# ECE-218 Embedded Microcontroller Projects

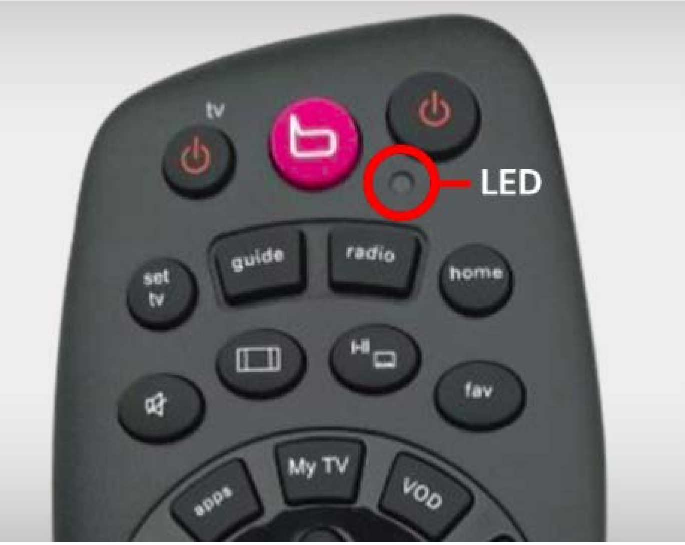
Background for Exercise 1

MPLAB X IDE and C

LED Blink, Button Toggle, State Machine

# Purpose

- LEDs as indicators
- Switch inputs for user control
- State machines – manage complexity of time ordered systems



# Overview of Exercise 1

- Integrated Development Environment (IDE)
  - Creating new project, debug mode
- Hardware:
  - LED with current limiting resistor
  - Pushbutton switch with pullup resistor
- Programs:
  - LED blink program (on Nexus)
  - Blink with button press (modify blink program)
  - State machine (fill in template from Nexus)

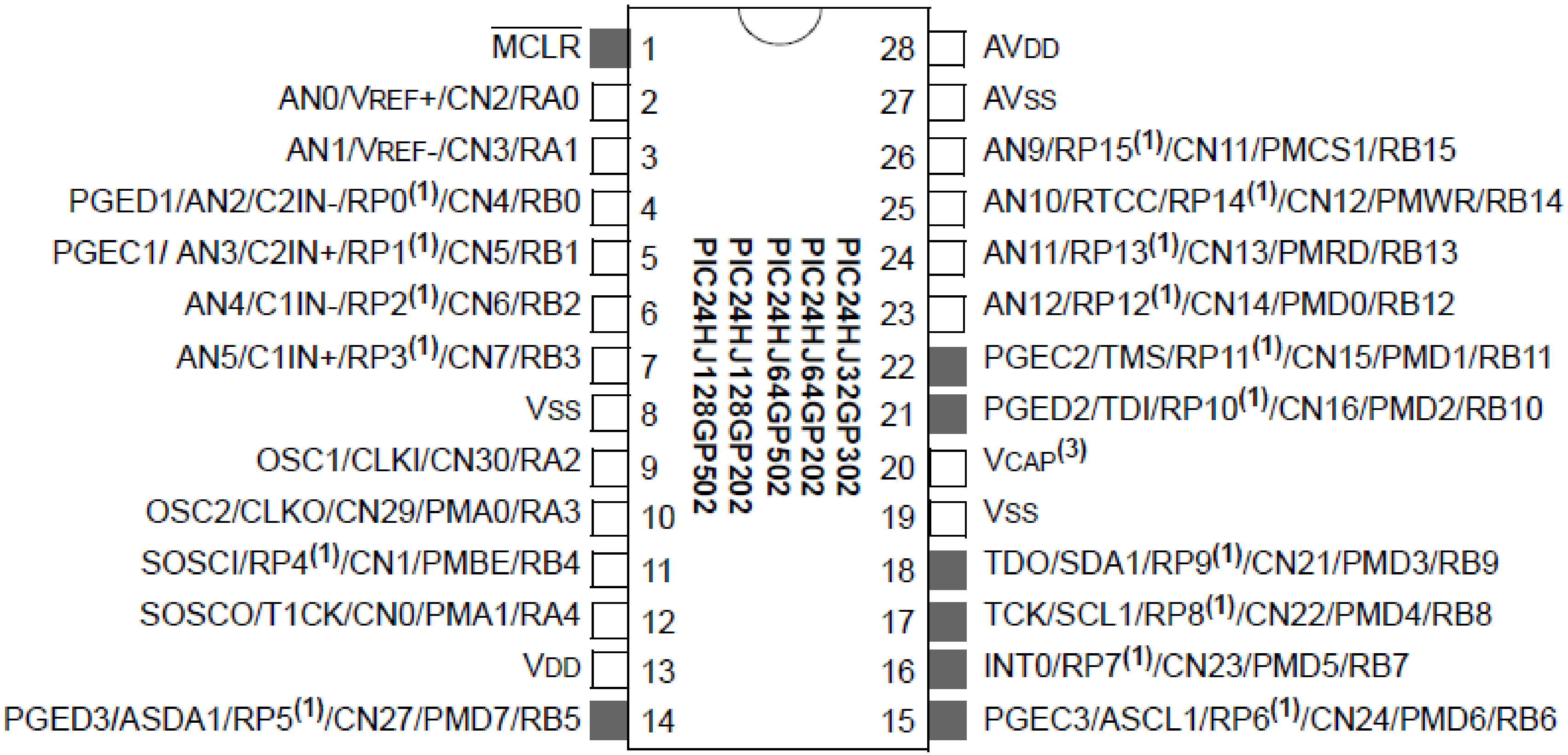
# New Interfacing Concepts

- PIC24 I/O Ports A and B
  - available ports on our 28-pin PIC24
  - how to configure them to be digital inputs/outputs
- Digital outputs – LED example
- Digital input – pushbutton switch example

# I/O Ports on PIC24

- For our particular chip, there are two 16-bit ports, but not all port pins are available:
  - PORTA (5 bits)
    - RA0, RA1, RA2, RA3, RA4
  - PORTB (16 bits)
    - RB0, RB1, RB2, RB3, RB4, RB5, ….. RB15

# 28 Pins – MANY are shared


MCLR
1
AN0/VREF+/CN2/RA0
2
AN1/VREF-/CN3/RA1
3
PGED1/AN2/C2IN-/RP0(1)/CN4/RB0
4
PGEC1/ AN3/C2IN+/RP1(1)/CN5/RB1
5
AN4/C1IN-/RP2(1)/CN6/RB2
6
AN5/C1IN+/RP3(1)/CN7/RB3
7
VSS
8
OSC1/CLKI/CN30/RA2
9
OSC2/CLKO/CN29/PMA0/RA3
10
SOSCI/RP4(1)/CN1/PMBE/RB4
11
SOSCO/T1CK/CN0/PMA1/RA4
12
VDD
13
PGED3/ASDA1/RP5(1)/CN27/PMD7/RB5
14
28
AVDD
27
AVSS
26
AN9/RP15(1)/CN11/PMCS1/RB15
25
AN10/RTCC/RP14(1)/CN12/PMWR/RB14
24
AN11/RP13(1)/CN13/PMRD/RB13
23
AN12/RP12(1)/CN14/PMD0/RB12
22
PGEC2/TMS/RP11(1)/CN15/PMD1/RB11
21
PGED2/TDI/RP10(1)/CN16/PMD2/RB10
20
VCAP(3)
19
VSS
18
TDO/SDA1/RP9(1)/CN21/PMD3/RB9
17
TCK/SCL1/RP8(1)/CN22/PMD4/RB8
16
INT0/RP7(1)/CN23/PMD5/RB7
15
PGEC3/ASCL1/RP6(1)/CN24/PMD6/RB6
PIC24HJ32GP302
PIC24HJ64GP202
PIC24HJ64GP502
PIC24HJ128GP202
PIC24HJ128GP502

Pinout from Datasheet. Priority read left to right

# Port Special Function Registers

- For each port x (x = A or B) there are 3 special registers:
  - TRISx – used to set pin as Input (1) or Output(0)

```
TRISA = 0x0000;  //set PortA pins to outputs
_TRISA1 = 0;  //set Port A, bit 1 (pin 3) as output
```

  - PORTx – used for reading an input port

```
X = PORTB;  //set variable X to PortB values
Y = _RB0;     //set variable Y to PortB pin 0 value.
```

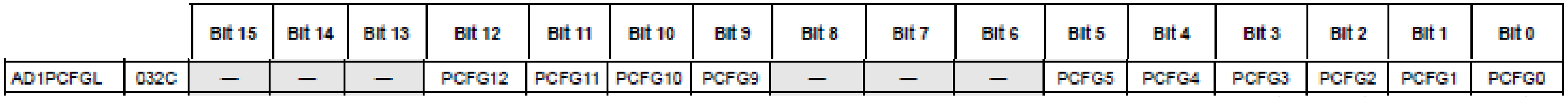
  - LATx – used for writing to output port

```
LATA = 0xFFFF;  //set all PortA pins high
_LATA1 = !_LATA1;  //toggle bit 1 of Port A
```

# Another Port Special Function Register

## AD1PCFGL - ADC1 Port ConFiGuration register Low

- Sets shared analog/digital pins to:
  - analog (0) (default) or
  - digital (1)



| | | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AD1PCFGL | 032C | — | — | — | PCFG12 | PCFG11 | PCFG10 | PCFG9 | — | — | — | PCFG5 | PCFG4 | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
| | | | | | AN12 | AN11 | AN10 | AN9 | | | | AN5 | AN4 | AN3 | AN2 | AN1 | AN0 |

```
AD1PCFGL = 0xffff;                //make all shared analog/digital pins digital
AD1PCFGL = 0x00002; //set RA1 to digital (and all other shared pins analog)
_PCFG0 = 1; //set RA0 to digital (and leave others unchanged)
```

# Accessing Individual Port Pins

- Two ways to specify PIC24 individual bits in C:

  Example: bit 0 of Port B

  1. _RB0
  2. PORTBbits.RB0

  Example: bit 1 of TRISB

  1. _TRISB1
  2. TRISBbits.TRISB0

  Example: bit 2 of LATA

  1. _LATA2
  2. LATAbits.LATA2

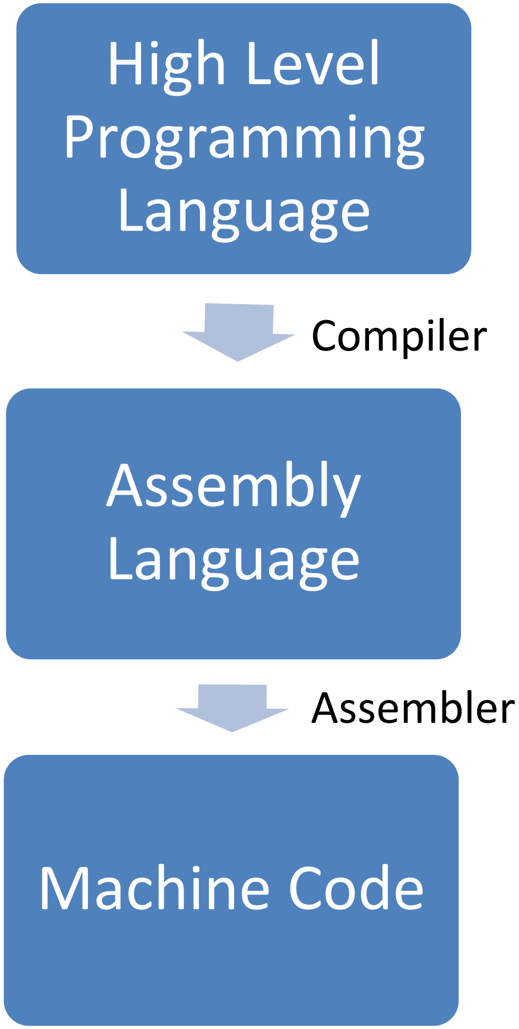
# New Software Concepts

- What is an IDE?
- Steps from C to machine code
- Some C Programming basics
  - Compiler directives
  - Functions
  - Assignment statements
  - While loops
  - If statements
  - Logical “not” operator
  - Switch statement

# Integrated Development Environment

- For Microcontrollers
  - **MPLAB X IDE**
  - Arduino IDE
  - CodeWarrier IDE
  - Silicon Labs IDE
  - Lots more…
- For FPGAs and other programmable logic
  - Quartus
  - Xilinx
  - Others…
- For high level programming
  - Python IDE (Eclipse, etc)
  - Matlab
  - Lots more….

# Typical Development Process



High Level Programming Language

Compiler

Assembly Language

Assembler

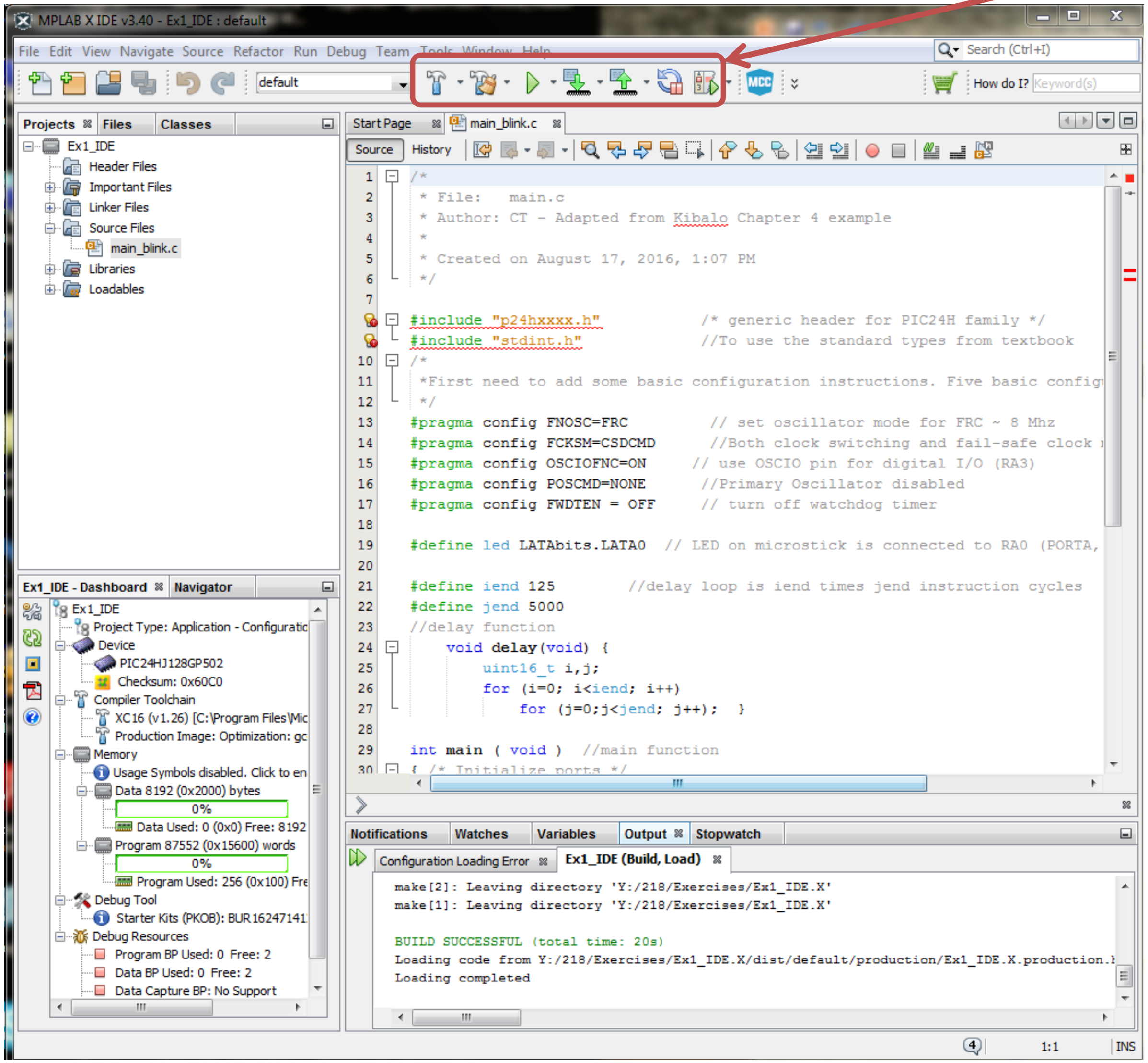
Machine Code

- programmer productivity
- code clarity
- code portability

- Special purpose arithmetic functions
- Performance critical functions
- Special processor-dependent features

# MPLAB X IDE

Several tools for converting .c files to machine code



Ex1_IDE
Project Type: Application - Configuratio
Device
PIC24HJ128GP502
Checksum: 0x60C0
Compiler Toolchain
XC16 (v1.26) [C:\Program Files\Mic
Production Image: Optimization: gc
Memory
Usage Symbols disabled. Click to en
Data 8192 (0x2000) bytes
0%
Data Used: 0 (0x0) Free: 8192
Program 87552 (0x15600) words
0%
Program Used: 256 (0x100) Fre
Debug Tool
Starter Kits (PKOB): BUR16247141
Debug Resources
Program BP Used: 0  Free: 2
Data BP Used: 0  Free: 2
Data Capture BP: No Support

```
/*
 * File:   main.c
 * Author: CT - Adapted from Kibalo Chapter 4 example
 *
 * Created on August 17, 2016, 1:07 PM
 */

#include "p24hxxxx.h"              /* generic header for PIC24H family */
#include "stdint.h"                //To use the standard types from textbook
/*
 *First need to add some basic configuration instructions. Five basic config
 */
#pragma config FNOSC=FRC          // set oscillator mode for FRC ~ 8 Mhz
#pragma config FCKSM=CSDCMD       //Both clock switching and fail-safe clock 1
#pragma config OSCIOFNC=ON      // use OSCIO pin for digital I/O (RA3)
#pragma config POSCMD=NONE       //Primary Oscillator disabled
#pragma config FWDTEN = OFF      // turn off watchdog timer

#define led LATAbits.LATA0  // LED on microstick is connected to RA0 (PORTA,

#define iend 125        //delay loop is iend times jend instruction cycles
#define jend 5000
//delay function
    void delay(void) {
        uint16_t i,j;
        for (i=0; i<iend; i++)
            for (j=0;j<jend; j++);  }

int main ( void )  //main function
{ /* Initialize ports */
```

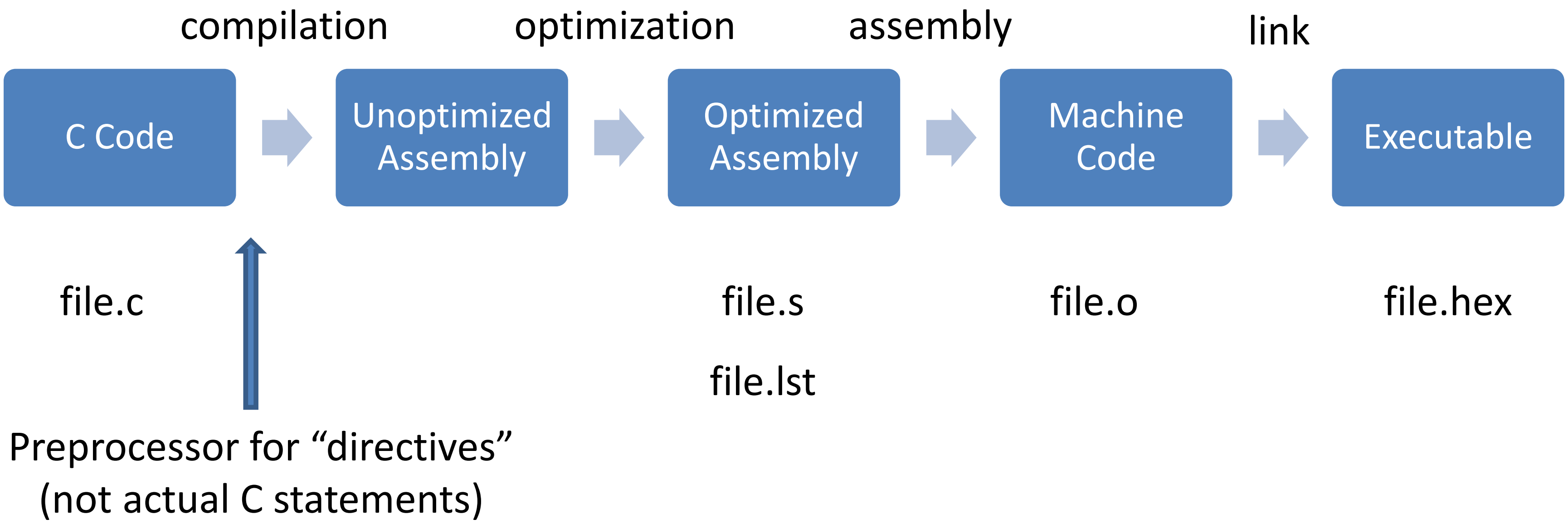
```
make[2]: Leaving directory 'Y:/218/Exercises/Ex1_IDE.X'
make[1]: Leaving directory 'Y:/218/Exercises/Ex1_IDE.X'

BUILD SUCCESSFUL (total time: 20s)
Loading code from Y:/218/Exercises/Ex1_IDE.X/dist/default/production/Ex1_IDE.X.production.h
Loading completed
```

# C Programming Language

- Originally developed to provide higher level language for simple processors
- Not intended for complex data structures used by object-oriented languages
- One of the most popular embedded systems development languages
- Not going to cover ALL the C language
- Topics covered as they arise in text and exercises.

# Detailed Compilation – PIC24



compilation

optimization

assembly

link

C Code

Unoptimized Assembly

Optimized Assembly

Machine Code

Executable

file.c

file.s

file.lst

file.o

file.hex

Preprocessor for “directives” (not actual C statements)

# C for Exercise 1

**Pre-processor Code**

- Comments (ignored)

  /*  multi-line comment */

  // single line comment
- Compiler Directives
  - #include
  - #define

**C language Code**

- Functions
  - main()
  - configClock()
  - DELAY_MS()
  - others…
- Statements
  - assignments
  - while loop
  - if statement
  - others…

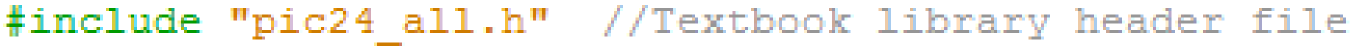
# Files

- file.h   “Header” files
  - C declarations
  - macro definitions
- There are “system” header files and “user” header files.

- file.c   “Source” files

# #include

- Used to include header files in your source code.



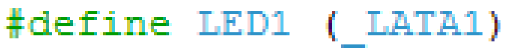
```
#include "pic24_all.h"  //Textbook library header file
```

- The textbook header files call some system header files
- Later we will create our own header files

# #define

- Allows definitions of constants and macros
- Used for defining constants that represent numbers, strings, or expressions.
- Not for defining variables.

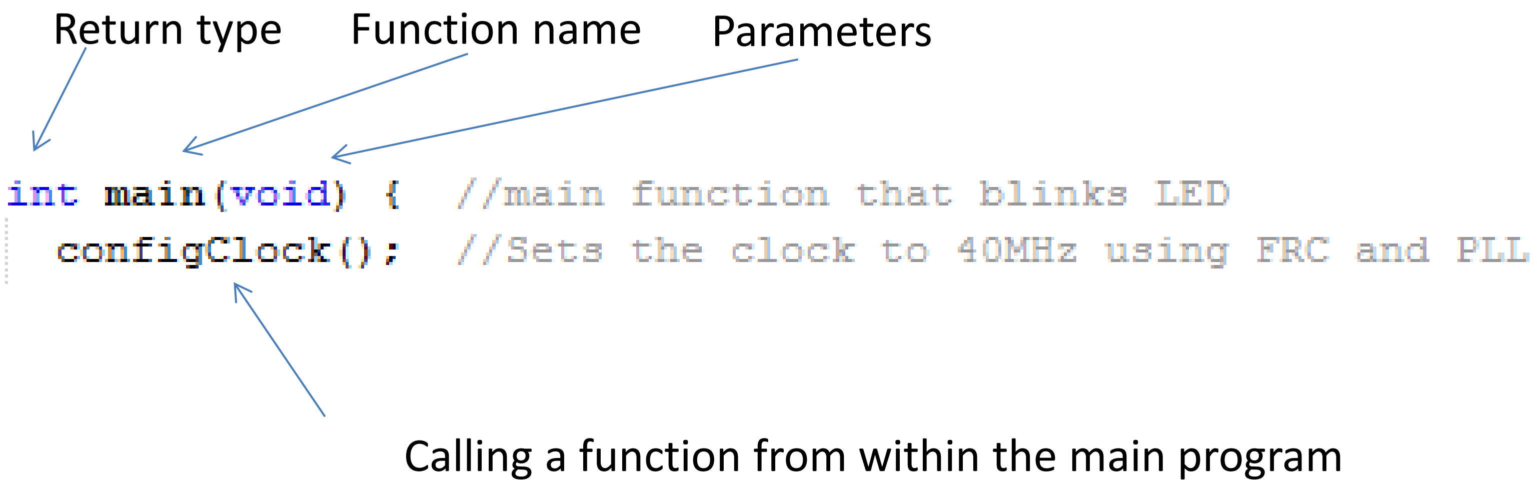
Defining **LED1** to be a string, which corresponds to a register port latch pin on PIC24.



```
#define LED1 (_LATA1)
```

# Functions

- Functions are sequences of statements that accomplish a task.
- All C programs have at least one function, main().



Return type

Function name

Parameters

```
int main(void) {   //main function that blinks LED
  configClock();   //Sets the clock to 40MHz using FRC and PLL
```

Calling a function from within the main program

# Functions from Textbook Library

**External library components**

- Development support
  - A single header file to include all the files below, in pic24_all.h
  - Delay routines in pic24_delay.h → DELAY_MS(250);
  - Setup of IO ports in pic24_ports.h
  - Unit testing / assertions, in pic24_unittest.h
  - Miscellaneous utilities in pic24_util.h
  - Processor-specific information in pic24_chip.h
  - Debug/data transfer abilities in dataXfer.h
  - A byte-wise access to multibyte data in pic24_unions.h
- Peripheral support
  - Analog to digital converter routines, in pic24_adc.h
  - Direct memory access (DMA) routines, in pic24_dma.h
  - Enhanced controller area network (ECAN) routines, in pic24_ecan.h
  - Read/write of FLASH, in pic24_flash.h
  - Setup of timers, in pic24_timer.h
  - Serial communication support, in pic24_serial.h
  - I$^2$C support, in pic24_i2c.h
  - SPI support, in pic24_spi.h

**Internal library components**

- User-configurable library settings, in pic24_libconfig.h
- Clock configuration, in pic24_clockfreq.h and pic24_clockfreq.c → configClock();
- Configuration bit settings, in pic24_configbits.c
- UART support in pic24_uart.h and pic24_uart.c, which is typically called by routines in pic24_serial.h.

**Link to library and documentation on Nexus and lab handout**

# Statements

- Similar to statements in other programming languages.
- Three types of statements:
  - expression statements, assign value to variable

    Example from our program: LED1 = !LED1;
  - compound statements, occur in curly brackets:

    Example from our program:

    ```
    {
     DELAY_MS(250);
     LED1 = !LED1;
    }
    ```
  - control statements, impact program flow

    Example – while loop

    ```
    while (1) {
      DELAY_MS(250);
      LED1 = !LED1;
    }
    ```

# C Data Types

- All variables must be declared, and have a “type”.
- Data type for Exercise 1:
  - “Enumerated” type: user-defined type used to assign names to integral constants

**enum** flag { const1, const2, ..., constN } var;

name of enumerated type

variable of type flag

//Define variable state, of type “States”, with possible values init, R0, P0, R1, P1.

**enum** States {init, R0, P0, R1, P1} state;

# C vs Python

```
if (condition) {
statement(s) will execute if condition is true
} else {
statement(s) will execute if condition is false
}
```

```
while(condition) {
 statement(s) will execute if condition is true
}
```

```
if condition :
    indentedStatementBlockifTrue
else:
    indentedStatementBlockifFalse
```

```
while condition:
    indentedStatementBlockifTrue
```

# Switch – Case Statement in C

```
switch(expression) {
case constant-expression :
statement(s);
break; /* optional */
case constant-expression :
statement(s);
break; /* optional */
/* you can have any number of case
statements */
default : /* Optional */
statement(s);
}
```

- The **expression** used in a **switch** statement must have an integral or enumerated type.
- Each **case** is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

# C Relational Operators (same as Python)

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) |

https://www.tutorialspoint.com/cprogramming/c_operators.htm

# Commenting C Code

```
// to begin one-line comments (anywhere on line)
/* multi-line
     comments  - C ignores white space and indentations  */
```

- Add comments that will be helpful to anyone trying to understand your code
  - Include specific details
  - Do not simply explain what the instruction/segment does, but <u>why it is included</u>

**BAD - NOT helpful**

```
AD1PCFGL = 0x00002;;  //Initialize AD1PCFGL
```

```
AD1PCFGL = 0x00002;  //set RA1 (pin3) to digital and all others analog
```

**GOOD – Has intent and specific details – pin number is important**