Программирование микроконтроллеров STM32

Лекция №3
Организация циклов, условные операторы, написание функций

Разработчик доц. Зубков О.В.

План лекции

- Операторы сравнения и ветвления
- Организация циклов
- Описание, создание и вызов функций
- Примеры программной реализации
- Структура главной программы

Условный оператор

Оператор if ("если") позволяет организовать ветвление в программе. Он имеет две формы: оператор "если" и оператор "если... иначе".

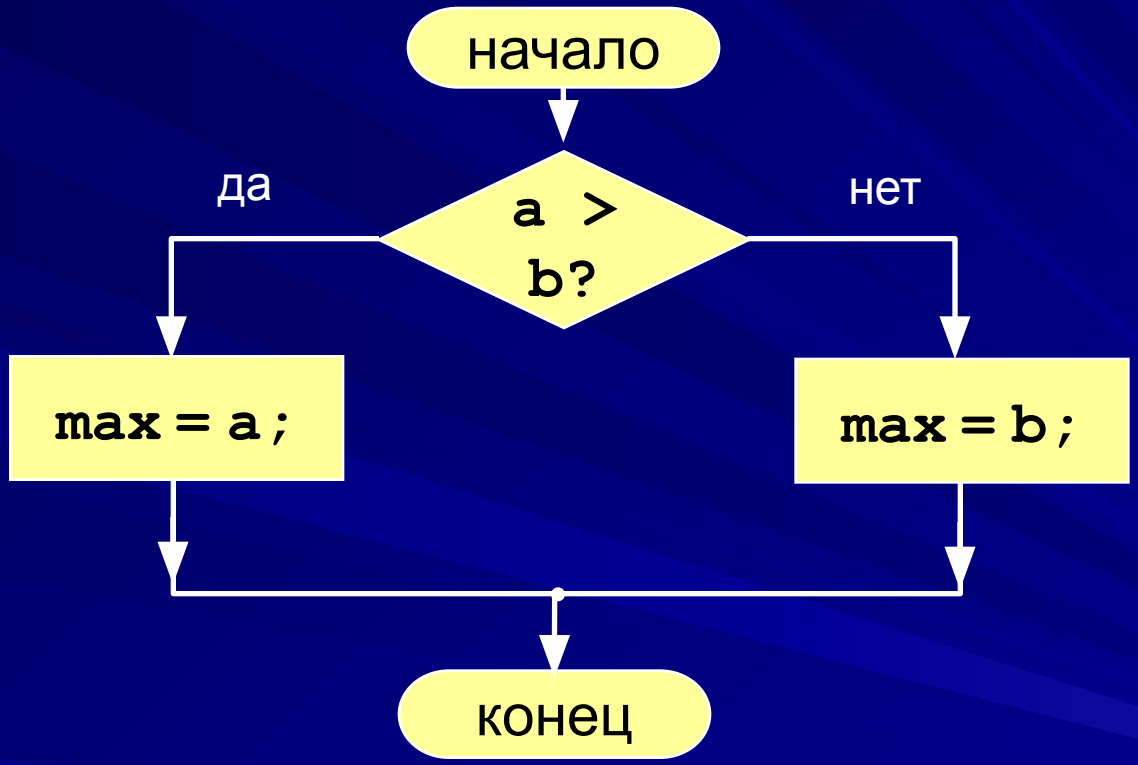
Условный оператор “если”

```
if ( условие )
   {
        // что делать, если условие верно
   }
```

Условный оператор “если…иначе”

```
if ( условие )
   {  // что делать, если условие верно
   }
else
   {  // что делать, если условие неверно
   }
```

Пример нахождения максимального из двух чисел



```
Int32_t a, b, max;

if (a > b) {
      max=a;
     }
   else {
      max=b;
     }
```

Выбор из нескольких возможностей if-else if

Несколько условных операторов типа "если... иначе" можно записывать последовательно, то есть действие после else может снова представлять собой условный оператор.

В результате реализуется выбор из нескольких возможностей.

Синтаксис:

```
if   (выражение 1)
        оператор 1
else
      if (выражение 2)
            оператор 2
      else
            оператор 3
```

Пример:

```
if   ((a>5)&&(b<120)
        y=a*5+b;
else
       if (a>20)
             y=a*4+b*2
       else
             y=b*8+3;
```

Оператор множественного выбора **switch**

Синтаксис

```
switch (выражение)
        {
          case константа1: оператор; break;
          case константа2: оператор; break;
          case константа3: оператор; break;
           ...
          default: оператор
        }
```

1. сначала вычисляется значение выражения в заголовке *switch*;
2. затем осуществляется переход на метку "*case константа L*:", где *константа L* совпадает с вычисленным значением *выражения* в заголовке;
3. если такого значения нет среди меток внутри тела *switch*, то

3.1 если есть метка "*default*:", то осуществляется переход на нее;

3.2 если метка "*default*:" отсутствует, то ничего не происходит.

```
int M, D;
switch ( M ) {
   case 2:  D = 28; break;
   case 4: case 6: case 9: case 11:
            D = 30; break;
   case 1: case 3: case 5: case 7:
   case 8: case 10: case 12:
            D = 31; break;
   default: D = -1;
   }
```

выйти из **switch**

ни один вариант не подошел

Цикл

Цикл – это многократное выполнение одинаковой последовательности действий.

- цикл с **известным** числом шагов
- цикл с **неизвестным** числом шагов (цикл с условием)

Цикл for

```
for (выражение;
     условие продолжения цикла;
     изменение на каждом шаге)
 {
 // тело цикла
 }
```

выражение - описывает инициализацию цикла;
условие продолжения цикла - проверяет условие завершения цикла, если оно истинно то выполняется тело цикла;
изменение на каждом щагу – итератор, который вычисляется после каждой итерации.
Цикл повторяется до тех пор, пока *условие продолжения цикла* не станет ошибочным.

Примеры цикла for

```
for (a=2; a<b; a+=2) { ... }
```

```
for (a=2, b=4; a<b; a+=2) { ... }
```

```
for (a=1; c<d; x++) { ... }
```

```
for (; c<d; x++) { ... }
```

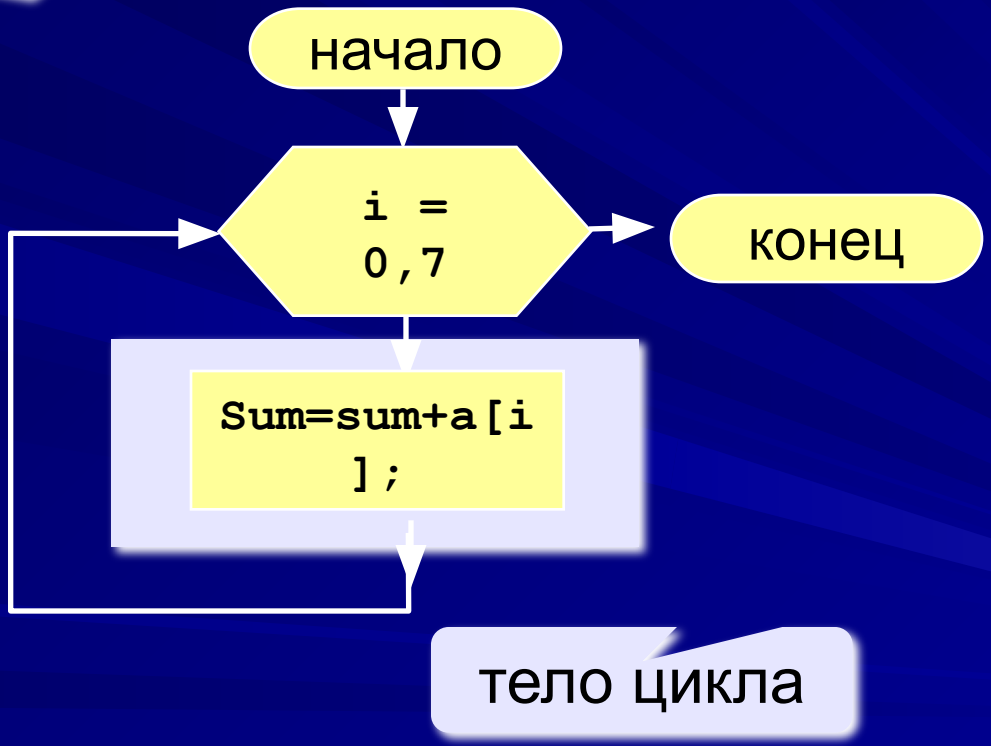
```
for (; c<d; ) { ... }
```

```
for (;; ) { ... }
```

Бесконечный цикл

Пример программы нахождения суммы элементов массива

блок «цикл»



```
main()
{
int32_t i, a[8];
for (i=0; i<=7; i++)
    {
    sum=sum+a[i];
    }
}
```

начальное значение

конечное значение

ЦИКЛ

переменная цикла

изменение на каждом шаге: `i=i+1`

начало цикла

тело цикла

конец цикла

цикл работает, пока это условие верно

Цикл с условием

```
while ( условие )
   {
   // тело цикла
   }
```

Особенности:

- можно использовать сложные условия:

```
while (a<b && b<c ) { ... }
```

- если в теле цикла только один оператор, скобки { } можно не писать:

```
while ( a < b ) a ++;
```

Особенности:

- условие пересчитывается **каждый раз** при входе в цикл
- если условие на входе в цикл ложно, цикл не выполняется ни разу

```
a = 4; b = 6;
while ( a > b) a=a-b;
```

- если условие никогда не станет ложным, программа **зацикливается**

```
a = 4; b = 6;
while ( a<b) d=a+b;
```

- **Бе ательным элементом программы микроконтроллера**

```
while ( 1) {…};
```

Цикл **do-while**

do
оператор
while (выражение);

Если *выражение* истинное, то оператор *выполняется* и снова вычисляется значение *выражения*. Это повторяется, пока *выражение* не станет ошибочным.
Оператор выполняется не меньше одного раза .
Очень часто после
while (выражение)
забывают ставить точку с запятой.

Пример:

```
void main()
{
    uint8_t x;
    x=1;
    do {
     x++;
          } while(x<=10);
}
```

Выход из цикла **break**

break;

Оператор *break* перерывает выполнение ближайшего вложенного внешнего оператора *switch*, *while*, *do* или *for*. Управления передается следующему оператору. Например в примере выход из бесконечного цикла *"while (true)"* осуществляется с помощью оператора *"break"*.

Пример:

```
main()
{    uint8_t x, v;
      x=0; v=0;
      while (v==0)
          {    if  x==10
                {  break;
                // условие выхода из
                // программы при x=10
                 }
               else { x++; }
          }
}
```

Оператор перехода на метку **goto**

goto (метка);

Оператор перехода *goto* позволяет изменить естественный порядок выполнения программы и осуществить переход на другой участок программы, обозначенный *меткой*. Переход может осуществляться только внутри функции, т.е. оператор *goto* не может ни выйти из функции, ни войти внутрь другой функции.

В качестве *метки* можно использовать любое имя, допустимое в Си (т.е. последовательность букв, цифр и знаков подчеркивания "_", начинающуюся не с цифры).

Метка может стоять до или после оператора *goto*. *Метка* выделяется символом двоеточия ":". Лучше после него сразу ставить точку с запятой ";", помечая таким образом пустой оператор - это общепринятая программистская практика, согласно которой *метки* ставятся между операторами, а не на операторах.

Не следует увлекаться использованием оператора *goto* - это всегда запутывает программу. Большинство программистов считают применение оператора *goto* дурным стилем программирования. Вместо *goto* при необходимости можно использовать операторы выхода из цикла *break* и пропуска итерации цикла *continue*.

```
void main()
{
uint8_t i,j;
  i=1;
  while(i<10)
    {
     j=1;
      while(j<20)
        {
          if (i+j==25)
            { goto metka_1;
      }
          j++;
        }
      i++;
    }
 metka_1: ;
}
```

Единственная ситуация, в которой использование *goto* оправдано, - это выход из нескольких вложенных друг в друга циклов.

Функции

Функция – это вспомогательный алгоритм (подпрограмма), результатом работы которого является некоторое значение.

Примеры:

- вычисление модуля числа
- расчет значений по сложным формулам

Зачем?

- для выполнения одинаковых расчетов в различных местах программы
- для создания общедоступных библиотек функций

Синтаксис написания функции

```
Тип_возвращаемого_значения имя_функции(список параметров)
    {
        //Тело функции
    }
```

В качестве типа возвращаемого значения может использоваться ключевое слово ***void.***
Оно означает, что функция не возвращает никакого значения (такие функции еще называют процедурами).
Функция может не иметь списка параметров. В этом случае вместо списка параметров записывается ключевое слово ***void.***

Функцию вызывают по ее имени с указанием в круглых скобках перечня передаваемых параметров

```
int32_t Max ( int32_t a, int32_t b )
{
  ...
}
```

формальные параметры

```
main()
{
 int a, b, c;
 c = Max ( a, b );
 }
```

фактические параметры

вызов функции

Задача: составить функцию, которая вычисляет наибольшее из двух значений, и привести пример ее использования

Функция:

тип результата

формальные параметры

```
Int32_t Max(int32_t a, int32_t b )
{
  if ( a > b ) return a ;
  else         return b ;
}
```

return - вернуть результат функции

Пример функции без типа

```
int32_t p;
…
void my_func(uint8_t k; int32_t t)
{
   p=(int32_t)k*t+20;
}
…
my_func(5,3); //Вызов функции
```

Назначение библиотек

- Для использования при написании программы имен внутренних регистров вместо их адресов, что удобнее и понятнее.
- Использование разработанных другими программистами программ и функций, для реализации математических функций, управления устройствами (индикаторов, микросхем памяти и др.), реализации протоколов обмена информации и мн. др.

Подключение библиотек

Подключить дополнительную библиотеку можно используя директиву

#include *имя подключаемого файла*

Пример

```
#include "stm32f4xx.h"
```

Библиотека stm32f4xx.h - файл описания процессора

- Библиотека содержит соответствия между именами внутренних регистров процессора и их адресами.
- Библиотека содержит также соответствия между именами битов регистров управления и их номерами в регистре

Тело основной программы

Тело основной программы является функцией с именем main и типом int

В теле основной программы должен находиться бесконечный цикл содержащий повторяющуюся последовательность действий

```
int main(void)
{
}
```

Структура главной программы без операционной системы

#include <файл описания процессора>

#include <вспомогательный файл>//если он необходим

//Описание констант

#define …

//Описание переменных

//Описание функций

Тип_функции имя_функции(тип входных параметров имя входных параметров);

Структура главной программы без операционной системы

```
//Основная программа
int main(void)
{
//Блок начальной инициализации процессора
    while (1)
     {
          //Блок основной программы
     }
}
```

Файл обработчиков прерываний stm32f4xx_it.c

Файл содержит все функции использованных обработчиков прерываний. В файле отсутствует тело главной программы. В остальном структура программы аналогична рассмотренной выше. При использовании переменных из главной программы на них должны быть ссылки из stm32f4xx_it.c с ключевым словом extern

Структура главной программы с операционной системой

#include <файл описания процессора>

#include <вспомогательный файл>//если он необходим

//Описание констант

#define …

//Описание переменных

//Описание функций

Тип_функции имя_функции(тип входных параметров имя входных параметров);

```
int main(void)
{
//Блок начальной инициализации
   процессора
…
//Запуск задачи
defaultTaskHandle =
   osThreadNew(StartDefaultTask, NULL,
   &defaultTask_attributes);
//Передача управления планировщику задач
osKernelStart();
   while (1)
    {
    }
}
```

```
//Задачи операционной системы
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
for(;;)
 {
    osDelay(1);
  }
  /* USER CODE END 5 */
}
```

Время, через которое выполняется задача. Минимальное значение 1 мс.