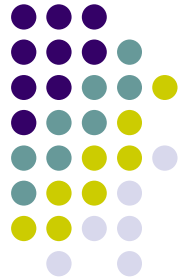
Embedded Systems
Programming and Architectures

Lecture No 9 : Programming in C the PIC 16F series

Dr John Kalomiros
Assis. Professor
Department of Post Graduate studies in Communications and Informatics

Programming embedded systems in C language: An introduction using the PIC16F series



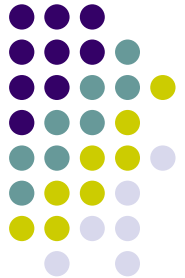
- With increasingly complicated programs, it becomes more difficult to apply assembler programming:
  - Difficult debugging
  - Difficult flow control
  - Difficult to implement mathematical tasks

A number of high-level languages are used in ES development: BASIC, PASCAL and C

In this lesson we introduce the Hi-Tech ANSI-C compiler which is bundled with MPLAB and fully supports the 16F midrange MCUs.

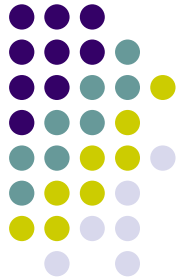
There are other compilers too, like the Microchip C18 for the 18F series. You can find a free student edition for some of them.

C is a HLL that remains close to hardware



- ✓ C language is a well defined standard established throughout the computer industry. It is used for development in many platforms including the desktop computer.
- ✓ It is a portable language that allows transferring programs from one computer to the other with minimal modifications. In Embedded Systems, hardware-specific programming is MCU dependent, while the main logic remains the same when migrating from one processor to the other.
- ✓ Use of C is possible in ES programming because modern MCUs are equipped with larger data and program memories.
- ✓ C programming is easier. However, assembly programming is more efficient and hardware-friendly.

In this course we assume that the audience is acquainted with C language…



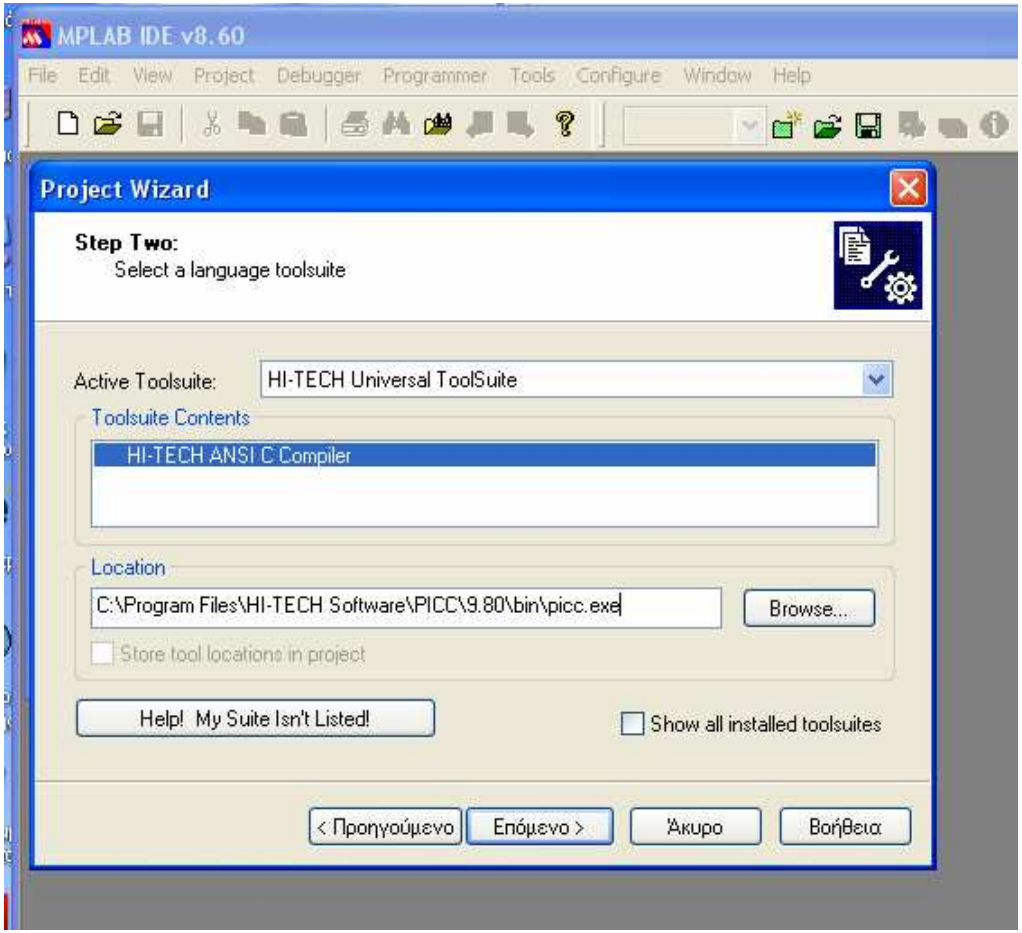
…However, we provide here some information and a PIC-specific textbook, for those who need to refresh their memory…

- Download the e-book: *PIC micro MCU C by Nigel Gardner*

- You are also expected to study chapter 14 (Introducing C) of *Designing Embedded Systems with PIC microcontrollers by Tim Wilmshurst*, which is core literature for this course.

- You may also refer to documents *PIC C Lite User's Guide*, by Hi-Tech software (PICC_lite.pdf) and *Hi-Tech C for 10/12/16 MCUs User's Guide, by Microchip.*

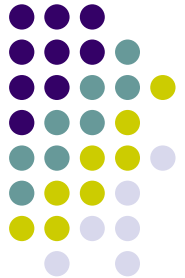
Choose the right compiler in MPLAB IDE



In Step two of the project wizard select the Hi-TECH Universal ToolSuite

With File-New write a program in C and save as xxx.c file in the project directory

Components of a C program



- Declarations create program elements, like variables and functions and indicate their properties. Example:

```
unsigned char counter;
```

- Definitions establish the content of variables or functions and allocate memory
- Statements perform mathematical or logical operations and establish program flow

```
PORTB=0xAF;
```

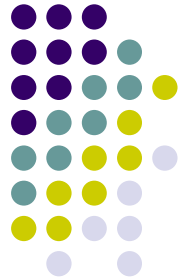
- Code blocks: declarations and statements grouped together and contained within curly brackets

```
While(1)
{
…statements…
}
```

- Space and comments /* comment goes here */ or // rem out line like this

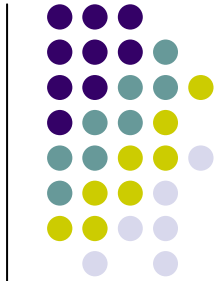
  also: leave empty space for readability

Example of a very simple program in C for midrange PIC MCUs



```
/*READS PORTD and OUTPUTS READING+1 TO PORTB */
#include <htc.h>    //or #include "pic1687x.h" for PIC16F877
/*Set CONFIGURATION BITS in code */
__CONFIG (UNPROTECT & PWRTDIS & WDTDIS & XT & LVPDIS);
unsigned int z=0;//variable declarations. Discuss the use of int!
//main function
void main (void)
{
        TRISD=0xFF; /* Set PORTD as input*/
        TRISB=0x00; /* Set PORTB as output */
        PORTB=0x00; /* Clear PORTB */
        while(1)
        {
                z=PORTD;
                PORTB=z+1;
        }
}
```

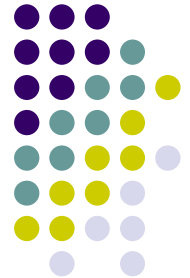
C data types in MPLAB PIC C Hi-TECH compiler



Type	Size (in bits)	Arithmetic Type
bit	1	boolean
char	8	signed or unsigned integer[a]
unsigned char	8	unsigned integer
short	16	signed integer
unsigned short	16	unsigned integer
int	16	signed integer
unsigned int	16	unsigned integer
long	32	signed integer
unsigned long	32	unsigned integer
float	24	real
double	24 or 32[b]	real

a.A char is unsigned by default, and signed if the `PICL -SIGNED_CHAR` option is used.
b.A double defaults to 24-bit, but becomes 32-bit with the `PICL -D32` option.

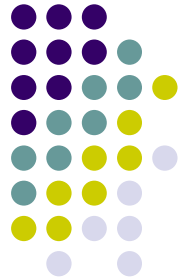
NOTE: See individual C compiler doc for actual data-types and numerical ranges



Radix formats for handling numbers

Radix	Format	Example
binary	0b*number* or 0B*number*	0b10011010
octal	0*number*	0763
decimal	*number*	129
hexadecimal	0x*number* or 0X*number*	0x2F

Preprocessor directives



The preprocessor is automatically invoked as the first step in compiling a program.

#include, #define and __CONFIG() are preprocessor directives.

The #include directive includes the code of external files in the program. Commonly, we include header files that contain function prototypes and definitions i.e.

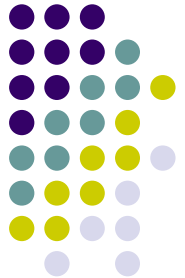
#include <math.h> (it contains many mathematical functions)

#include <stdio.h> (it contains I/O functions like printf and putc) or

#include “my_header_file.h” (for a user defined header file)

(angled brackets <> tell the preprocessor to look into predefined “include” directories for the file. Quotation marks “ ” direct the preprocessor to look into current directory for the file).

Preprocessor directives (continued)



Another category of header files is processor-specific files.
They contain declarations and definitions for SFRs and their bits:

htc.h is a necessary header file that contains declarations and definitions for hardware elements for all PIC16F MCUs. Always begin your code with the directive:

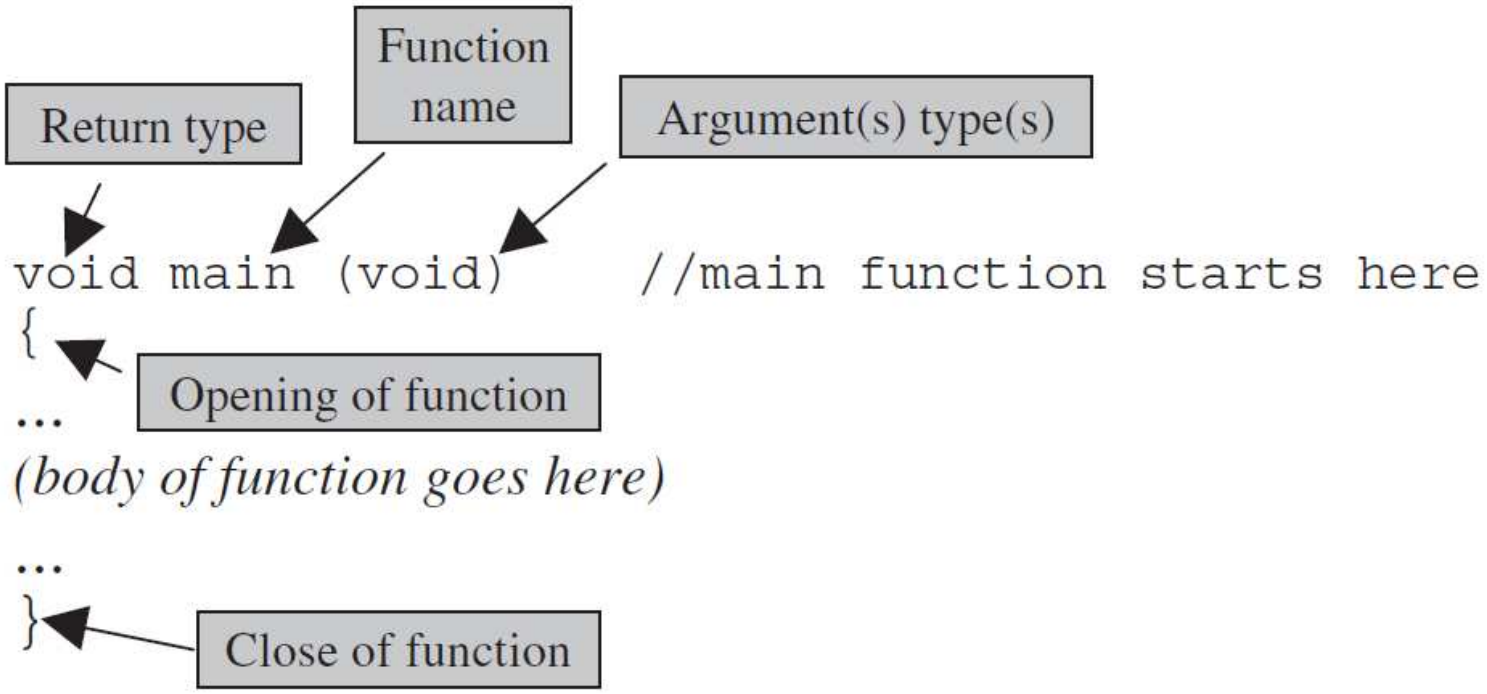
#include <htc.h>

__CONFIG(*arg1, arg2*,…) is a preprocessor macro that writes 16 bits into the configuration word. It takes arguments defined in header file htc.h

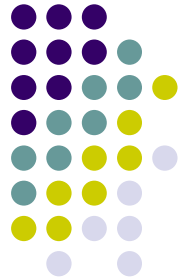
A constant can be declared using the #define directive:
#define <label> value, for example:
#define pi 3.1415926

The C function

C programs are structured from functions. All programs have at least one function, the "main" function.

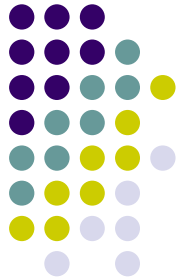


Working with user-defined functions: example



```
/*Example functions with HITECH Compiler for 16F series MCUs.
BLINK THE LED USING A DELAY FUNCTION */
#include <htc.h>
/*CONFIGURATION BITS */
__CONFIG(UNPROTECT & PWRTDIS & WDTDIS & XT & LVPDIS);
//function declarations
void my_delay (unsigned int);      //function prototype
void initialize (void);            //function prototype
//global variable declaration
unsigned int z=0;
//main function
void main(void)
{
        initialize();   //call user function for port initialization
        while(1)
        {
         z=PORTD;                  //Read PORTD
         PORTB=z;                  //Output motive to PORTB
         my_delay(20000);          //call user function for 240ms time delay
         PORTB=~z;                 //Toggle PORTB
         my_delay(20000);
        }
}
```

Working with user functions (example continued)

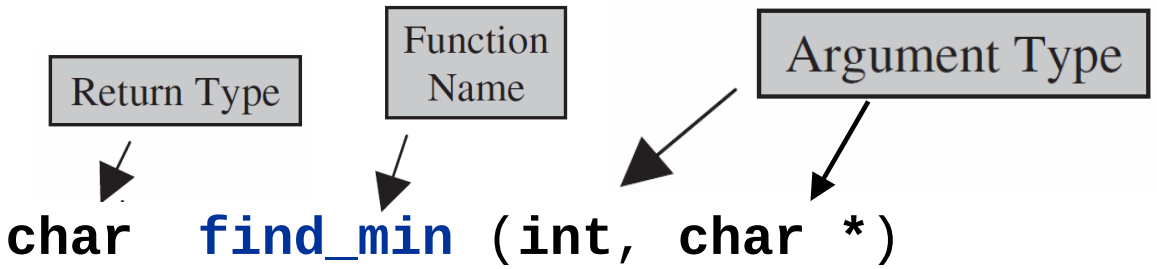


```
//user functions definitions
void initialize(void) //This is a cute way to perform all SFR init in one f
{
        TRISD=0xFF;         /* Set PORTD as input*/
        TRISB=0x00;         /* Set PORTB as output */
        PORTB=0b00000000;   /* Initialize PORTB with zeroes */
}

/*The following function produces approx. 200ms delay on a 4MHz processor*/
//delays are very important for display or button debouncing
void my_delay(unsigned int count_delay)    //on PICC and with fosc=4MHz
{
unsigned int counter=0;         //declaration of local variable
   while(counter<count_delay) //count_delay=3000 is 36msec, 20000 is 240msec
   {
   counter=counter+1;
   }
}
```

More on functions

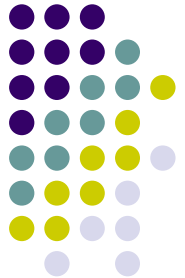
Function prototypes: A declaration which informs the compiler of the type of the function's arguments and its return type:



Function definitions

```
char find_min (char N, char *my_pointer) //find min among array elements
{
for (index=0; index<N; index++)
    {
      if ( *(my_pointer+index)<min)      //compare current element with min
      {
      min = *(my_pointer+index);         //update min
      }
    }
return min;
}
```

More on functions (continued)



How to call function find_min:

```
min = find_min(N, my_pointer);
```

Beside user functions, there are also compiler built-in functions grouped in LIBRARIES

A library is a *collection of functions* grouped for reference and ease of linking. The HI-TECH compiler comes equipped with libraries for software and hardware tasks in the **lib** directory. They can be linked directly into an application using the MPLINK linker. Each function in a collection has an associated header file:

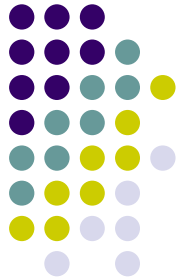
```
#include <math.h>
double cos (double f)
double sqrt (double f)
```

Mainly mathematical, string and i/o operations are supported. In other compilers peripheral tasks are also implemented using built-in functions:

```
#include <adc.h>
void ConvertADC(void) //convert analog sample to digital
```

Just to refresh your memory

…however, if your memory is blank, this is your last chance to fill the gap…
Remember: if you don't find C, then C is going to find you



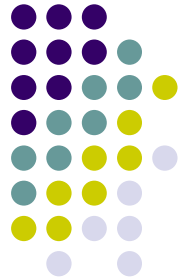
Control structures: for loop, while loop, if

```
for (index=0; index<10; index++)          // for (initialization; conditional_test; increment)
    {
    sum=sum + matrix [index];             //statements;
    }

while (expression)   //This is how we implement the super-loop in embedded systems
    {
    statements;      //you may have conditional break here;
    {

if (*(my_pointer+index2)<min)             // if (expression)
    {
    min=*(my_pointer+index2);             // statements;
    }
```

Just to refresh your memory (continued)

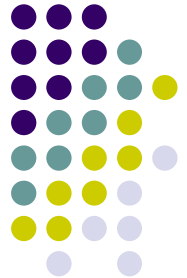


if-else, switch

```
if (expression)
   {
   statements;
   }
else
   {
   statements;
   }
```

```
switch (variable)
{
    case constant1:
        statement(s);
        break;
    case constant2:
        statement(s);
        break;
    case constantN:
        statement(s);
        break;
    default:
        statement(s);
}
```

C and the embedded environment: Working with SFR bits, buttons and diagnostics



Code elements particular to MCU programming are:

1. Processor-specific header files which define mnemonic names on top of SFR memory locations and their bits (see for example file htc.h). SFRs are referred to with their standard names, as defined by Microchip data-books. Example:

```
OPTION = 0b10000111; PORTB=0b01011100; INTCON=0b01000000;
```

SFR bit names depend on the compiler. Hi-Tech compilers name port bits as: **RD0** (meaning `PORTD bit 0`), **RB4** (meaning `PORTB bit 4`) `etc`. Flags and enable bits are bit variables defined with their original names, like **T0IF** or **GIE**.

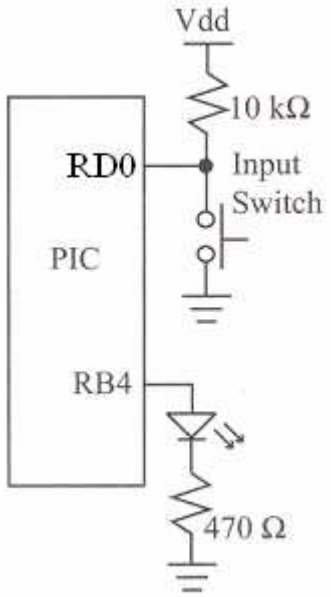
2. Delay functions depend on processor clock and are very important for data display and push-button debouncing. You can make your own or look for a built-in function in lib.

3. Configuration bits are processor specific and are defined using special macros

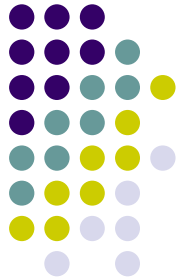
4. Processor peripherals like timers, interrupts, PWMs, ADCs etc. are often controlled by specific hardware-oriented functions providing *some* hardware abstraction, otherwise they have to be operated working immediately with their registers, flags and enable bits, as we do in assembly.

C and the embedded environment (continued): Working with SFR bits and buttons

```
#include <htc.h>
#define _XTAL_FREQ 4000000          //define the XT frequency
//don't forget function prototypes!
unsigned char z=1;                  //variable declarations
void main(void)
{
   initialize(); //call user function for port initialization
   diagnostic(); //call user function for diagnostic test
   while(1)
   {
        RB4=z;
        while(RD0==1)     //pin RD0 is 1 when not pressed
        {
         //Wait for press
        }
   __delay_ms(30);        //debounce using library delay functic
        while(RD0==0)
        {
         //Wait for release
        }
   __delay_ms(30);        //debounce using library delay function
        z=~z;             //TOGGLE RB4 LED!
   }
}
```



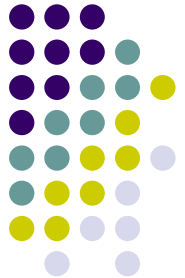
C and the embedded environment (continued): Working with diagnostics



```
//Function diagnostic
//blinks the LEDS of PORTB 10 times
void diagnostic(void)
{
unsigned char id;
   for(id=0; id<10; id++)
   {
   PORTB=0xAA;
   __delay_ms(250);    //delay with user function
   PORTB=0x55;
   __delay_ms(250);    //delay with user function
   }
   PORTB=0x00;
}

void initialize(void) //PORT initialization function
{
        TRISD=0xFF;    /*Set PORTD as input*/
        TRISB=0x00;    /* Set PORTB as output */
        PORTB=0x00;    /*Initialize PORTB with zeroes */
}
```

Working with arrays



An array is a list of related variables of the same data type. Any data type can be used. Array elements are stored in consecutive memory locations.

Array declaration:

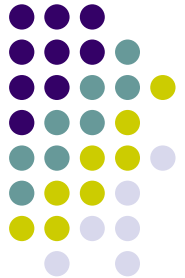
```
unsigned char matrix1[10];
//matrix1 has 10 elements of type char
```

We access array elements using an index or index-variable, starting from 0.

The name of an array is set equal to the address of the initial element.

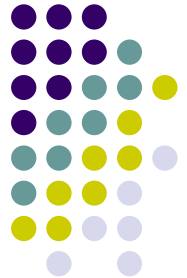
A string is defined as a null-terminated character array.

Working with arrays (example)



```
//SUM MATRIX ELEMENTS
#include <htc.h>                                    //or #include "pic1687x.h" for PIC16F877
/*Set CONFIGURATION BITS in code*/
__CONFIG (UNPROTECT & PWRTDIS & WDTDIS & XT & LVPDIS);
void initialize (void);                 //user function prototype
void diagnostic (void);                 //user function prototype
//variable and array declarations
unsigned int index, sum;
int matrix[10] = {10, 2, 8, 9, 14, 1, 7, 6, 5, 3};
//main function
void main (void)
{
        initialize( );                  //TRISB=0x00, TRISD=0xFF, PORTB=0
        index=0;
        sum=0;
        for (index=0; index<10; index++)
                {
                sum=sum + matrix [index];
                }
                PORTB=sum;              //Output sum
        while(1)  {   //endless loop  }
}
```

Using pointers



Instead of specifying a variable by name, we can specify its address. This address is called a pointer. In other words, a pointer is a memory location (variable) that holds the address of another memory location:

```
my_pointer=&my_variable;//note the use of unary operator &
```

Then, variable **my_pointer** holds the address of variable **my_variable**

Reciprocally, we can use the * operator in order to “dereference” a pointer:

```
my_variable=*my_pointer;
```

***my_pointer** is read as “the value pointed to by **my_pointer**”.

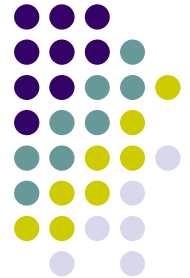
A pointer is declared by the data type it points to:

```
int *my_pointer;
```

indicates that **my_pointer** points to a variable of type **int.**

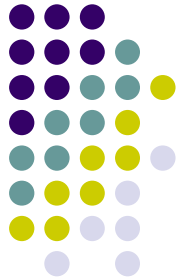
```
char *my_pointer=&my_array[0];
```

Find the minimum among matrix elements



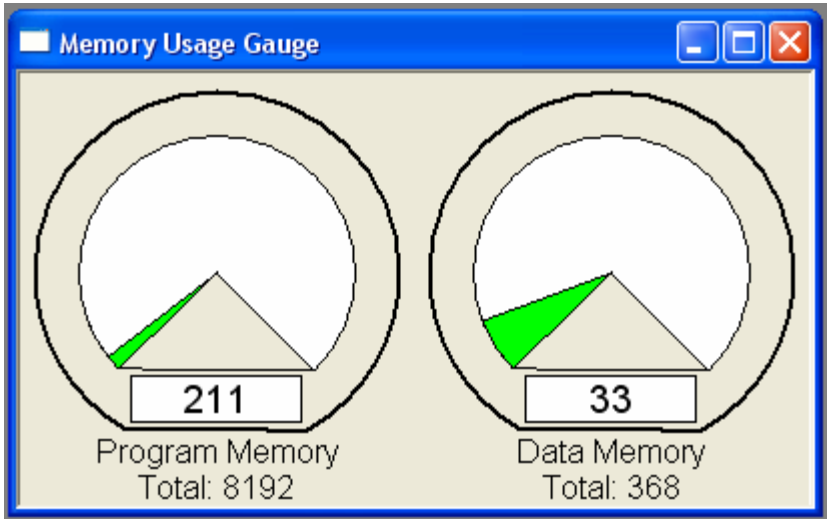
```
//Fill array with values using PORTD and find minimum among matrix elements
#include <htc.h>
//Define constants
#define _XTAL_FREQ 4000000  //define the XT frequency for __delay_ms()
#define _Number 5 //define number of elements in the matrix
__CONFIG(UNPROTECT & PWRTEN & WDTDIS & XT & LVPDIS); /*Set CONFIGURATION BITS*/
//user function prototypes
void initialize (void);     //Set PORTB as output, PORTD as Input and RC0 as input
void diagnostic (void);     //Same as in previous code
void input_vals (char);
char find_min (char, char *);
//variable declarations
char my_array[_Number];
char min=255;               //min initially acquires the maximum value possible
unsigned char N=_Number, index=0;
char *my_pointer=&my_array[0];
void main(void) //main function
{
        initialize();               //call user function for port initialization
        diagnostic();               //call user function for diagnostic test
        input_vals(N);              //call user function to input values
        min=find_min(N, my_pointer);         //call user function for min value
        PORTB=min;                  //output min value
        while(1)
        {//endless loop. Execution is trapped here
        }
}
```

Important user functions: input_vals and find_min (continued)



```
void input_vals (char N)                //function to input values from PORTD
{
unsigned char index1=0;
      for (index1=0; index1<N; index1++)
            {
                  while(RC0==0)
                  {           //Wait until button pressed
                  }
                  __delay_ms(30);                  //debounce
                  my_array[index1]=PORTD;          //Transfer PORTD to array
                  PORTB=my_array[index1];          //Show input value on PORTB
                  while(RC0==1)
                  {           //Wait until button released
                  }
                  __delay_ms(30);                  //debounce
            }
}
//function to calculate the minimum value: pass pointer to function
char find_min (char N, char *my_pointer)
{
unsigned char index2=0;
for(index2=0; index2<N; index2++)
      {
            if (*(my_pointer+index2)<min)       //compare current element with min
            {
            min=*(my_pointer+index2);           //update min
            }
      }
return min;
}
```

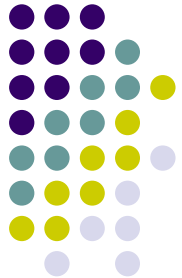
Build the project and View-Memory Usage Gauge



The gauge presents program and data memory usage for our code

When the compiler is best optimized, machine code is as compact as possible. This is a good reason to give some money and buy a compiler (~200€).

(You may even find it entertaining to stage compiler games!)



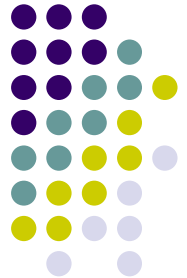
Surprise! Project No 6:

Write a C application that performs the following successive tasks:

1. Use DIP switches on PORTB in order to input ten 8-bit integers into an array `my_values[10]`. Use a push-button on RC0 to trigger each reading. Write code to debounce, each time the button is pushed or released.
2. Sort the array of values starting from minimum.
3. Display the values in serial order using again RC0 push-button to trigger the display of successive array elements.

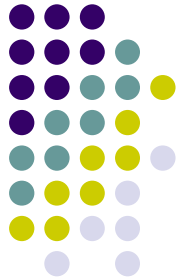
Along with the code please hand-in a circuit design of the application.

Working in C with timers - Programming Timer0



```
#include <htc.h>
/*Set CONFIGURATION BITS in code*/
__CONFIG(UNPROTECT & PWRTEN & WDTDIS & XT & LVPDIS);
int Count = 0; //variable declarations
//main function
void main()
{
   TRISB = 0b00000000;    // PORTB is output
   PORTB = 0b00000000;    // clear PORTB
   TMR0 = 0;              // Intialize TMR0 with 0
   OPTION = 0b10000111;   // select internal instruction clock
                          // connect prescaler,
                          // Prescaler = 1:256
   while(1)
   {                      // (256 x 256)/1MHz = 65536µs
        while(!T0IF);     // delay until overflow (upon OVF T0IF=0, TMR0=0)
        T0IF = 0;         // Clear flag
        Count++;
        if(Count == 15)
        {                 // 65536 x 15 = 0,983s
        Count = 0;
        PORTB = ~PORTB;   // Toggle bits of PORTB
        }
   }
}
```

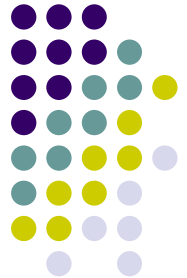
Working in C with interrupts –a typical main function



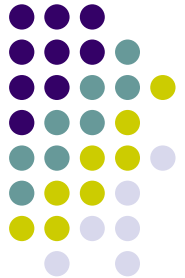
```
//This is a test program for interrupts in C language.
//Interrupt is produced by TIMER0
#include <htc.h>
/*Set CONFIGURATION BITS in code*/
__CONFIG(UNPROTECT & PWRTDIS & WDTDIS & XT & LVPDIS);

void initialize (void);   //function prototype
unsigned char counter;    //defines a global variable
void main (void)          //main function
{
   counter=0;
   initialize( );         //interrupt initialization function
   while(1)               //endless loop
   {
        PORTB=counter;    //count interrupts on PORTB
   }
}
```

Initialization of interrupts and a simple ISR

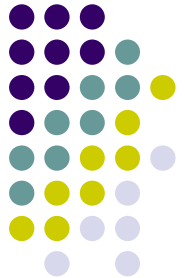


```
void initialize (void)
{
   TRISB=0b00000000;      // PORTB output
   OPTION=0b00001000;     // connect prescaler, divide by 2
   T0IE=1;                // Enable Timer0 interrupts
   GIE=1;                 // or ei();
   TMR0=0;                // for simulation purposes
}
//Interrupt service routine
interrupt Timer0_ISR(void)
{
   counter++;             // increment counter upon interrupt
   if(T0IE && T0IF) //If source of interrupt is Timer0 AND interrupt
   occurred
   {
        counter++;        // increment counter
        T0IF=0;           // clear Timer0 interrupt flag
        TMR0=0;           // Reload TMR0
   }
}
```



Surprise again! Project No 7:

Write code for an application that receives interrupts from the following two sources: a. from pin INT (RB0) and b. from Timer0.
Timer0 produces an interrupt with frequency 25Hz. Consider external clock frequency 4MHz.
Upon timer0 interrupt an led connected to RD0 toggles.
Upon INT interrupt an 8-bit counter is incremented and its content is displayed on PORTC LEDs.



(Absolutely) required reading

1. Chapters 14, 15, 16 of *Designing Embedded Systems with PIC microcontrollers by Tim Wilmshurst.*
2. Anything else you need in order to cope with the material presented in this lecture. For example, look at the literature given in slide No 4.