr, int data_size)
9 {
10     uint8_t *u8ptr = (uint8_t *)ptr;
11     for (int i = 0; i < data_size; i++)
12     {
13         // ...
14     }
15 }
16
17
18 int main()
19 {
20     int data[100] = {0};
21     func_with_any_ptr(data, sizeof(data)); // явных кастов не нужно
22
23     void *ptr = 0;
24     *ptr = 10; // Это ошибка, так нельзя
25     ptr += 100; // Так тоже нельзя
26
27     int32_t var;
28     ptr = &var; // Зато так можно и запросто
29 }
30
```

# Указатели и const

Кстати это UB!



```
6 int main()
7 {
8     int var;
9     int another_var;
10
11     // Вообще без const
12     int *x0 = &var;
13     *x0 = 10; // Можно менять как "цель"
14     x0 = &another_var; // так и сам указатель
15
16     // const слева от *
17     const int *x1 = &var;
18     *x1 = 10; // Нельзя менять цель
19     x1 = &another_var; // указатель менять можно
20
21     // const справа от *
22     int *const x2 = &var;
23     *x2 = 10; // Можно менять цель
24     x2 = &another_var; // Нельзя менять указатель
25
26     // const с обеих сторон
27     const int *const x3 = &var;
28     *x3 = 10; // Нельзя
29     x3 = &another_var; // Ничего нельзя
30 }
```

```
main() {
    const int a = 50;
    int* pa = &a;
    *pa = 60;
}
```



*Reality can be whatever I want.*

# Снова UB

```
0
7 int * f(void)
8 {
9     int value;
10    int * ptr = &value;
11    return ptr;
12 }
13
14 int main()
15 {
16     int * pointer = f();
17     *pointer = 10; // Ой ой
18 }
19
```

# Указатель на указатель

```
6 int main()
7 {
8     int var, another_var;
9     int *var_ptr = &var;
10    // var_ptr лежит в памяти... А значит у него есть адрес...
11    int **ptr_ptr = &var_ptr; // А вот и он.
12
13    *ptr_ptr = &another_var; // Меняем указатель
14    **ptr_ptr = 10; // Меняем на что указываем
15 }
```

# Указатель на указатель на указатель

[illegible]

# Работа с кучей

Запросить память:

```
void * malloc(size_t size);
```

Отдать память:

```
void free(void * ptr);
```

```
1 int main()
2 {
3     while(1)
4     {
5         uint8_t * buffer = (uint8_t *) malloc(10*1024*1024);
6         // ...
7         // ...
8         // ...
9         // ...
10        // ...
11        free(buffer);
12    }
13 }
```

