Программирование на языке С

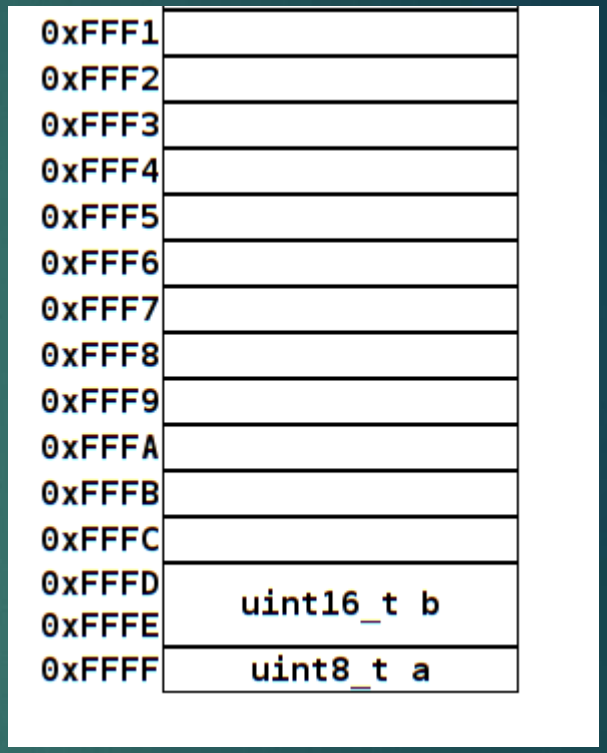
ЛЕКЦИЯ 3. МОДЕЛЬ ПАМЯТИ И МИКРОНОКТРОЛЛЕРЫ

Модель памяти языка С

Модель памяти языка С

Работа стека

```
uint16_t getSum(uint8_t arg1, uint16_t arg2) {
    uint16_t retval = arg1+arg2;
    return retval;
}

int main() {

    uint8_t a = 0;
    uint16_t b = 10;

    {
        uint16_t c,d;
        uint32_t e = c + d;
    }

    uint16_t sum = getSum(a, b);

    return 0;
}
```



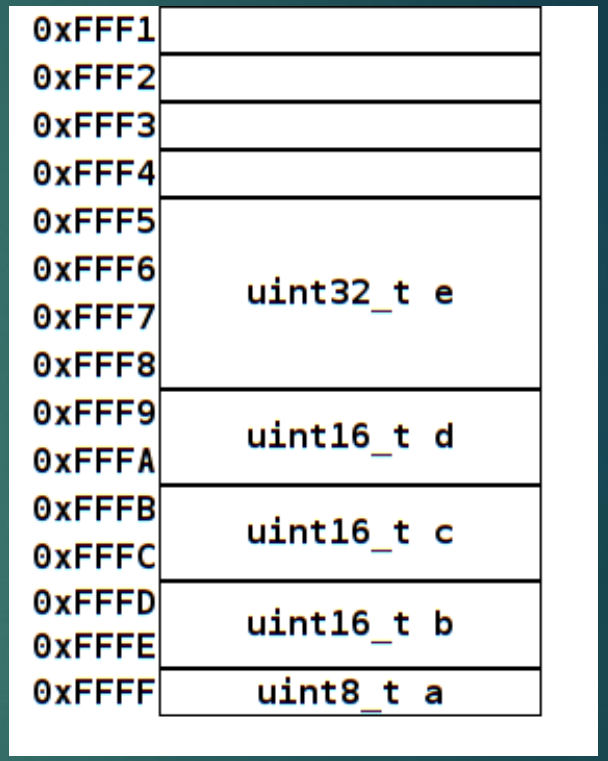
Код программы

Состояние стека

Модель памяти языка С

Работа стека

```
uint16_t getSum(uint8_t arg1, uint16_t arg2) {
    uint16_t retval = arg1+arg2;
    return retval;
}

int main() {

    uint8_t a = 0;
    uint16_t b = 10;

    {
        uint16_t c,d;
        uint32_t e = c + d;
    }

    uint16_t sum = getSum(a, b);

    return 0;
}
```



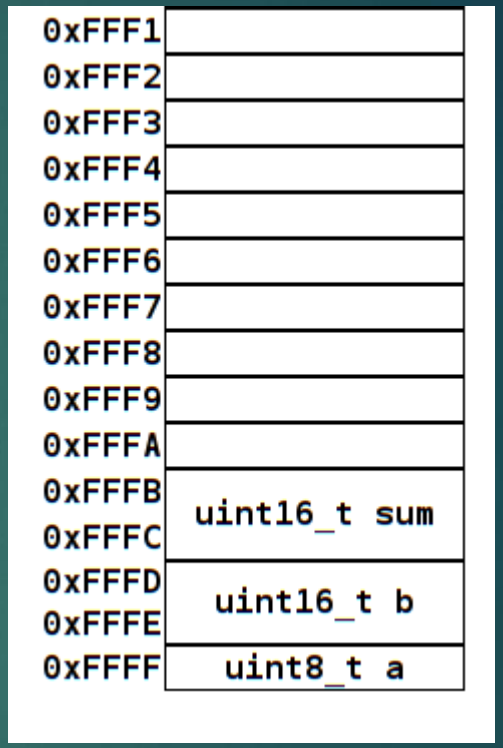
Код программы

Состояние стека

Модель памяти языка С

Работа стека

```
uint16_t getSum(uint8_t arg1, uint16_t arg2) {
    uint16_t retval = arg1+arg2;
    return retval;
}

int main() {

    uint8_t a = 0;
    uint16_t b = 10;

    {
        uint16_t c,d;
        uint32_t e = c + d;
    }

    uint16_t sum = getSum(a, b);

    return 0;
}
```



Код программы

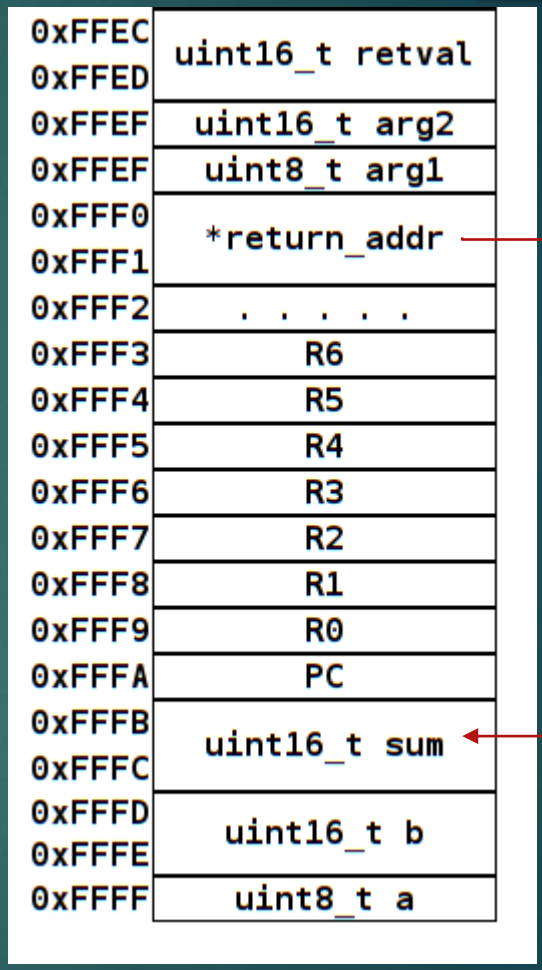
Состояние стека

Модель памяти языка С

Работа стека

```
uint16_t getSum(uint8_t arg1, uint16_t arg2) {
    uint16_t retval = arg1+arg2;
    return retval;
}

int main() {

    uint8_t a = 0;
    uint16_t b = 10;

    {
        uint16_t c,d;
        uint32_t e = c + d;
    }

    uint16_t sum = getSum(a, b);

    return 0;
}
```

Код программы

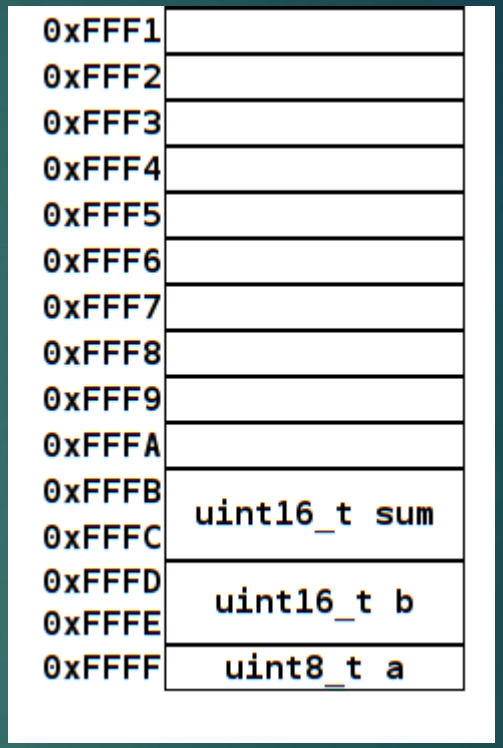


Состояние стека

Модель памяти языка С

Работа стека

```
uint16_t getSum(uint8_t arg1, uint16_t arg2) {
    uint16_t retval = arg1+arg2;
    return retval;
}

int main() {

    uint8_t a = 0;
    uint16_t b = 10;

    {
        uint16_t c,d;
        uint32_t e = c + d;
    }

    uint16_t sum = getSum(a, b);

    return 0;
}
```



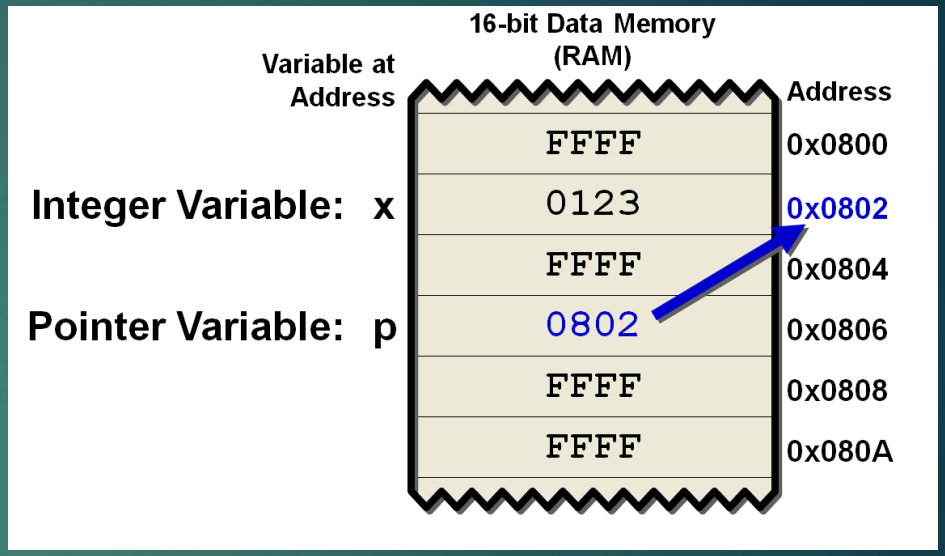
Код программы

Состояние стека

Указатели языка C

Доступ к памяти через указатели

```c
int main() {
    int x;
    int *p;

    p = &x;
    printf("p = %p\n", p);

    x = 0;
    printf("x до изменения = %d\n", x);

    *p = 10;
    printf("x после изменения = %d\n", x);

    return 0;
}
```



Вывод программы:

```
p = 0x7ffed142f464
x до изменения = 0
x после изменения = 10
```

Указатель, это переменная, которая хранит в себе адрес другой переменной определенного типа.

Фактически, указатель это целое число типа size_t из файла <stddef.h> (которое как правило определено как: **typedef unsigned int size_t**).

Указатели языка С

Базовые операции с указателями

```c
typedef struct {
    int x,y,z;
} CustomStruct;

int main() {
    // объявление указателей
    int *ptr;
    double *ptr1, *ptr2;
    CustomStruct *structPtr;

    // операция взятия адреса
    // и инициализация указателей
    int x;
    ptr = &x;
    ptr1 = 0x10;

    // Разыменование указателя
    *ptr = 10;
    int y = *ptr + 10;

    return 0;
}
```

Определяется указатель как **(Тип) * (имя_указателя)**;

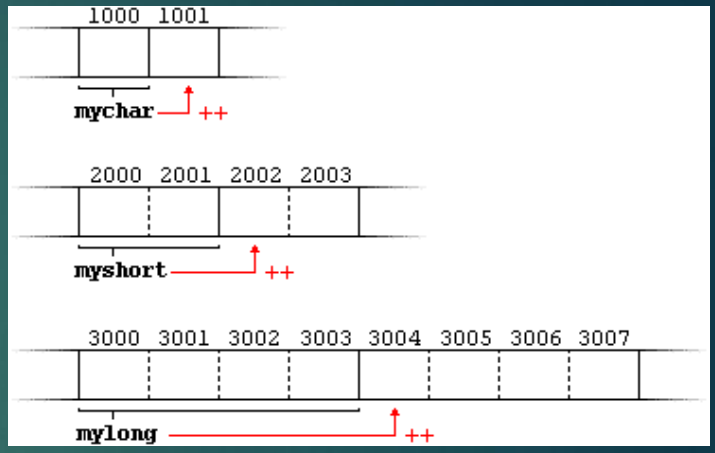
Для указателей определена операция *разыменовывания*. Она описывается в коде как ***имя_указателя**. Разыменовывания указателя возвращает объект, на который он указывает.

Для всех переменных определена операция *взятия адреса*. Эта операция возвращает адрес переменной в памяти

Указатели языка С

Указательная арифметика и массивы

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

int main() {
    // Указательная арифметика
    int a;
    int * ptr = &a;
    ptr++; // теперь ptr показывает на int следом за a

    // Элементы массива располагаются в памяти подряд
    // а сам массив - это фактически указатель на его начало
    int array[10];
    int * arrayPtr = array;

    // Доступ к элементу массива через указатели
    bool isTrue = (arrayPtr+5 == &array[5]);
    // или *(arrayPtr+5) и array[5] это одно и тоже

    // Указатели даже можно использовать как массивы и наоборот
    int elem = arrayPtr[5];
    // одно и тоже, что
    elem = *(array+5);
```



Указатели языка C

Преобразования типов указателей

```
int main() {
    // uint32_t на стеке
    uint32_t x = 0x12345678;
    // указатель uint32_t* на него
    uint32_t *intPtr = &x;

    // указатель uint8_t* на ту же область памяти
    uint8_t *bytesPtr = (uint8_t*)intPtr;

    // печатаем байты uint8_t
    for (size_t i = 0; i < sizeof(x); i++) {
        printf("bytesPtr[%d] @ %p = 0x%X\n", i, bytesPtr+i, *(bytesPtr+i));
    }


    return 0;
}
```

Вывод программы:

```
<terminated> example [C/C++ Application] /home/snork/pr
bytesPtr[0] @ 0x7ffd7550dda4 = 0x78
bytesPtr[1] @ 0x7ffd7550dda5 = 0x56
bytesPtr[2] @ 0x7ffd7550dda6 = 0x34
bytesPtr[3] @ 0x7ffd7550dda7 = 0x12
```

Указатели языка C

Указатель void *

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

// Стуктура статуса зонда
typedef struct {
    uint8_t legsSensorOk:   1, // Статус выдвижения ног
      parachuteSensorOk:    1, // Статус парашюта
      radioModuleOk:        1, // Статус радиомодуля
      thermometerOk:        1, // Статус термометра
      pressuremetherOk:     1, // Статус барометра
      gpsModuleOk :         1  // Статус GPS модуля
    ;

    uint16_t position[3]; // Положение WGS84 [x,y,z]
    uint16_t velocity[3]; // Скорость WGS84 [vx, vy, vz]
} SpaceshipState;


void printBytes(void * dataPtr, size_t dataSize) {
    // C void* нельзя совершать математических операций
    // и разыменовывать, поэтому преобразуем его в uint8_t*
    uint8_t* bytesPtr = (uint8_t*)dataPtr;

    for (size_t i = 0; i < dataSize; i++) {
      printf("dataBytes[%zd] = 0x%02X\n", i, bytesPtr[i]);
    }
}

int main() {
    SpaceshipState state = {1,0,1,0,1,0, {1,2,3}, {4,5,6}};

    // любые указатели неявно преобразуются в void*
    // поэтому явное преобразование (void*) тут не нужно
    printBytes(&state, sizeof(state));

    return 0;
}
```

С указателем void * нельзя совершать математических операций и операций разыменовывания (компилятор укажет на ошибку)

Это чистая абстракция – указатель указывающий на «нечто»

Вывод программы:

```
<terminated> example [C/C++ Application] /home/snork/prog/
dataBytes[0] = 0x95
dataBytes[1] = 0xC2
dataBytes[2] = 0x01
dataBytes[3] = 0x00
dataBytes[4] = 0x02
dataBytes[5] = 0x00
dataBytes[6] = 0x03
dataBytes[7] = 0x00
dataBytes[8] = 0x04
dataBytes[9] = 0x00
dataBytes[10] = 0x05
dataBytes[11] = 0x00
dataBytes[12] = 0x06
dataBytes[13] = 0x00
```

Указатели языка С

Доступ к полям структуры через указатель и «возвращаемые» аргументы функций

```
typedef struct {
    int a,b,c,d;
} MyCustomStruct;


void printStructFieldAndInc(MyCustomStruct * ptr){
    printf("field a = %d\n", ptr->a);
    // для доступа к полям структуры через указатель можно использоваться символ -> вместо .
    ptr->a = 10; // тоже самое что и (*ptr).a = 10;
}

int main() {
    // выделение памяти под пользовательскую структуру
    MyCustomStruct data = {0, 1, 2, 3};
    printf("Поля структуры до вызова %d %d %d %d\n", data.a, data.b, data.c, data.d);
    printStructFieldAndInc(&data);
    printf("Поля структуры после вызова %d %d %d %d\n", data.a, data.b, data.c, data.d);

    return 0;
}

```

В отличии от изменений аргументов «переданных по значению», изменения аргуменентов переданных по указателю возвращаются в подпрограмму верхнего уровня. Сам указатель при этом не меняется, так как он передается «по значению».

Указатели языка C

Указатели и модификатор **const**

В некоторых случаях, помимо возможности «возвращения» изменений аргументов передача аргументов по указателю еще и более эффективна в плане производительности и объемов памяти.

Например, при передаче большой структуры как аргумента «по значению», в стек копируются все её поля. При передаче структуры «по указателю» копируется только лишь указатель нее.

Если «возвращение изменений» аргумента из функции при этом является не желательным, его можно явно запретить, объявив аргумент указателем на константу

```
4
5  typedef struct {
6      int a,b,c,d;
7  } MyCustomStruct;
8
9  void cantTouchThis(const MyCustomStruct * constPtr) {
10     int innerValue = constPtr->a; // все в порядке
11     constPtr->a = 10; // Ошибка, нельзя изменять константу
12
13     // Если для логики программы нужны изменения в структуре - можно сделать локальную копию и работать с ней
14     MyCustomStruct copy = *constPtr;
15     constPtr = &copy; // изменения указателя (а не того, что на он указывает) обратно не передаются
16 }
```

Указатели языка C

Указатели и модификатор **const**

При том, что объект, на который указывается указатель обозначенный как **const T * ptr** изменять нельзя, сам указатель при этом изменять можно (например переуказать на другой объект в памяти).

Эту возможность так же можно ограничить, но для этого нужно указать модификатор **const** после ***

```
int main() {
    int value1, value2;

    const int *constPtr = &value1;
    *constPtr = 10; // ошибка, нельзя изменять константу
    constPtr = &value2; // перенаправить сам указатель при этом можно

    int * const ptrConst = &value1;
    *ptrConst = 10; // без проблем. Значение, на которое указывает указатель? не защищено как константа
    ptrConst = &value2; // А вот это нельзя. Указатель константен

    const int * const constPtrConst = &value1;
    *constPtrConst = 10; // нельзя
    constPtrConst = &value2; // тоже нельзя

    return 0;
}
```

Указатели языка C

Значение NULL

Это специальной макрос, определенный в стандартной библиотеке, который обозначет указатель «в никуда». Такой указатель нельзя разыменовывать.

Как правило, под значением NULL используется обычный 0

При помощи этого значения удобно делать опциональные аргументы функций.

Указатели языка С

Указатель на указатель

Поскольку указатель это тоже переменная и тоже хранится в памяти – его адрес так же можно взять. Получится тип «указатель на указатель», который определятся как

тип ** имя_указателя

```
5  int main() {
6      int value; // int на стеке
7      int * valuePtr = &value;    // указатель на value
8      int ** valuePtrPtr = &valuePtr; // указатель на указатель на value
9
10     return 0;
11 }
12
```

Полная аналогия двумерных массивов. Используется редко, либо в случаях, когда функция в аргументе должна вернуть указатель, тогда нужно указатель передать по указателю, либо при передаче двумерного массива в функцию.

Двумерные массивы (и соответсвенно указатели на указатели) это как правило списки строк, так как строка это уже массив типа char.

Указатели языка С

Указатель на указатель на указатель

Поскольку указатель на указатель это тоже переменная для которой определена опрация взятия адреса...

Допустимыми являются конструкции
int *** ptr;
И даже int ********* ptr;

На практике такие указатели, как и массивы размером с количеством измерений более двух, используются крайне редко (считай не используются вовсе)

Работа с кучей

Для управления памятью в куче нужен специальный программный компонент – «аллокатор».

Это сложная программа, которая управляет динамическими переменными, создаваемыми и удаляемыми во время выполнения программы.

Работа с кучей

Для доступа к куче используются две функции из файла **<stdlib.h>**

```
void * malloc(size_t memBlockSize);
void free(void * memBlockSize);
```

malloc (от memory allocate) выделяет в куче блок памяти указанного размера и возвращает на него «обезличенный» (void*) указатель. Если выделение памяти не удалось (скорее всего это значит, что она просто закончилась) **malloc** вернет NULL

Когда выделенный блок становится не нужен приложению, оно должно вызвать функцию **free** и передать ей указатель на не нужный блок. После этого блок возвращается в кучу и может быть заново аллокирован.

Из-за высоких накладных расходов в плане производительности и проблемы фрагментации памяти использование кучи не рекомендуется в приложениях для встраиваемых устройств.

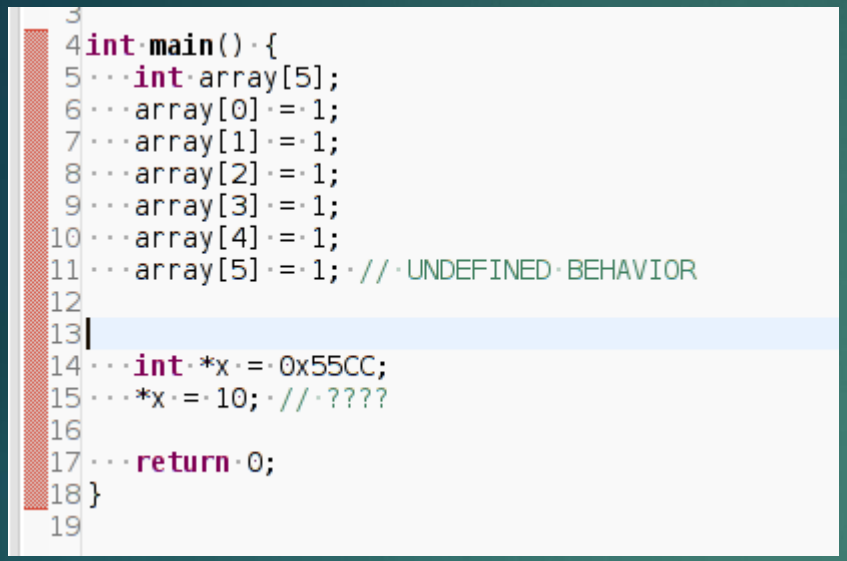
Работа с кучей

Пример

```c
#include <stdlib.h>


typedef struct {
    int a,b,c,d;
} MyCustomStruct;

int main() {
    // выделение памяти под пользовательскую структуру
    MyCustomStruct* structPtr = (MyCustomStruct*)malloc(sizeof(MyCustomStruct));

    // ... работа со structPtr;

    // освобождение памяти
    free(structPtr);
}
```

Сам по себе блок памяти не освободится (если не вызвана free). Ошибки с неосвобожденными блоками памяти называются «утечками памяти» и являются одними из самых трудно устранимых ошибок в программировании на C/C++ и других языках с подобной моделью памяти

Опасность указателей



```
int main() {
    int array[5];
    array[0] = 1;
    array[1] = 1;
    array[2] = 1;
    array[3] = 1;
    array[4] = 1;
    array[5] = 1; // UNDEFINED BEHAVIOR

    int *x = 0x55CC;
    *x = 10; // ????

    return 0;
}
```

Помимо опасности с утечками памяти, указатели опасны сами по себе. Чтение и запись пол указателю, который указывает непонятно куда может привести к самым неожиданным ошибкам, которые очень тяжело отлавливать.

Поэтому при работе с указателями и массивами нужно быть предельно внимательным.

Выравнивание структур

Компилятор может вставлять паразитные поля в структуры между её членами для оптимизации обращений процессора к памяти, занимаемой ими. Правила по которым происходит это выравнивание достаточно сложны и зависят от компилятора и его настроек. Один из способов «борьбы» с этим – использование директивы компилятора **#pragma pack**

Хорошая статья на тему:
http://habrahabr.ru/post/142662/

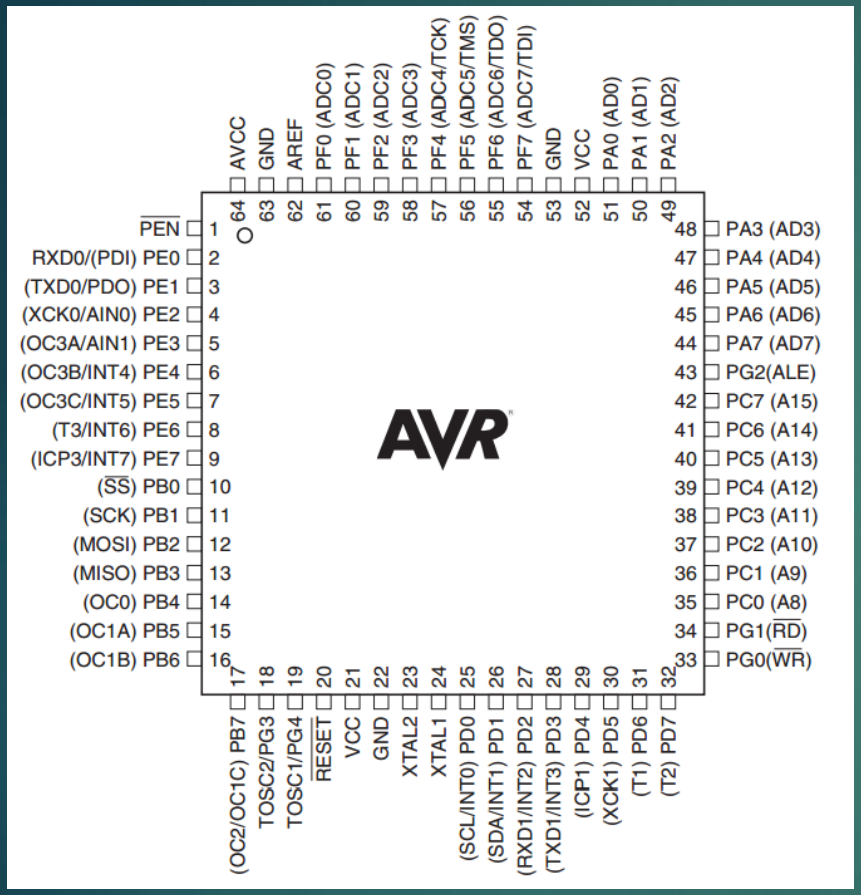
```c
#include <stdio.h>
#include <stdint.h>

typedef struct {
    uint8_t x;
    uint16_t y;
    uint32_t z;
    uint16_t a;
} CustomStruct;

int main() {
    printf("sizeof = %zd\n", sizeof(CustomStruct));

    return 0;
}
```

Вывод: sizeof = 12

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


#pragma pack(push, 1)

typedef struct {
    uint8_t x;
    uint16_t y;
    uint32_t z;
    uint16_t a;
} CustomStruct;

#pragma pack(pop)

int main() {
    printf("sizeof = %zd\n", sizeof(CustomStruct));

    return 0;
}
```

Вывод: sizeof = 9

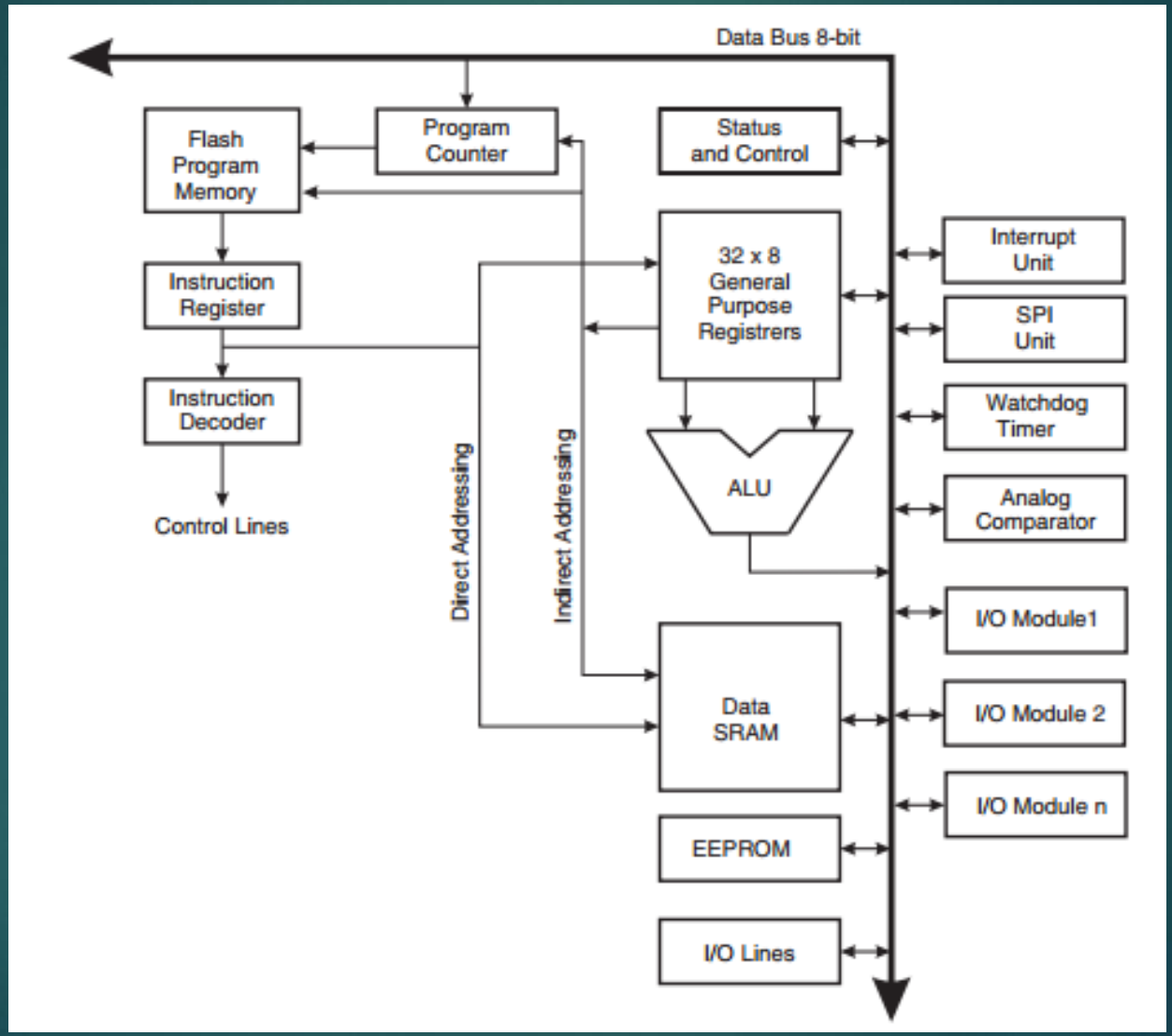
ATmega128



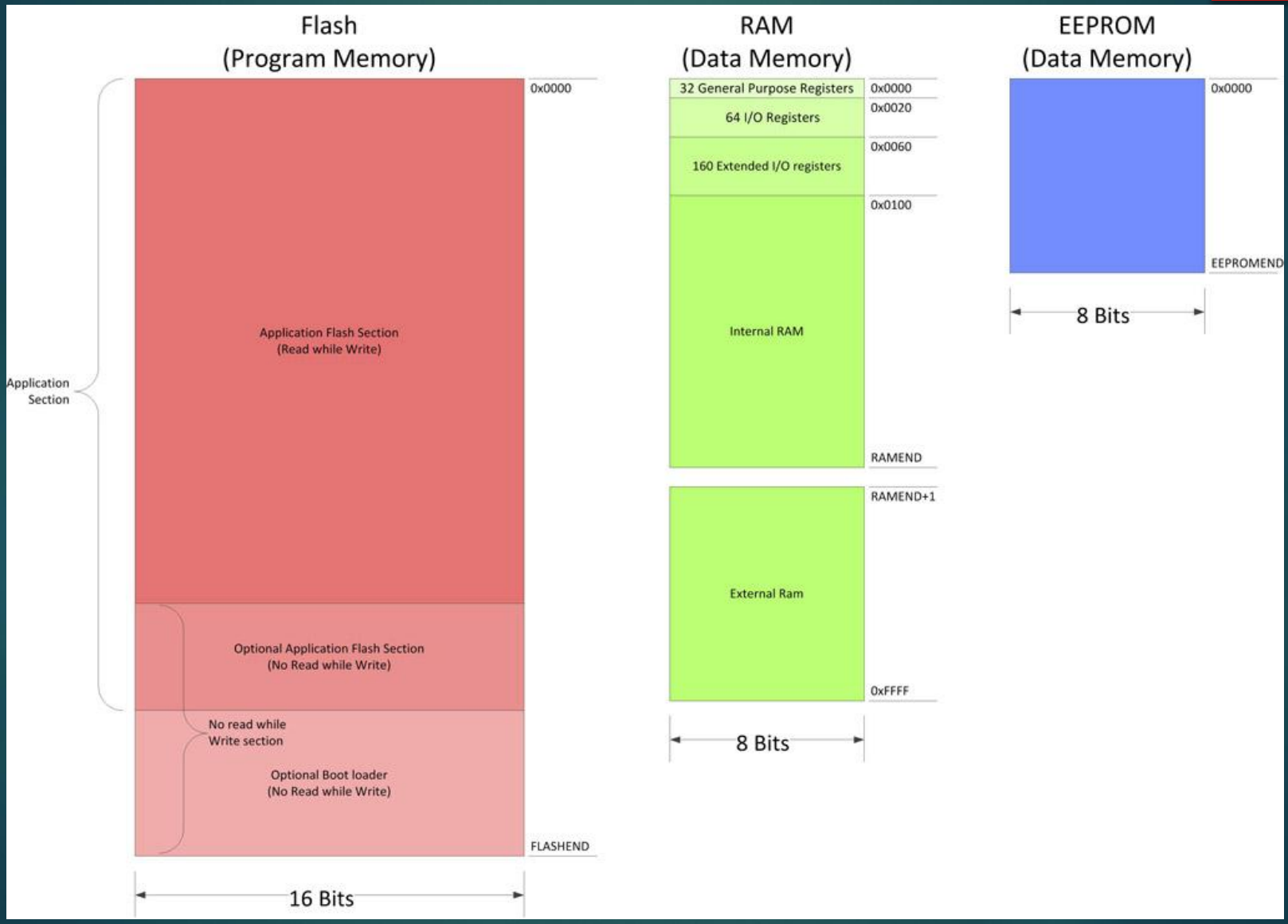
Выводы ATmega128

ATmega 128 в конструкторе cansat

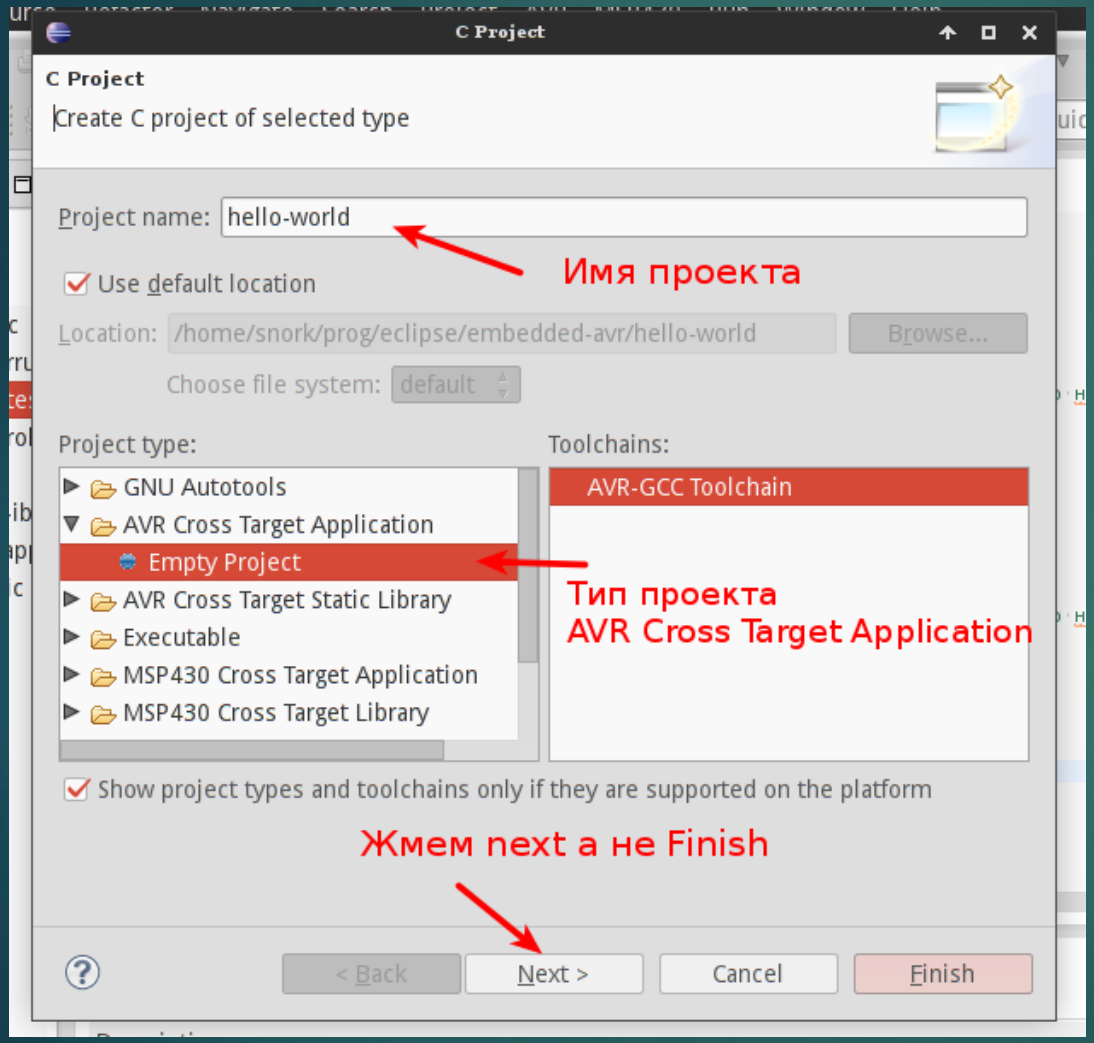
Ядро AVR


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n
Data SRAM
EEPROM
I/O Lines

Карта памяти



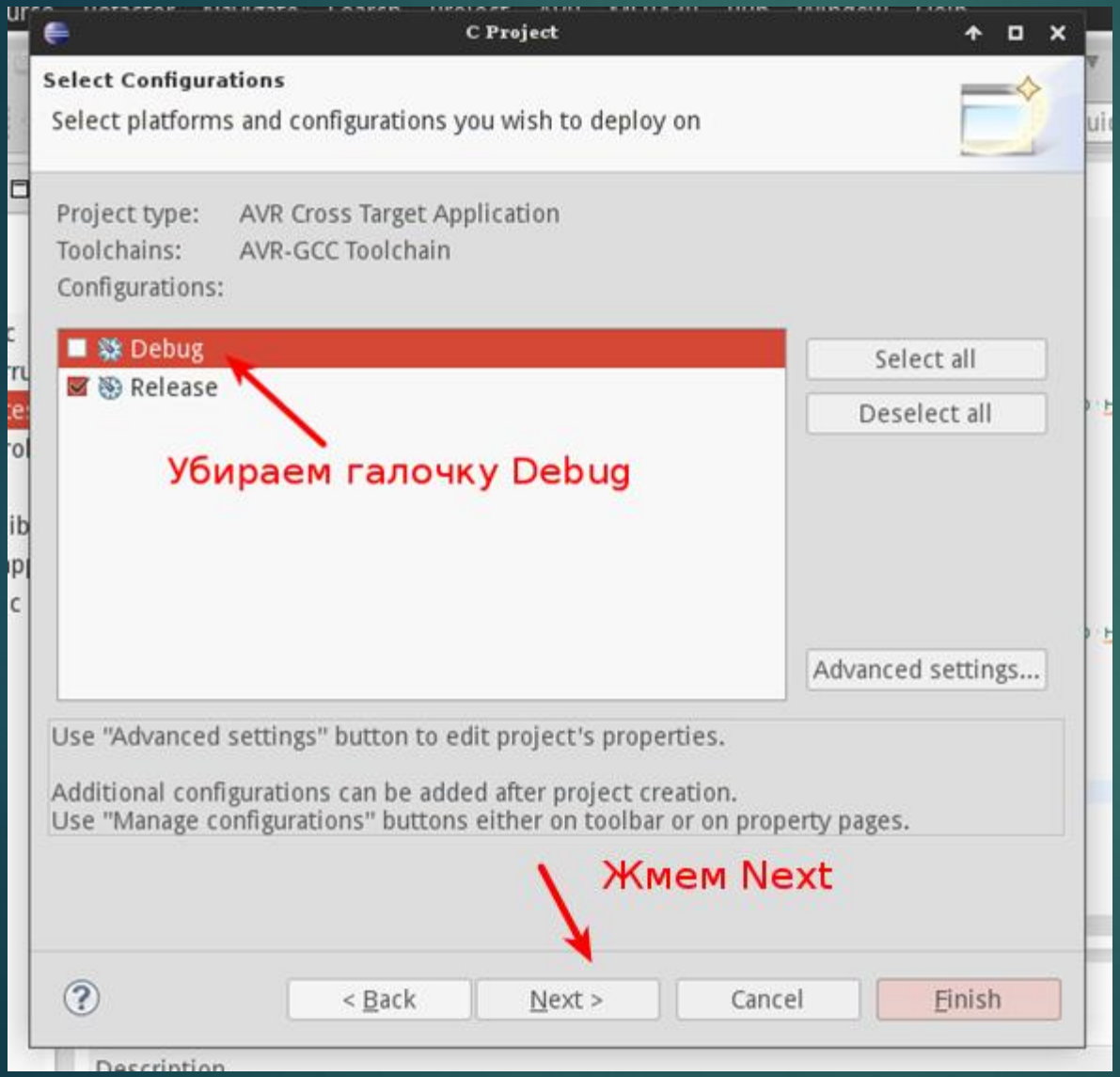
Создание и настройка проекта AVR



Проект для микроконтроллера создается аналогично проекту для настольного комьюптера

Нужно только выбрать другой тип проекта

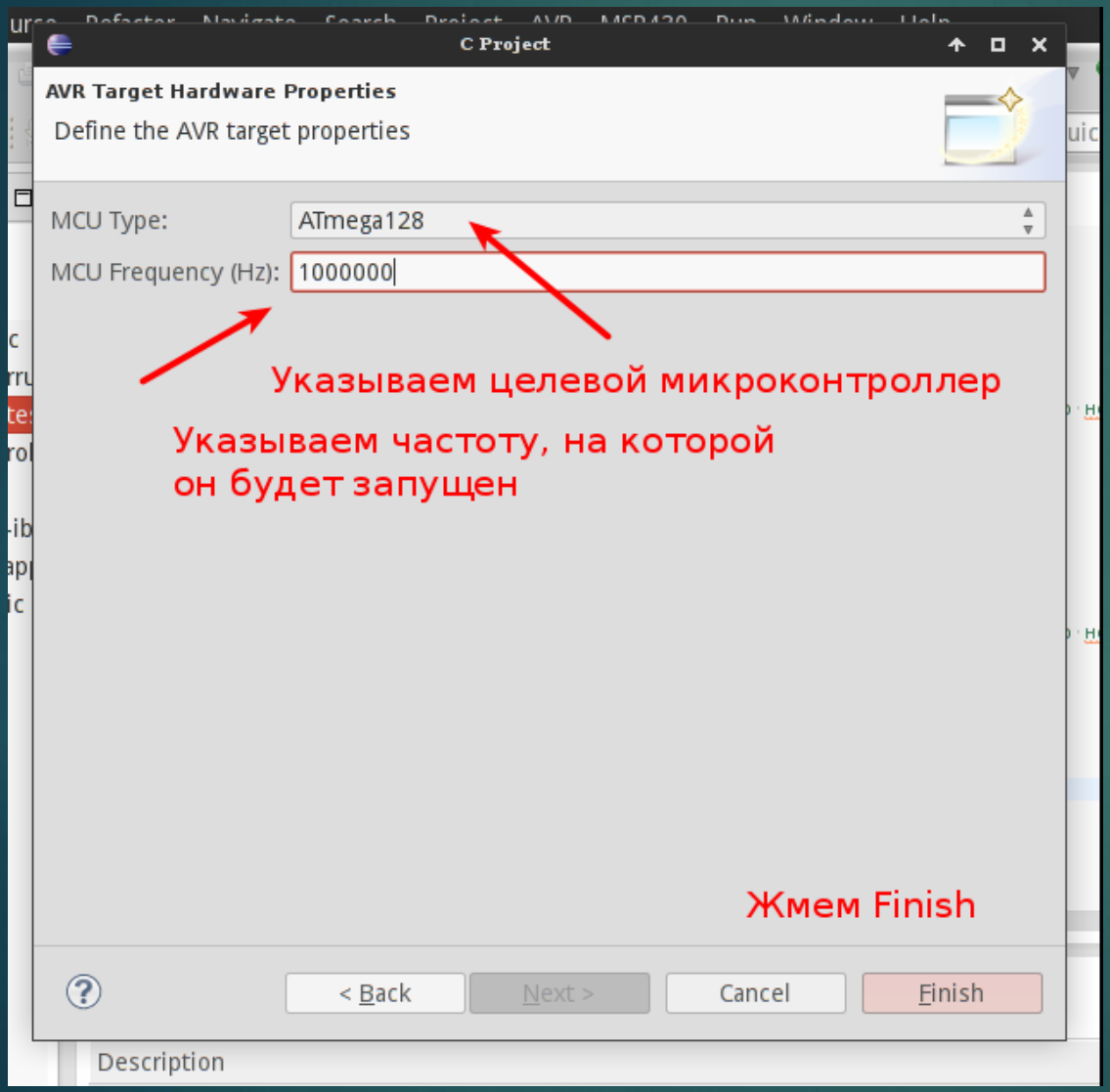
Создание и настройка проекта AVR



Отладочную конфигурацию проекта стоит отключить. Скорее всего у нас не будет внутрисхемного отладчика, с которым она могла бы быть полезна.

Простую прошивку по кнопке отладочной конфигурации eclipse не настраивает, что может вызывать ошибки и путаницу при сборке и прошивке проекта

Создание и настройка проекта AVR



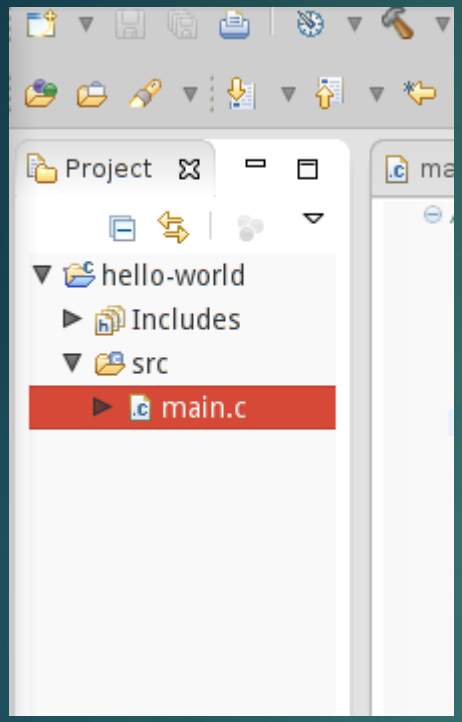
Частота – не настройка контроллера.

Этот параметр будет передан библиотеке avr-libc, для правильного рассчета параметров, зависящих от частоты.

Фактическая частота контроллера указывается иначе.

Пока не понятно на какой частоте будет запущен контроллер, но можно предположить, что это 16 мГц

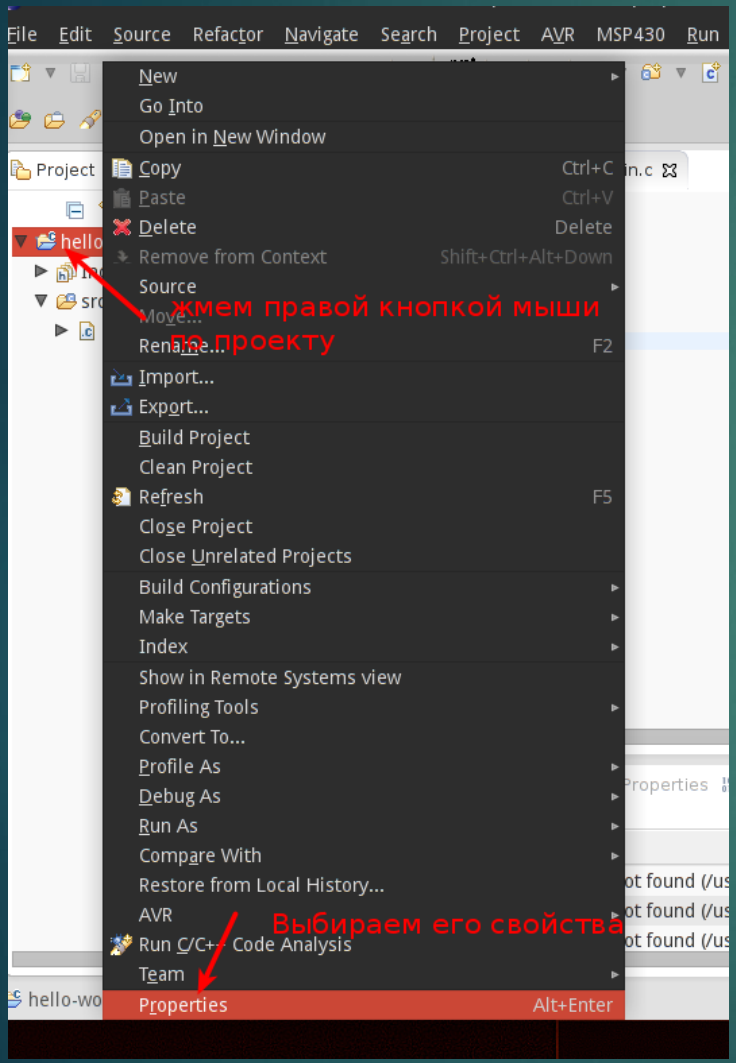
Создание и настройка проекта AVR



Папки с исходниками
и файлы исходников
добавляются так же как в
обычном проекте

Смотри лекцию 1

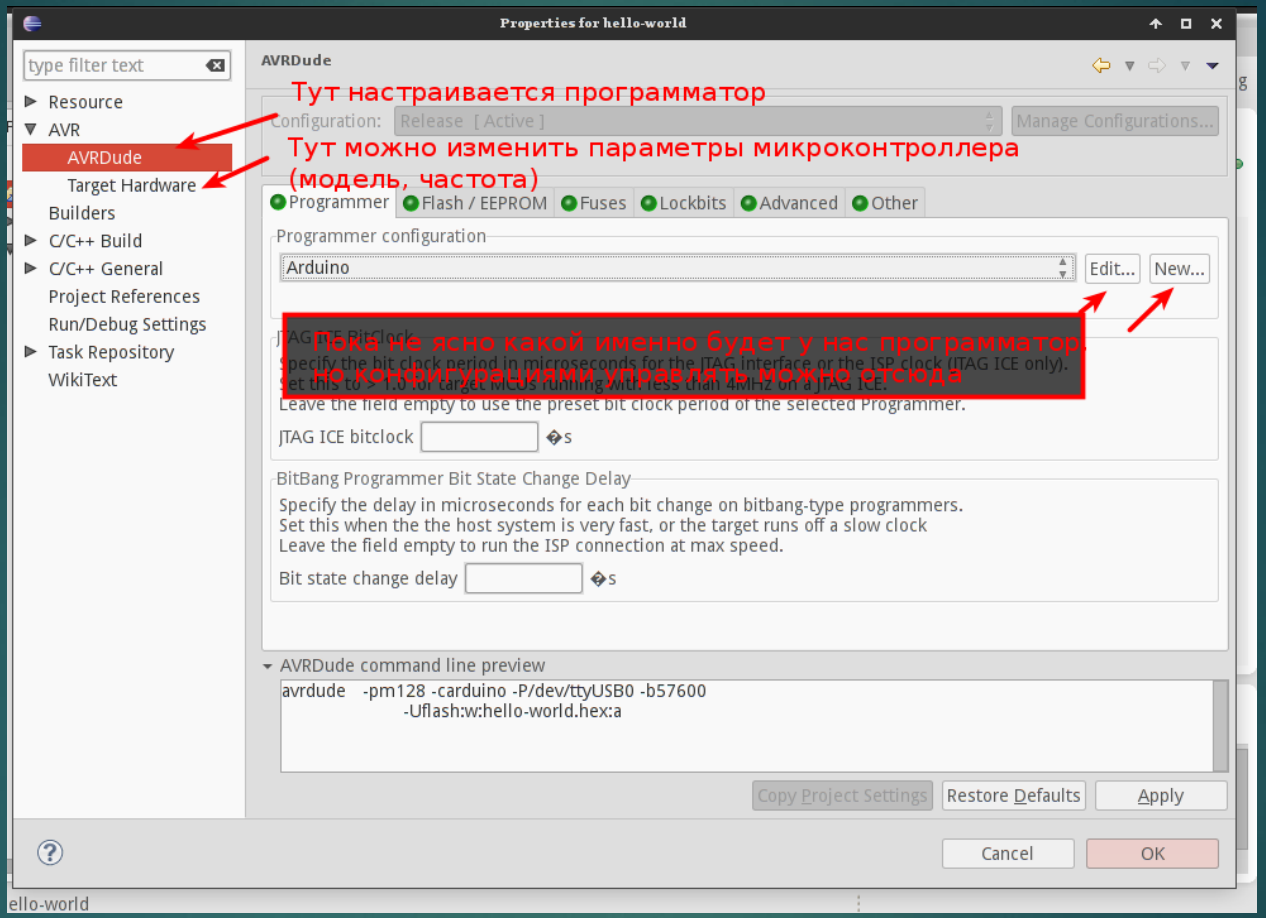
Создание и настройка проекта AVR



Идем настраивать программатор.

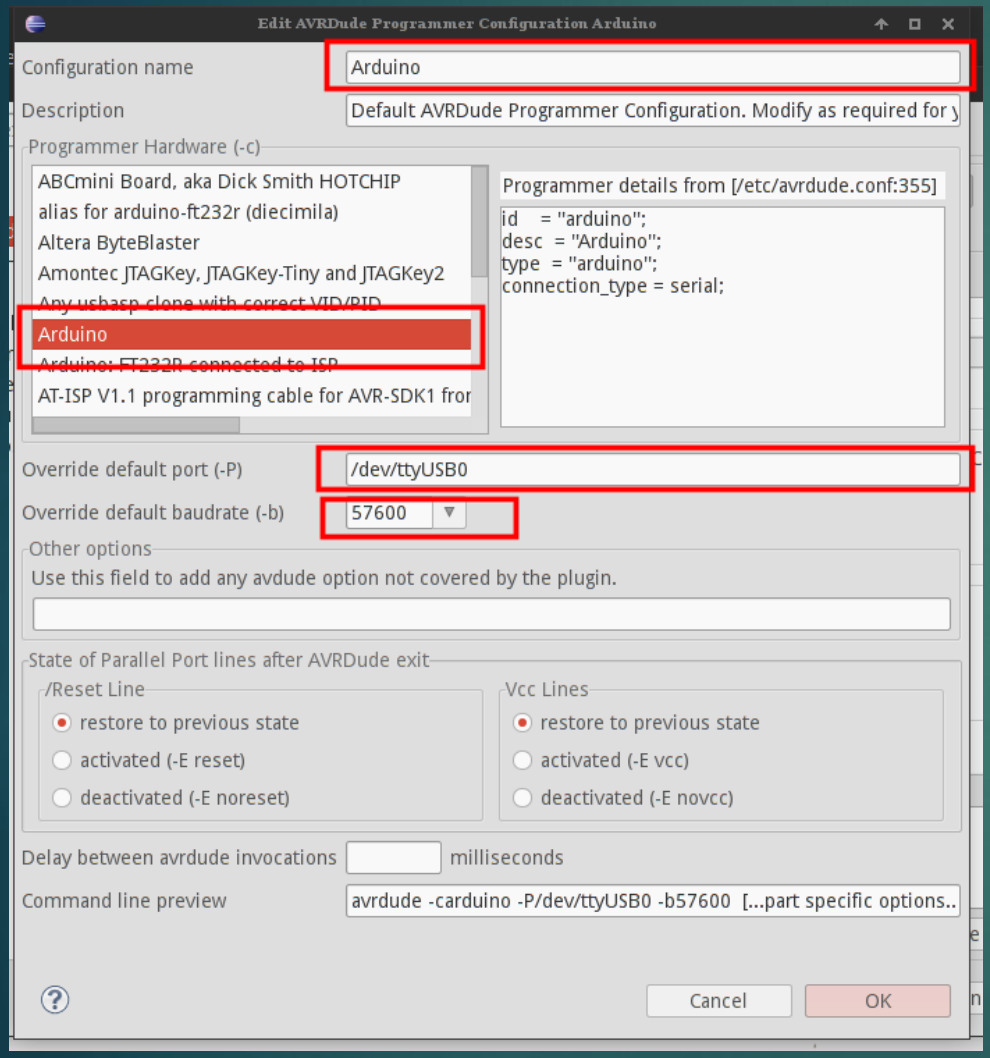
Открываем свойства проекта

Создание и настройка проекта AVR



Для отправки прошивки на контроллер нужны две вещи: аппартное устройство (программатор) и программа им управляющая.
Мы будем использвать программу avrdude, по аппаратной части пока ничего не понятно.

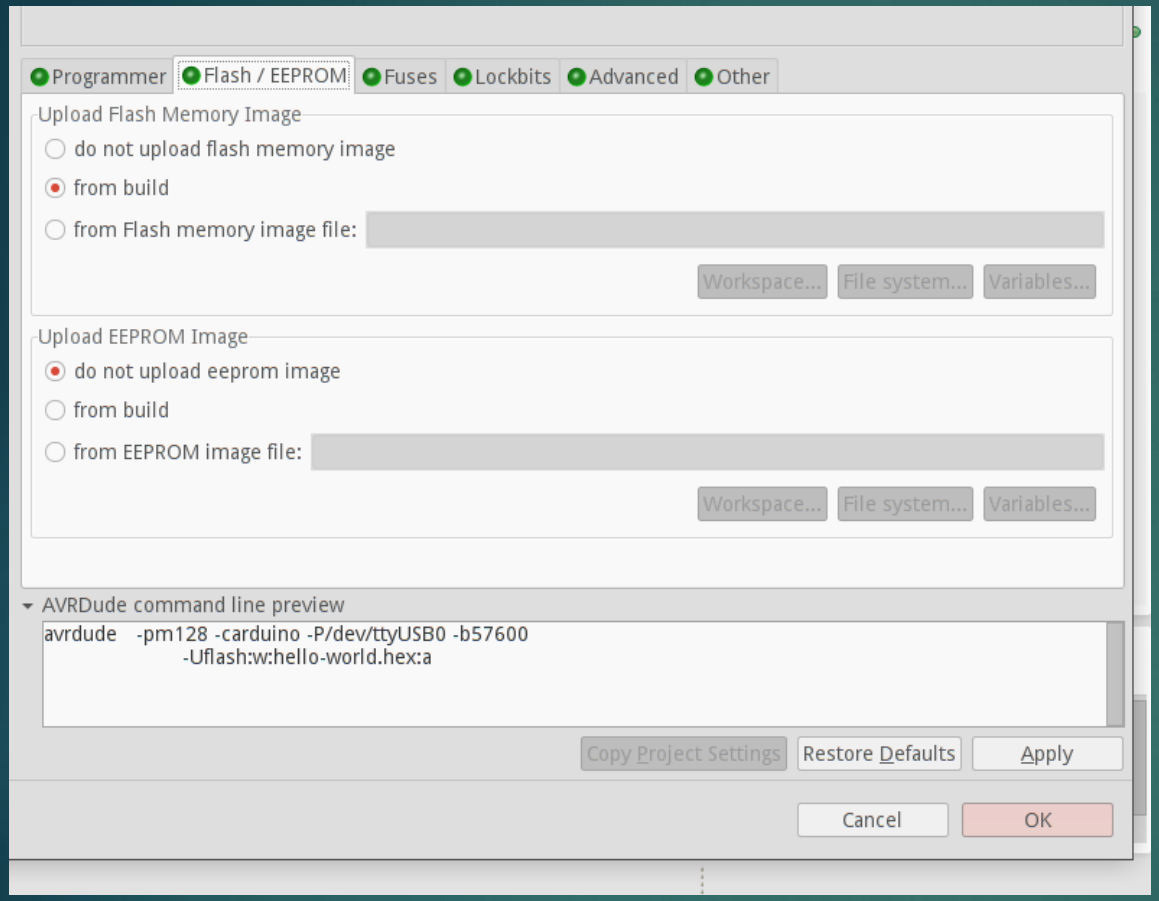
Создание и настройка проекта AVR



Как пример конфигурация avrdude для arduino

При этом нужно указать соответсвующий Микроконтроллер и частоту, так как на arduino используются не 128ые атмеги

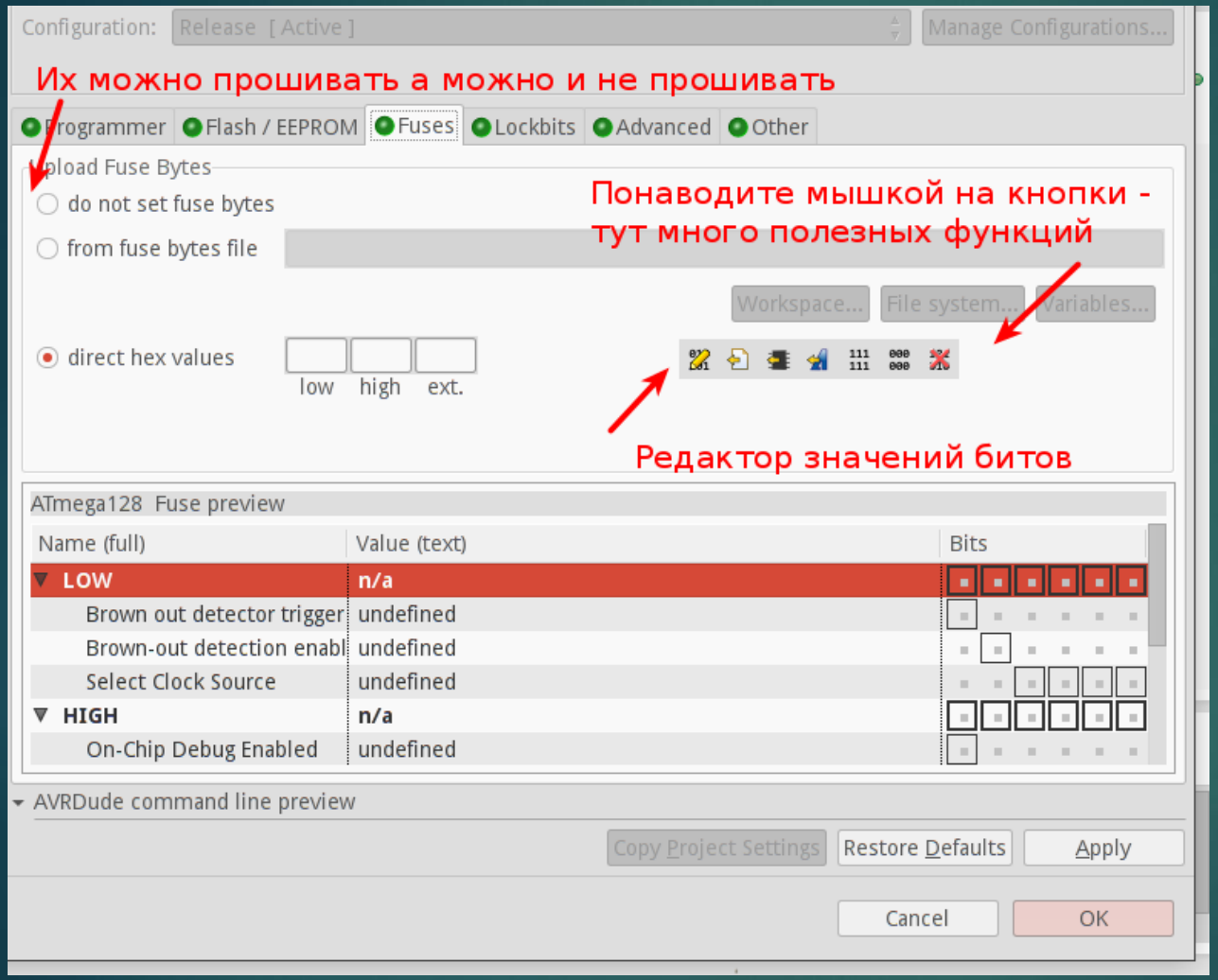
Создание и настройка проекта AVR



Вкладка настройки заливаемых на контроллер образов.

Тут можно включить прошивку EEPROM образа

Создание и настройка проекта AVR



Настройка FUSE битов (и аналогичная для LOCK битов)

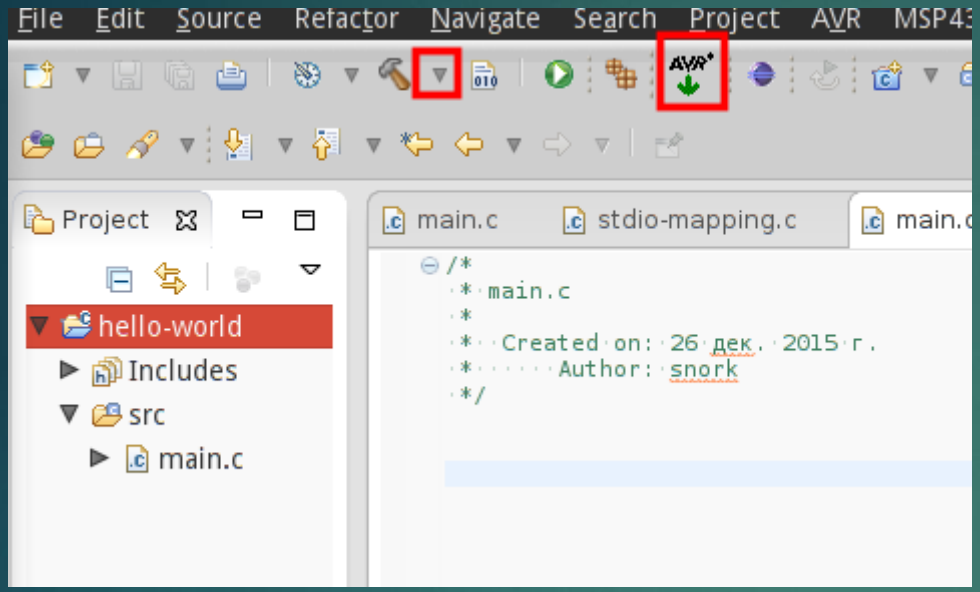
Fuse биты это биты трёх байтов контроллирующие самые критичные элементы системы, например частоту и источник тактовых импульсов.

Lock биты позволяют заблокировать контроллер для перепрошивки и загрузки прошивки с него.

Править по документации к контроллеру и **ОЧЕНЬ** внимательно, есть шансы испортить контроллер без возможности восстановления

Хорошая публикация по fuse битам **(обязательно к прочтению)**:
http://easyelectronics.ru/avr-uchebnyj-kurs-konfiguraciya-fuse-bit.html

Создание и настройка проекта AVR



При сборке проекта убедитесь, что собираете конфгирацию Release (нажав на маленькую стрелочку у молотка – иконки сборки.

После успешной сборки проекта – по кнопке со значком AVR прошивка будет залита в микроконтроллер

Следите за ошибками в консоли, они не выделяются красным.

Домашнее задание

Написать задачи из анкеты на языке С

Условия:

7. Пусть имеется переменная Ut, которой ежесекундно присваивается значение напряжения на электрических контактах аналогового датчика температуры. Связь температуры и напряжения определяется зависимостью:

$$T(Ut) = minT + (maxT - minT) \cdot \frac{Ut}{Umax}$$

где minT, maxT – границы измерения температуры датчиком, Umax – максимальное напряжение на датчике.

На любом языке программирования составьте программу определения, хранения и вывода на экран среднего значения температуры за последние 5 секунд наблюдения.

8. Предположим, что после выполнения миссии Вашего аппарата по результатам N измерений Вы получили массив arrH размерностью 2xN, который содержит следующую информацию:

элемент arrH[0,i] – содержит время i-того наблюдения;

элемент arrH[1,i] – содержит высоту аппарата во время i-того наблюдения.

Придумайте и опишите способ определения и идентификации промежутков времени, для которых, используя данную информацию, возможно рассчитать среднюю скорость движения.

На любом языке программирования составьте программу вычисления средней скорости движения для всех возможных промежутков времени.

Материалы для самостоятельного изучения

По языку Си

Брайан Керниган, Деннис Ритчи - Язык программирования Си. (*учебник от авторов языка*)

Герберт Шилдт - Полный справочник по C (*содержит отличное описание стандартной библиотеки Си*)

http://www.cplusplus.com/reference/clibrary/ - Документация к стандартной библиотеке Си

Материалы для самостоятельного изучения

По языку Си:

- Брайан Керниган, Деннис Ритчи - Язык программирования Си. (*учебник от авторов языка*)
- Герберт Шилдт - Полный справочник по C (*содержит отличное описание стандартной библиотеки Си*)
- http://www.cplusplus.com/reference/clibrary/ - Документация к стандартной библиотеке Си
- http://stackoverflow.com/ Платформа вопросов/ответов на любые тематики в том числе и по программированию. На большинство вопросов там уже есть ответ

Материалы для самостоятельного изучения

По AVR:

- Документация на МК на официальном сайте Atmel
- Документация avr-libc http://www.atmel.com/webdoc/AVRLibcReferenceManual/index.html
- Материалы портала http://easyelectronics.ru и в частности учебный курс по AVR http://easyelectronics.ru/category/avr-uchebnyj-kurs
- Материалы форума http://www.avrfreaks.net/
- Белов А.В. - Самоучитель разработчика устройств на микроконтроллерах AVR
- Белов А.В. - Микроконтроллеры AVR в радиолюбительской практике