**AVR Урок 9. Оформление кода. Функции**

**Урок 9**

**Оформление кода. Функции**

Сегодня мы попытаемся немного покрасивее оформить наш код. Главное — это конечно не красота, а то, чтобы код мы в любой момент могли прочитать и понять, причем не только мы.

Для этого существует много способов оформления кода. И один из них — **функции**. Функции в языке C/C++ используются для того, чтобы объединить какой-то участок кода, который особенно может в любой момент повторяться, или который представляет собой какую-то логически-законченную процедуру в отдельный фрагмент и может быть в любой момент вызван из какого-то другого участка кода.

Как пишутся функции — мы в принципе знаем с самого первого занятия, но до сих пор мы использовали только одну функцию. Это была функция **main()**, которая является точкой входа в нашу программу. Всё выполнение кода начинается именно с этой функции. А в дальнейшем мы будем использовать очень много различных функций. Поэтому, если кто-то что-то по функциям не поймёт, то в процессе наших дальнейших занятий он обязательно возместит данный пробел за счёт огромной практики использования функций.

Кроме тела у функции также существуют аргументы, которые мы можем передавать данной функции из фрагмента кода при вызове — это входные аргументы. А также функция может возвращать один аргумент. Как правило по данному аргументу мы будем судить, как именно прошел процесс в вызываемой функции.

Сегодня мы попытаемся написать такую функцию, которая благодаря входному аргументу, будет отображать на индикаторе определённую цифру.

Сегодня я уже не буду рассказывать, как создавать проект, я думаю, это уже всем надоело. Запустим уже созданный заранее из кода прошлого занятия и целиком настроенный проект и начнём писать нашу функцию. Напишем мы её до фунции main(), так как интерпретатор команд «смотрит» код сверху вниз и наша функция будет поэтому уже ему «знакома» в коде функции main(), и будет доступна для вызова в коде (или как говорят «видна»). Можно также писать функции и после того места, откуда они вызываются, но тогда нужно будет уже писать **прототип** функции. С прототипами функций мы познакомимся в более поздних занятиях. А сегодня напишем функцию сразу после объявления глобальных переменных. Глобальные переменные «видны» во всех функциях файла. Вот наша функция — пока без кода, с пустым телом. Дадим ей имя **segchar**. Имя функции должно говорить тому, кто будет читать код о том, чем данная функция будет заниматься. В нашем случае функция будет с помощью сегментов (seg) определённые символы (char). Вот поэтому и такое название

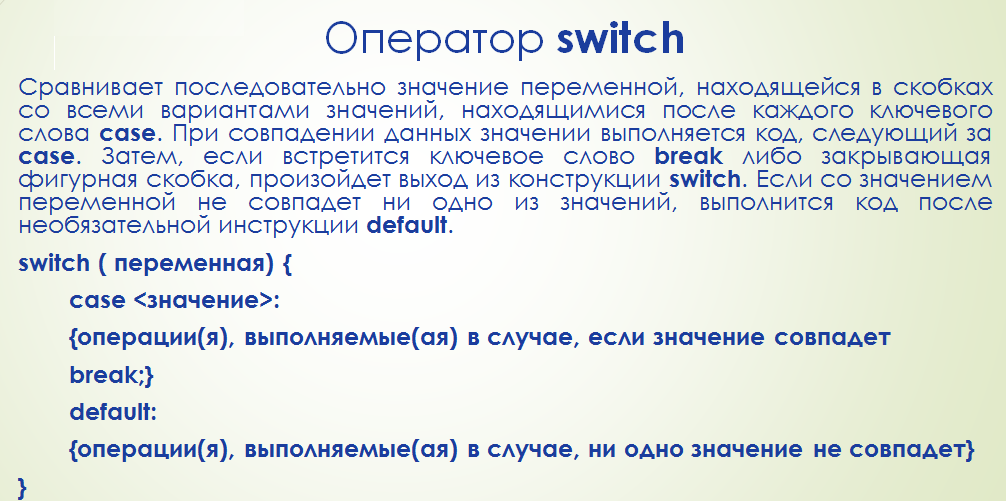
**void segchar (unsigned char seg)**

**{**

**}**

У данной функции будет только один входной аргумент — это целочисленное короткое значение с именем seg. Возвращать функция ничего не будет, о чём свидетельствует значение void перед её именем, то есть мы будем «верить» функции, что она справится с выводом символа на экран, и проверять это не будем.

Также чтобы функция «понимала», какой именно зажигать символ, нам желательно изучить ещё один оператор — это оператор **switch**, который в зависимости от значения аргумента выполняет определённый участок кода. Это непростой оператор. Для этих целей конечно подойдёт и оператор **if**, но когда вариантов слишком много (цифр ведь целых 10), то удобнее всё-таки применить оператор ветвления **switch**. Вот что это за операор



Я думаю, несмотря на сложность данного оператора, мы с ним непременно разберёмся в процессе многочисленного его использования. Мастерство приходит с практикой.

Давайте применим данный оператор в нашей функции

void segchar (unsigned char seg)

{

**switch (seg)**

**{**

**case 1: PORTD = 0b11111001; break;**

**case 2: PORTD = 0b10100100; break;**

**case 3: PORTD = 0b10110000; break;**

**case 4: PORTD = 0b10011001; break;**

**case 5: PORTD = 0b10010010; break;**

**case 6: PORTD = 0b10000010; break;**

**case 7: PORTD = 0b11111000; break;**

**case 8: PORTD = 0b10000000; break;**

**case 9: PORTD = 0b10010000; break;**

**case 0: PORTD = 0b11000000; break;**

**}**

}

Давайте немного разберёмся. Здесь у нас в зависимости от значения переменной **seg** мы будем попадать в определённый участок кода, и будем его выполнять, пока не встретится оператор **break**. Как только он попадётся, то мы из тела оператора switch выйдем, ну и так как за закрывающей тело фигурной скобкой следует следующая скобка, то мы вернёмся из функции в то место, откуда мы данную функцию вызвали. Ну конечно попадать, возвращаться, выходить будем не мы, это так образно говорится. Заниматься этим будет АЛУ. Мы просто представляем себя в роли АЛУ.

Здесь уже в значениях, присваваиваемых порту D, я не стал применять инвертирование, а поменял нули на единицы и наоборот.

Перейдём в функцию **main()** и удалим оттуда ненужные нам комментарии

PORTB = 0b00000001;~~// 1 2 3 4 5 6 7 8~~

while(1)~~// 0b|dp|g|f|e|d|c|b|a|~~

Перед тем, как вызвать нашу функцию в бесконечном цикле, мы организуем цикл, который будет перебирать подряд цифры от 0 до 9. Код скопируем из закомментированного участка бесконечного цикла, переместив его до обработчика кнопки, а комментарий потом уберем

while(1)

  {

**for (i=0;i<10;i++)**

**{**

**segchar(i);**

**\_delay\_ms(500);**

**}**

  if (!(PINB&0b00000001)){

Код кнопки пока не трогаем, кнопку эту мы сегодня ещё вспомним.

Теперь соберём проект и посмотрим в протеусе. Файл протеуса также был скопирован с переименованием, и в свойствах контроллера был добавлен путь к новой прошивке. Запустим проект в протеусе и убедимся в том, что у нас циферки таким же образом работают, как и на прошлом занятии. Тем самым мы добились значительного сокращения кода в главной функции программы.

Но это ещё не всё. у нас без дела простаивает кнопка. Давайте и ей дадим какую-нибудь работу. Здесь уже одной переменной для кнопки не обойдёмся, так как нам нужен будет флаг состояния кнопки. Добавим переменную

unsigned butcount=0,**butstate=0**;

Также нам нужен будет ещё один цикл, но не бесконечный, как я его назвал неправильно в видеоверсии, а условный, то есть тот, который выполняется, пока выполняется условие в скобках. Данный цикл мы вставим в тело цикла for. Условием будет определённое значения флага состояния кнопки. А в тело данного цикла мы вставим весь код отслеживания состояния кнопки, который мы написали на позапрошлом занятии, когда занимались с кнопкой. Также поубираем из данного кода управление состоянием порта D, а вместо этого, если кнопка будет нажата, то мы будем обнулять счётчик. Также в обоих случаях, то есть и в случае нажатой кнопке, и в случае отжатой, мы будем устанавливать статус кнопки в единицу, чтобы выйти из нашего цикла. А после того как мы выйдем из данного цикла, мы будем вызывать функцию отображения нашей цифры, затем вызывать задержку, а затем сбрасывать статус кнопки в ноль, чтобы в следующем цикле оператора for заново попадать в цикл с обработчиком кнопки. Вот такой вот у нас теперь получится код в бесконечном цикле

while(1)

{

  for (i=0;i<10;i++)

  {

**while(butstate==0)**

**{**

      if (!(PINB&0b00000001)){

        if (butcount < 5)

        {

          butcount++;

        }

        else

        {

**i=0;**

**butstate=1;**

        }

      }

      else

      {

        if (butcount > 0)

        {

          butcount--;

        }

        else

        {

**butstate=1;**

        }

      }

    }

    segchar(i);

    \_delay\_ms(500);

**butstate = 0;**

  }

}

Жирным шрифтом, как всегда, отмечен новый код. То есть, мы видим, что кода прибавилось немного, зато сколько получилось ветвлений. Но чем сложнее код, тем больше мы начинаем понимать в искусстве программирования. Вот теперь соберем код и проверим опять же в протеусе. Кнопка должна обнулять счётчик и счёт должен будет начинаться заново. В протеусе всё работает. Теперь прошьём контроллер и проверим на живом контроллере, живой кнопке и живом индикаторе. У нас также всё работает

