Программирование на языке C

ЛЕКЦИЯ 2

Краткое содержание предыдущей серии

```
// подключение заголовочного файла
#include <stdio.h>

// Определение макроса
#define MATH_PI 3.14159265358979323 8

// Определение макроса с параметром
#define RAD_TO_DEG(arg) arg*180.0/MATH_PI
#define DEG_TO_RAD(arg) arg*MATH_PI/180.0


// функция с двумя аргументами и с возвращаемым значением типа int
int summ(int arg1, int arg2) {
    return arg1 + arg2;
}


// функция без аргументов и без вовзращаемого значения
void printDoubledValue(int value) {
    int doubledValue = value*2;
    printf("Печатаю удвоенное значение: ", doubledValue);
    // return не обязателен
}

```

Краткое содержание предыдущей серии

```
26
27 // Функция main, с которой начинается программа
28 int main(){
29     int someVariable; // определение переменной.
30     int someVariable, someAnotherVariable; // определение нескольких переменных
31
32     someVariable = 42; // Присвоение значения переменной
33     someVariable = someAnotherVariable = 42; // присвоение значения переменной цепочкой
34
35     // определение и инициализация переменной
36     int yetAnotherVariable = 42 + 42 - 10; // математические операции
37
38     // вызов функции
39     int summedValue = summ(someVariable, someAnotherVariable);
40     summ(10, 15); // вовзращаемое значение не обязательно забирать
41     printDoubledValue(12);
42
43     // Использование макросов
44     double circleRadius = 10.0;
45     double circleArea = MATH_PI * circleRadius*circleRadius;
46     // перед компиляцией заменится на
47     // double circleArea = 3.14159265358979323 * circleRadius*circleRadius
48
49
50     return 0;
51 }
52
```

Краткое содержание предыдущей серии

```
1
2 #include <stdbool.h>
3 #include <stdio.h>
4
5 int main() {
6     int value = 10;
7
8     // Условный оператор
9     if (value > 10) {
10        printf("value больше 10ти");
11    } else {
12        printf("value не больше 10ти");
13    }
14
15    return 0;
16 }
17
18
```

Так же мы рассмотрели:

- Азы работы в эклипсе;
- Базовые типы данных языка С;
- Преобразования типов и переполнения;
- Операторы (+, -, >>, ~, !);

Типы литералов и целочисленное деление

```
int main() {

    {
        // если аргументы целочисленные - деление тоже целочисленное
        int x = 5;
        int y = x/2; // y == 2

        // остаток от деления можно получить при помощи оператора %
        int z = 5 % 2; // z == 1
    }

    {
        float x = 5;
        float y = x/2;      // y == 2.5
        y = 5/2;            // y == 2.0 (потому что 5 и 2 всеравно целочисленные)
        y = (float)5/2;     // y == 2.5
        y = 5.0f/2;         // y == 2.5
    }

    return 0;
}
```

```
int main() {

    {
        double a = 5.0;
        float  b = 5.0f;
        int    c = 5;
        long   d = 5L;
    }

    // ананлогично
    {
        double a = (double)5;
        float  b = (float)5;
        int    c = (int)5;
        long   d = (long)5;
    }

    return 0;
}
```

Константы

Константа ничем не отличается от обычной переменной, кроме того, что она должна быть инициализированна при определении и ей нельзя присводить значение.

Чтобы сделать переменную константой нужно использвать модификатор **const**

```
2  int main() {
3
4      const int x = 10;
5      const int y; // ошибка, константа должна быть инициализированна
6
7      x = 20;  // ошибка, нельзя изменять константу после инициализации
8      x += 20; // тоже ошибка
9
10     int a = x; // Можно читать значение константы
11     return 0;
12 }
13
```

Особенности передачи аргументов в функции

```c
#include <stdio.h>


void argModificationTest(int arg) {
    printf("Аргумент в функции %d", arg);
    arg = arg + 10;
    printf("Аргумент в функции после изменения %d", arg);
}


int main() {
    int x = 0;
    printf("Аргумент в main %d", x);
    argModificationTest(x);
    printf("Аргумент в main после вызова функции: %d", x);

    return 0;
}

```

Вывод:
Аргумент в main 0
Аргумент в функции 0
Аргумент в функции после изменения 10
Аргумент в main после вызова функции: 0

Область видимости переменных

- Переменные видны от их определения, до закрывающей скобочки блока { }, в котором они определены

```
int main() {

    printf("y = %d", y); // Ошибка: y еще не определена
    int y;

    {
        int x = 20;
        printf("y = %d", y); // все в порядке
    }

    printf("x = %d", x); // ошибка - x здесь не видна
    printf("y = %d", y); // все в порядке

    return 0;
}
```

- Все правила области видимости касаются скобочек { } управляющих конструкций, таких как
  if ( ) { … } else { … }

Область видимости переменных

- Имена переменных можно перекрывать на более глубоких вложенностях

```
int main() {

    int x;
    x = 0;

    {
        int x;
        x = 20;

        printf("x = %d", x);
    }

    printf("x = %d", x);

    return 0;
}
```

- Какой будет вывод?

Область видимости переменных

- Переменные определенные внутри функций или на более глубоких уровнях вложения блоков { } называеются **локальными**.
- Переменные опередеденные на самом первом уровне вложеннсти (вообще вне { } называются **глобальными.** Глобальные переменные видны от их определения, до конца файла.

```
// опредееление глобальной переменной
int globalX = 10;


void printGlobalX() {
    printf("globalX в функции = %d", globalX);
}

int main() {

    printf("globalX в main = %d", globalX);

    globalX += 10;
    printf("globalX в main = %d", globalX);

    globalX += 10;
    printGlobalX();

    return 0;
}
```

globalX в main = 10
globalX в main = 20
globalX в функции = 30

Область видимости функций

- Подобно глобальным переменным функции видны от их определения до конца файла

```
#include <stdio.h>

void f1() {
    printf("Вызвана функция f1");
}


void f2() {
    f1();
    printf("Вызвана функция f2");
}


void f3() {
    f2();
    printf("Вызвана функция f3");
}


int main() {
    f3();

    return 0;
}
```

Массивы и циклы

- Представим, что нам нужно сделать несколько измерений температуры и посчитать её среднее значение. Можно сделать так:

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

uint8_t getTemperature() {
    uint8_t tempValue;
    // ...
    // операции с датчиком
    // ...
    return tempValue;
}

int main() {
    uint8_t temp0 = getTemperature();
    uint8_t temp1 = getTemperature();
    uint8_t temp2 = getTemperature();
    uint8_t temp3 = getTemperature();
    uint8_t temp4 = getTemperature();

    uint8_t averageTemp = (temp0 + temp1 + temp2 + temp3 + temp4);
    averageTemp /= 5;

    printf("Среднее значение температуры: %d", averageTemp);
    return 0;
}
```

Массивы и циклы

- Гораздо лучше будет использовать массив и цикл

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

#define TEMP_ARRAY_SIZE 500

uint8_t getTemperature() {
    uint8_t tempValue;
    // ...
    // операции с датчиком
    // ...
    return tempValue;
}

int main() {
    uint8_t tempValues[TEMP_ARRAY_SIZE];

    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        tempValues[i] = getTemperature();
    }

    uint8_t averageTemp = 0;
    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        averageTemp += tempValues[i];
    }

    averageTemp /= TEMP_ARRAY_SIZE;
    printf("Среднее значение температуры: %d", averageTemp);
    return 0;
}
```

Массив – это набор переменных Одного и того же типа. Объявляется как Переменная, но в конце указывается размер
Массива в скобочках [ ].

Можно использовать модификаторы еременных (например **const**).

Для доступа к элементу массива нужно написать его имя и затем в скобочках [ ] номер элемента. Нумерация с нуля.

Массивы и циклы

Многомерные массивы и инициализация.

```
int main() {

    // многомерные массивы
    int twoDimensionalArray[5][10];
    int threeDimensionalArray[5][10][15];

    // доступ к элементу
    printf("arrayValue: %d", twoDimensionalArray[0][0]);
    printf("arrayValue: %d", threeDimensionalArray[0][0][0]);

    // инцилазиация массивов
    int array1[10]; // в элементах массива что угодно;
    int array2[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // все элементы массива указаны
    int array3[10] = {1, 2, 3}; // первые три элемента массива 1,2,3 - остальные ноль

    // инцилазиация многомерных массивов
    int array4[5][5] = {
            {0, 1, 2, 3, 4},
            {0, 1, 2, 3},
            {}
    };

    return 0;
}
```

Массивы нельзя использовать как аргументы функций

Массивы и циклы

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

#define TEMP_ARRAY_SIZE 500

uint8_t getTemperature() {
    uint8_t tempValue;
    // ...
    // операции с датчиком
    // ...
    return tempValue;
}

int main() {
    uint8_t tempValues[TEMP_ARRAY_SIZE];

    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        tempValues[i] = getTemperature();
    }

    uint8_t averageTemp = 0;
    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        averageTemp += tempValues[i];
    }

    averageTemp /= TEMP_ARRAY_SIZE;
    printf("Среднее значение температуры: %d", averageTemp);
    return 0;
}
```

Циклов в Си несколько: **for**, **while** и **do-while**
Они аналогичны паскалевским for, while и repeat-untill.

for(
то, что происходит перед первой итерацией
;
условие == true
;
то, что происходит после каждой итерации
)
{
Тело цикла
}

Массивы и циклы

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>


bool isLegDeployed() {
    bool status;
    //... работа с датчиками
    return status;
}

void stepLeg(int stepsCount) {
    // ... работа с двигателем ...
}

int main() {

    while(!isLegDeployed()){
        stepLeg(1);
    }

    // do while почти полностью аналогичен
    // но отрабатывает как минимум один раз
    do {
        stepLeg(i);
    } while(isLegDeployed());

    return 0;
}
```

```
while (условие == true) {
   тело цикла
}


do {
    тело цикла
} while(условие == true);
```

Массивы и циклы

```
int main() {

    bool someEventFlag;

    // бесконечный цикл
    while(1) {
        if (someEventFlag){
            break;
        }
    }


    while(1){
        if (someEventFlag)
            continue;

        // действия, которые не произойдут, если continue сработал
    }

    return 0;
}
```

Специальные операции в циклах

break == безусловный выход из цикла
continue == переход к следующей итерации

Массивы и циклы

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

#define TEMP_ARRAY_SIZE 500

uint8_t getTemperature() {
    uint8_t tempValue;
    // ...
    // операции с датчиком
    // ...
    return tempValue;
}

int main() {
    uint8_t tempValues[TEMP_ARRAY_SIZE];

    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        tempValues[i] = getTemperature();
    }

    uint8_t averageTemp = 0;
    for (int i = 0; i < TEMP_ARRAY_SIZE; i++) {
        averageTemp += tempValues[i];
    }

    averageTemp /= TEMP_ARRAY_SIZE;
    printf("Среднее значение температуры: %d", averageTemp);
    return 0;
}
```

В этой программе есть логическая (не синтаксическая) ошибка. Найдите её

Структуры

```
int main() {

    struct {
        int x;
        double y;
        float z;
        int a[100];
    } value;

    value.x = 0;
    value.y = 1.0;
    value.z = 2.0f;
    value.a[0] = 10;
    value.a[1] = 20;

    return 0;
}
```

Простая стукрутура

```
int main() {

    struct {
        int x;
        struct {
            int xx;
            double yy;
            int zz;
        } innerStruct;
    } value;

    value.x = 0;
    value.innerStruct.xx = 10;
    value.innerStruct.yy = 10;
    // и так далее

    return 0;
}
```

Вложенная структура

```
int main() {
    struct {
        int x;
        double y;
        float z;
        int a[100];
    } value = {1, 2.0, 3.0f, {10, 20, 30} };


    return 0;
}
```

Инициализация структур

Со структурами как с любыми типами можно использовать модификаторы, например **const**

Структуры

```c
//пример простого typedef
typedef int temp_t;

//пример typedef структуры
typedef struct {
    int x, y, z;
} my_struct_t;


//функция, позврвщющая структуру
my_struct_t getStruct() {
    my_struct_t retval;
    retval.x = 10;
    return retval;
}

//фукция, принимающая стукьтуру как аргумент
void useStruct(my_struct_t value){
    printf("struct value x = %d", value.x);
}


int main() {
    my_struct_t myStruct = getStruct();
    useStruct(myStruct);

    return 0;
}
```

Использование структуры и typedef