A Very Quick Introduction to C- Coding

Dr. Farahmand

Updated: 3/14/19

The C Compiler

- Programming in C-Language greatly reduces development time.
- C is NOT as efficient as assembly
  - A *good* assembly programmer can *usually* do better than the compiler, no matter what the optimization level – C WILL use more memory

C Compiler (Eclipse, Keil, XC8)

A compiler converts a high-level language program to machine instructions for the target processor

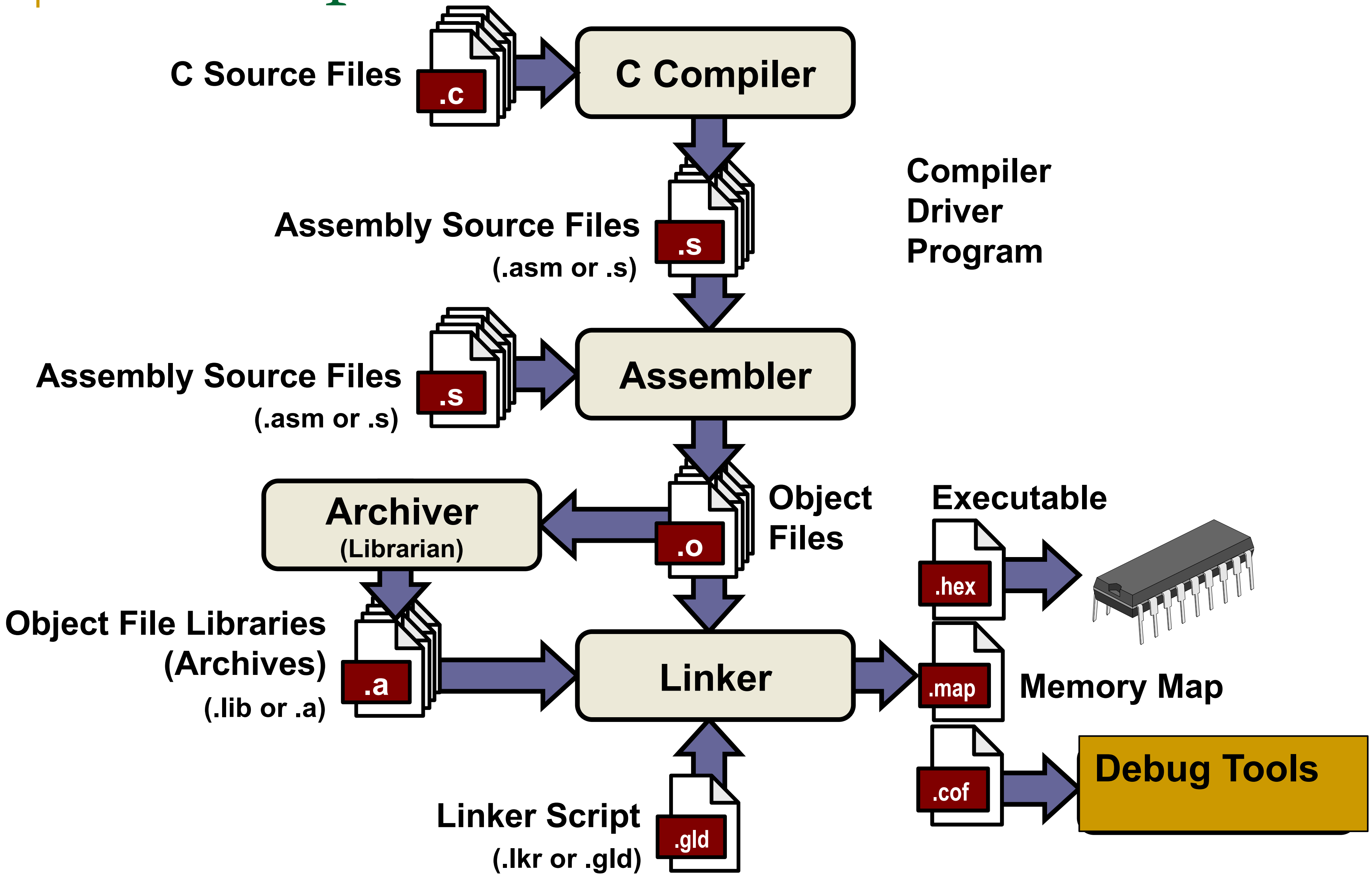
A cross-compiler is a compiler that runs on a processor (usually a PC) that is different from the target processor

Most embedded systems are now programmed using the C/C++ language

The C18/XC8 Compiler

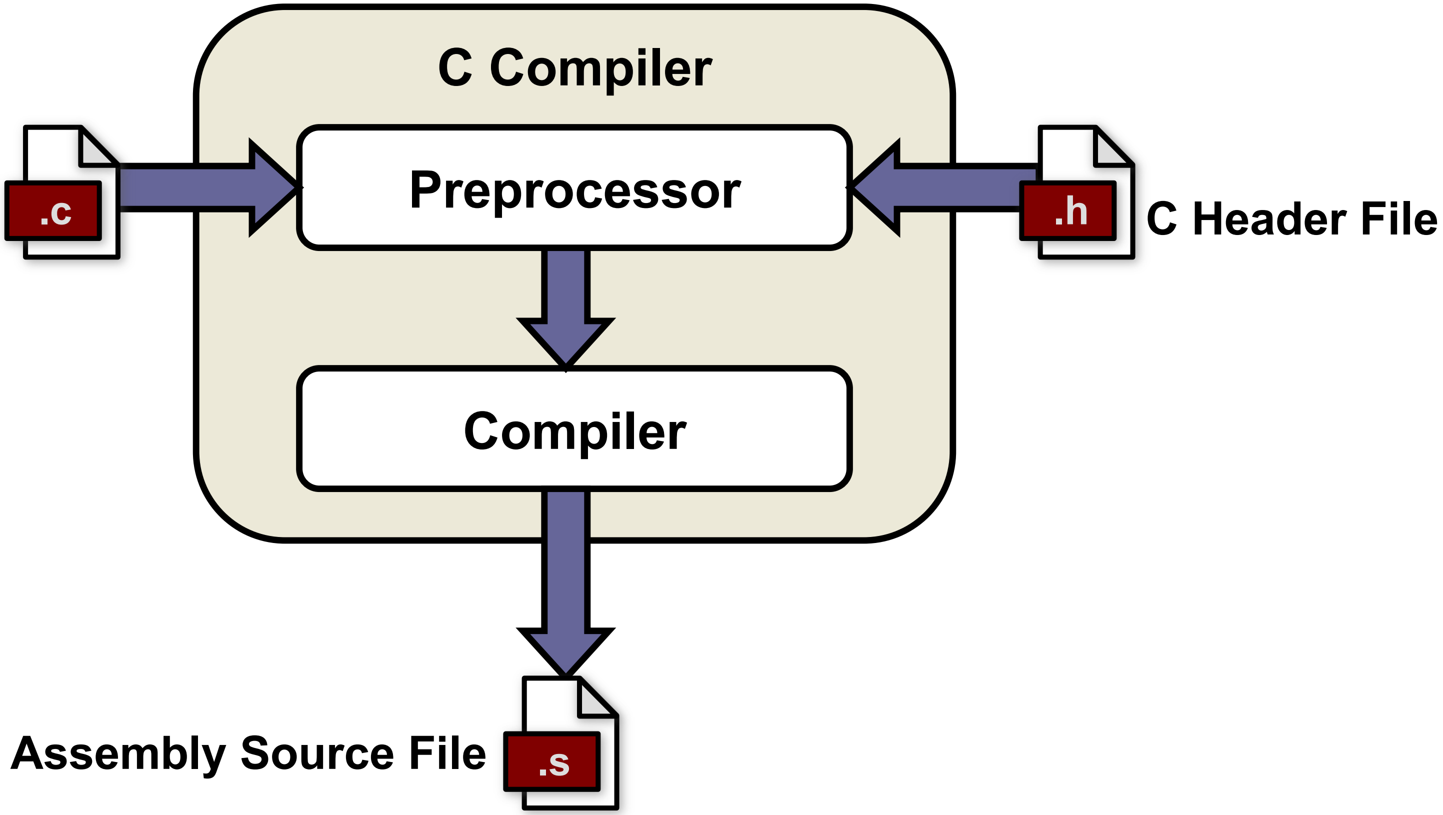
- Mixed language programming using C-language with assembly language is supported by the Keil
  - Assembly blocks are surrounded with at _asm and a _endasm directives to the C18/XC8 compiler.
  - Assembly code can also be located in a separate asm file
    - Example: asm("MOVLW 0x1F");
- The compiler normally operates on 8-bit bytes (char), 16-bit integers (int), and 32-bit floating-point (float) numbers.
- In the case of the PIC, 8-bit data should be used before resorting to 16-bit data.
  - Floats should only be used when absolutely necessary.

Development Tools Data Flow



Cof: Common object file format

Development Tools Data Flow



C Runtime Environment

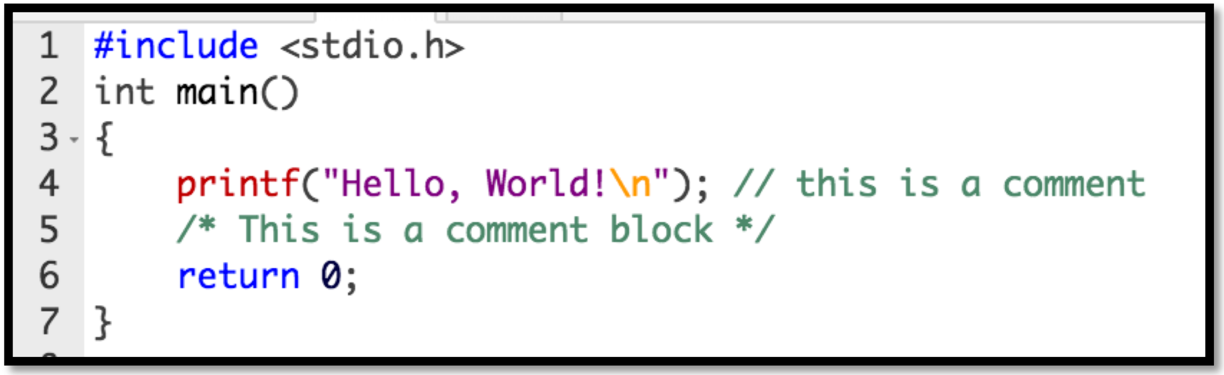
- C Compiler sets up a runtime environment
  - Allocates space for stack
  - Initializes stack pointer
  - Copies values from Flash/ROM to variables in RAM that were declared with initial values
  - Clears uninitialized RAM
  - Disables all interrupts
  - Calls main() function (where your code starts)

So, What is C?

- High-level general purpose language
- First implemented in 1972
- UNIX OS is written in C
- UNIX machines use gcc compilers
- To install gcc:
  - Windows – use MINGW.org
  - MAC – embedded in xcode ($gg –v)
- We use online compilers!

A Simple Program

- All statements are terminated using “statement terminator “;”
- Comments are after // or within /* blab blab */
- Variables can be _MyVariable, MyVariable, etc.
- There are may keywords: else, if, float, etc.

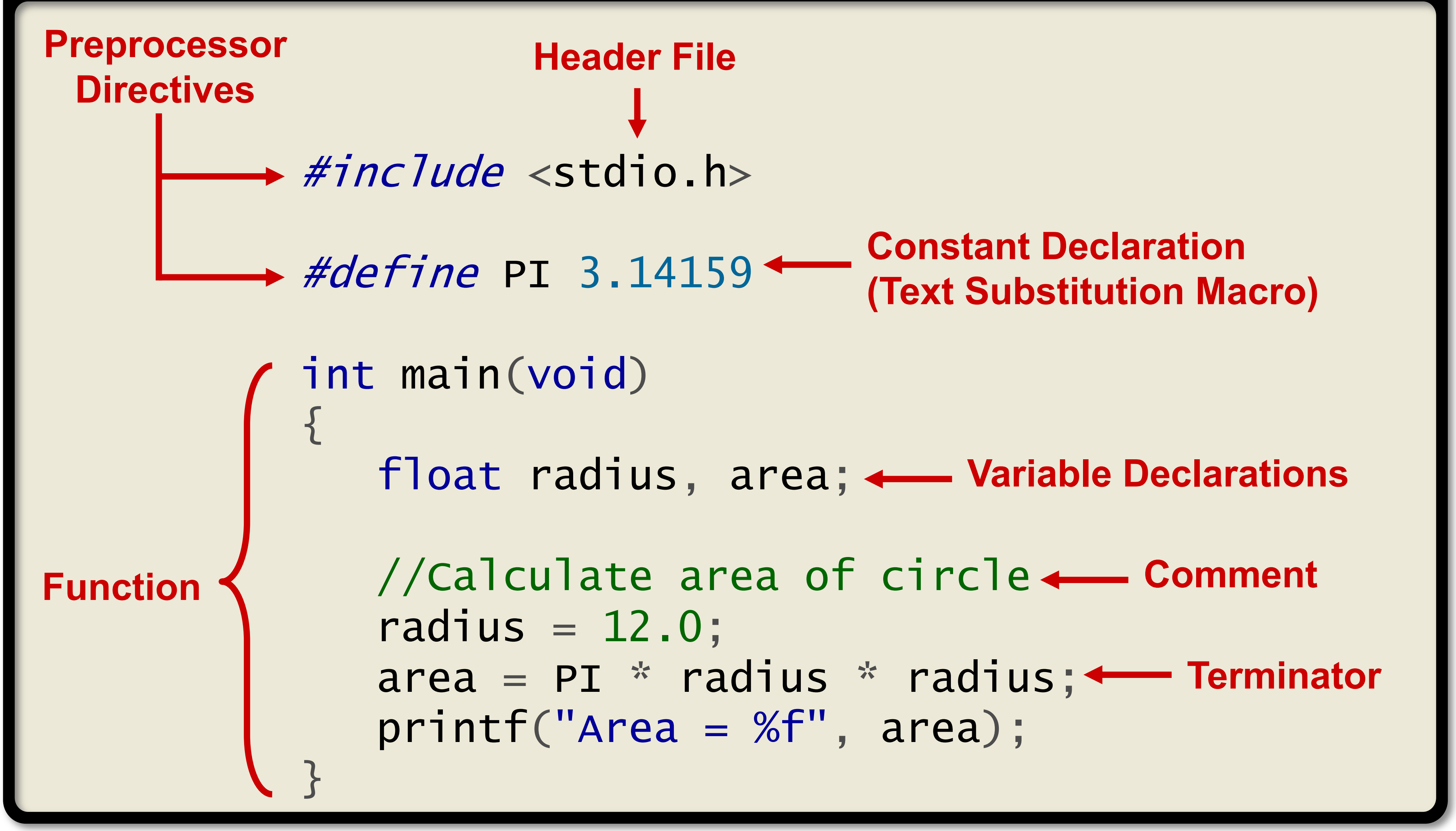


```
#include <stdio.h>
int main()
{
    printf("Hello, World!\n"); // this is a comment
    /* This is a comment block */
    return 0;
}
```

Fundamentals of C

Another Simple C Program

Example



```
#include <stdio.h>

#define PI 3.14159

int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```

Preprocessor Directives

Header File

Constant Declaration (Text Substitution Macro)

Function

Variable Declarations

Comment

Terminator

Comments

Two kinds of comments may be used:

Block Comment

/* This is a comment */

Single Line Comment

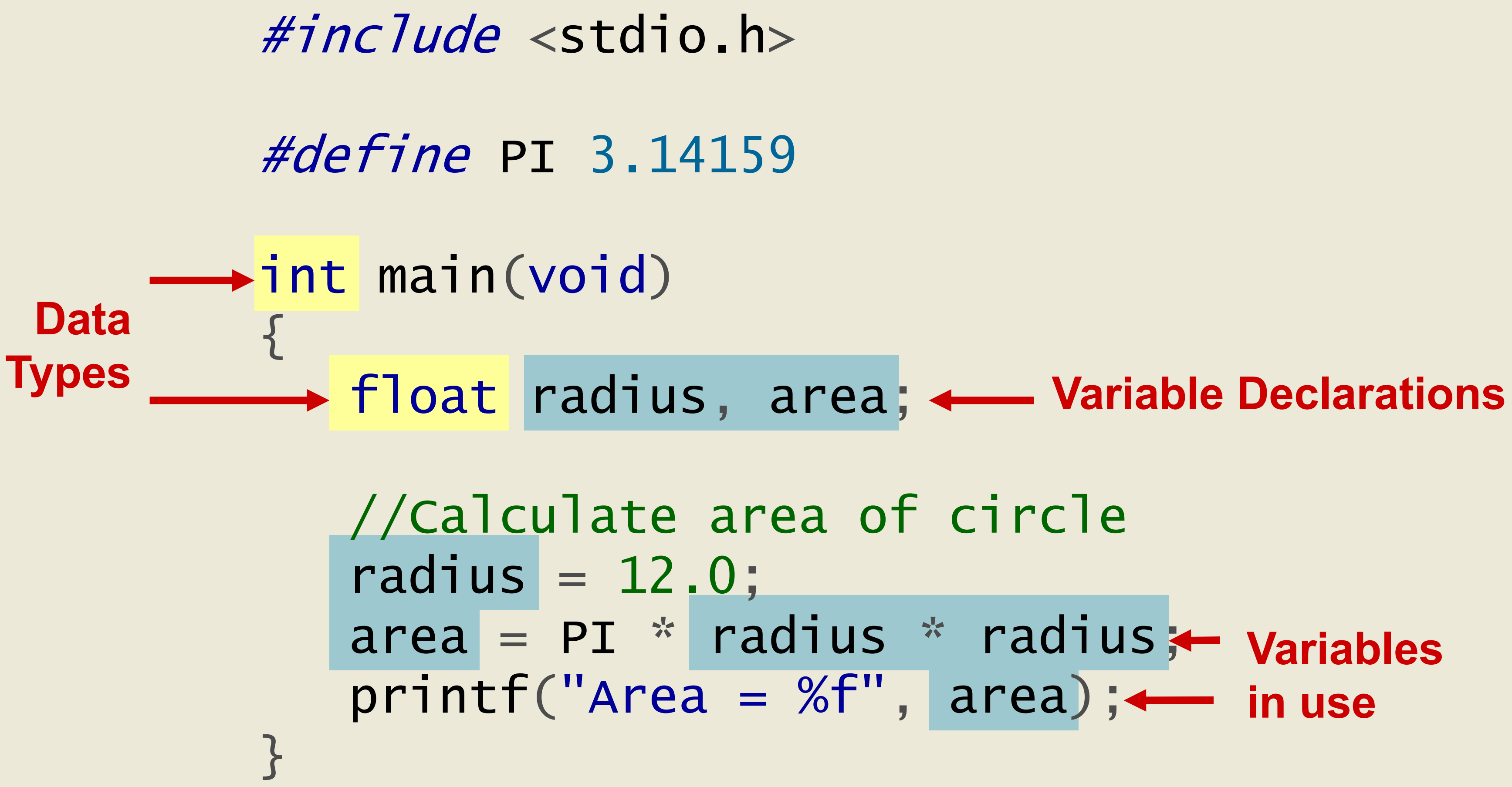
// This is also a comment

```
/********************************************************
* Program: hello.c
* Author:  A good man
********************************************************/
#include <stdio.h>

/* Function: main() */
int main(void)
{
  printf(“Hello, world!\n”); /* Display “Hello, world!” */
}
```

Variables and Data Types

A Simple C Program

Example



```
#include <stdio.h>

#define PI 3.14159

int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```

Data Types

Variable Declarations

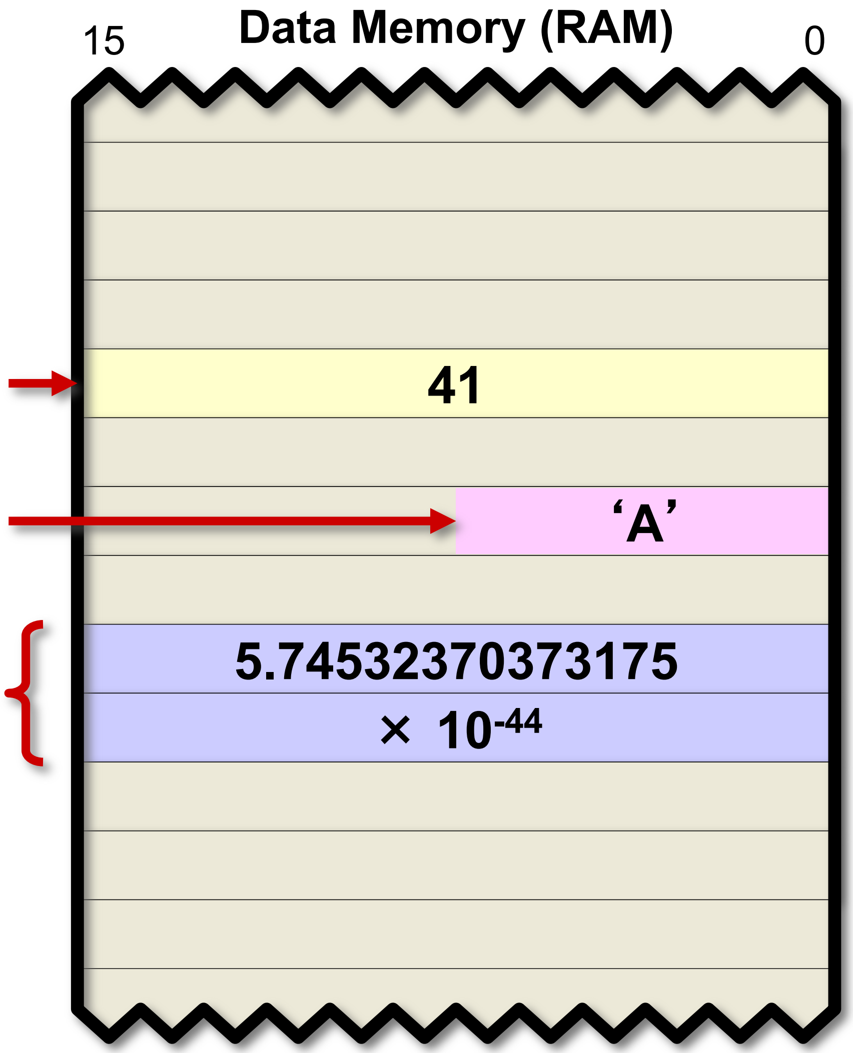
Variables in use

Variables

Variables are names for storage locations in memory

Variable declarations consist of a unique identifier (name)…

```
int warp_factor;
char first_letter;
float length;
```



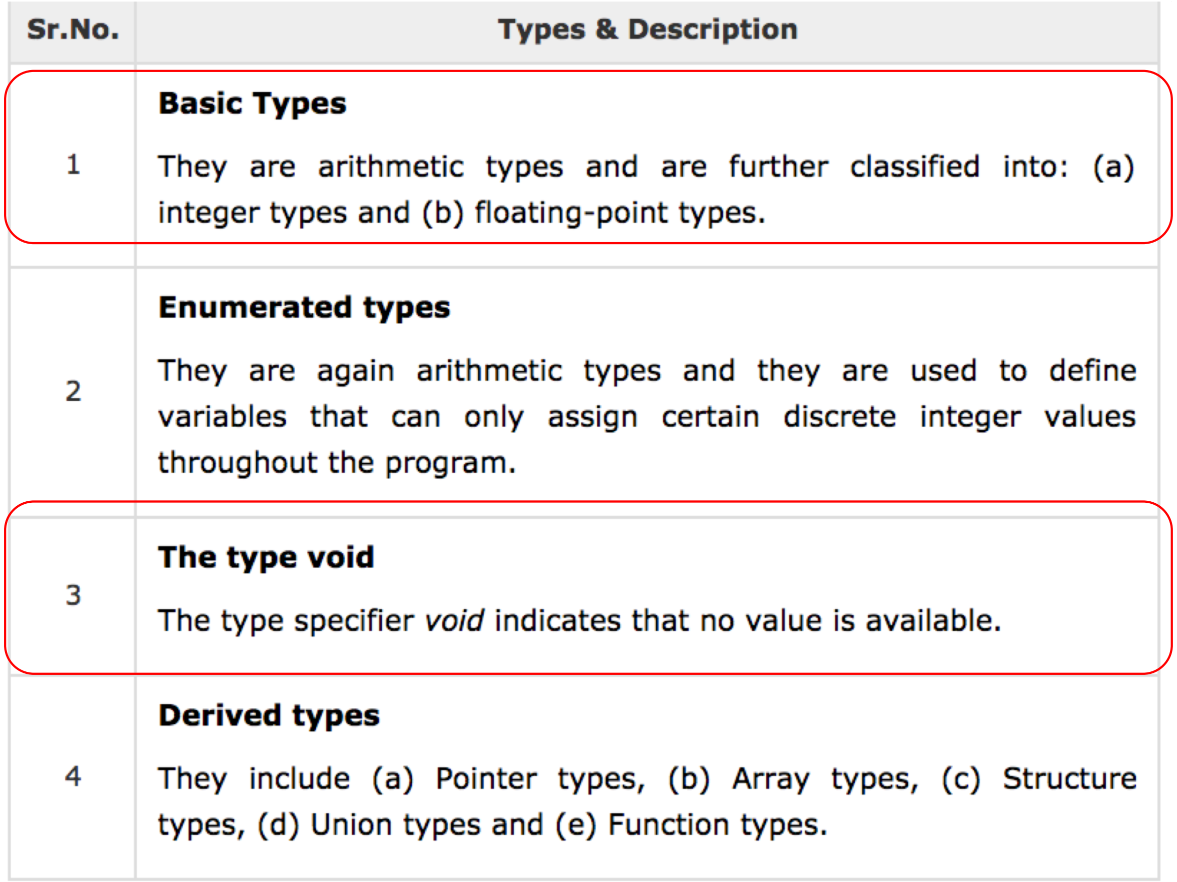
Important Topics to Write a C Code

- Data Types
- Qualifiers
  - Variables and Constants
- Operators (built-in)
- Data Modifiers
- Functions

Data Types

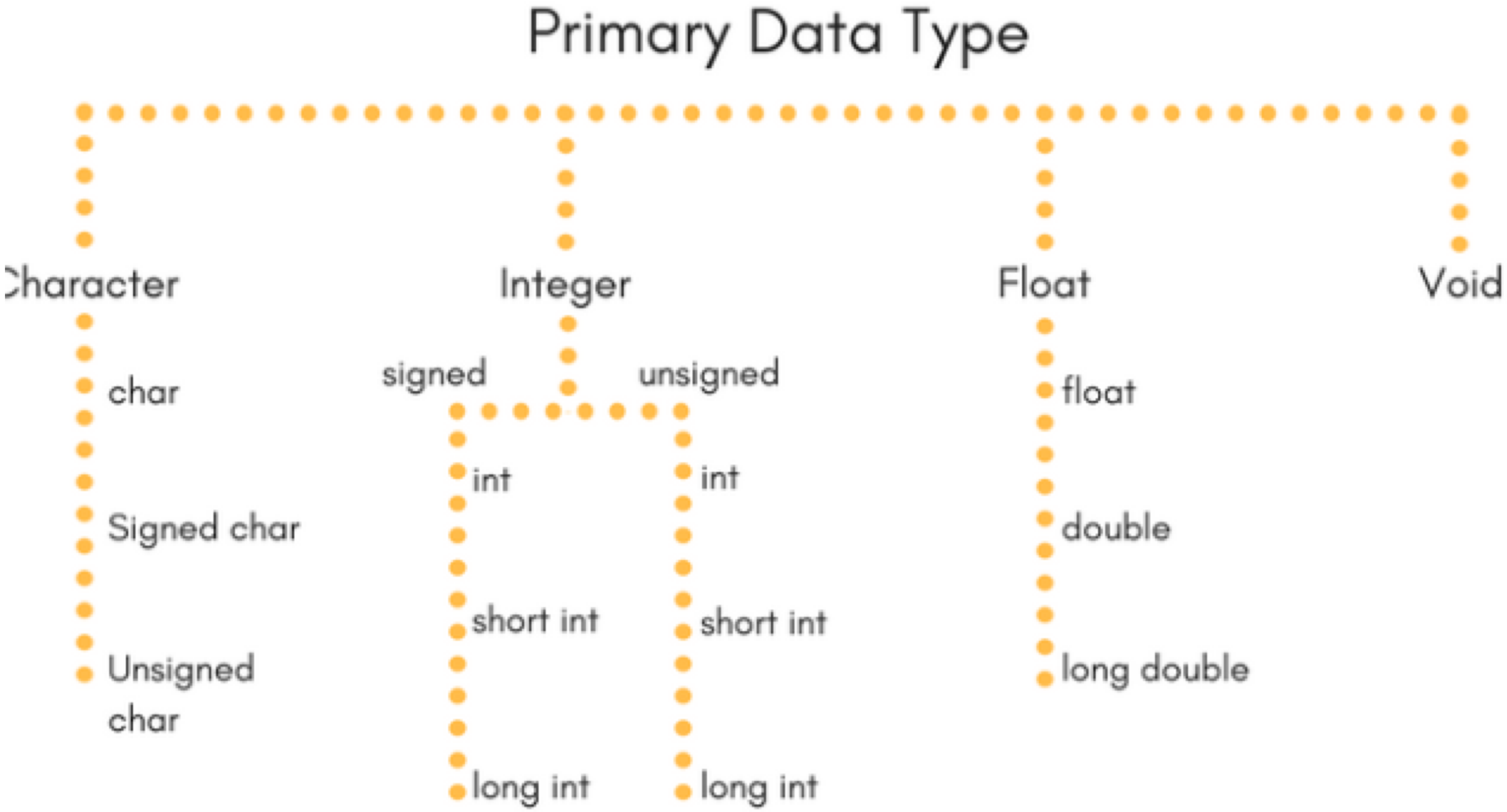
Every variable/function must have a specific data

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.
The types in C can be classified as follows −c



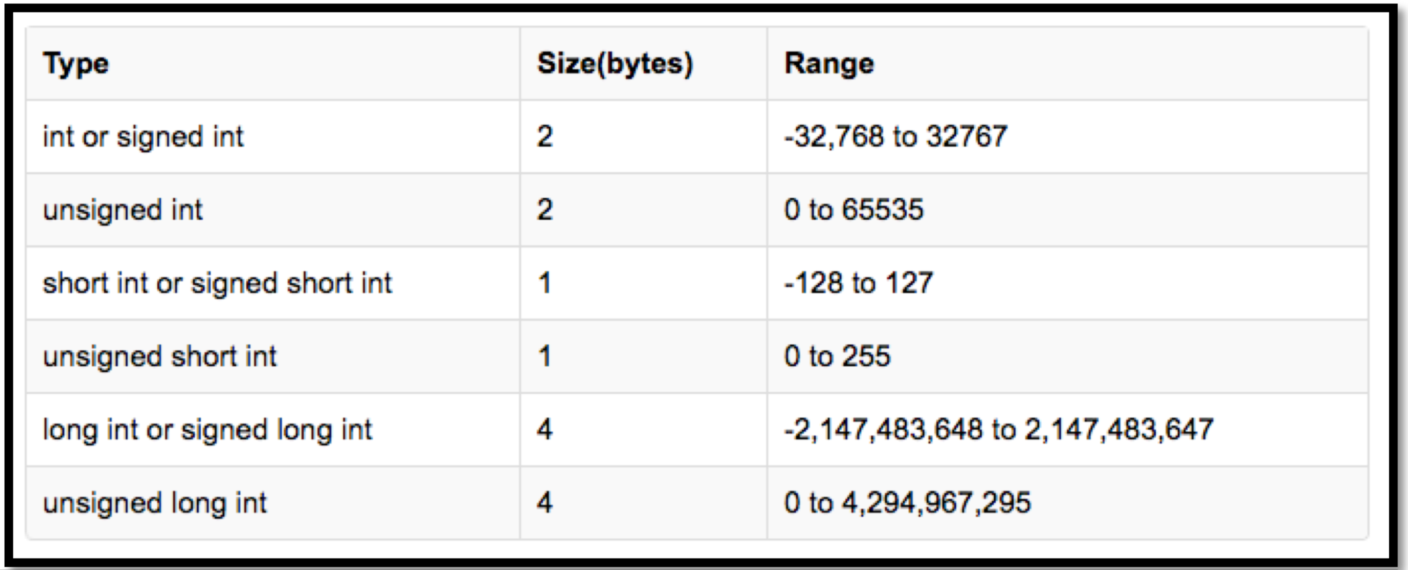
Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Data Types

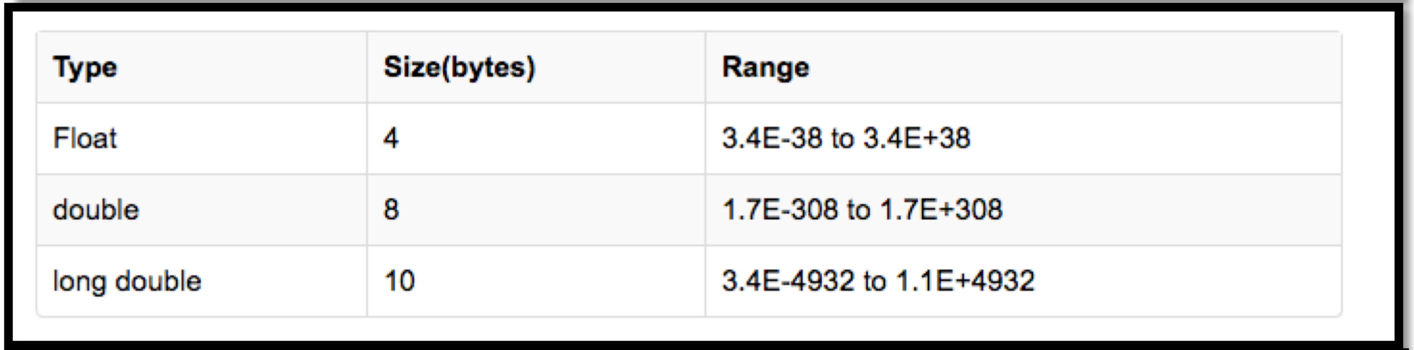


https://www.studytonight.com/c/datatype-in-c.php

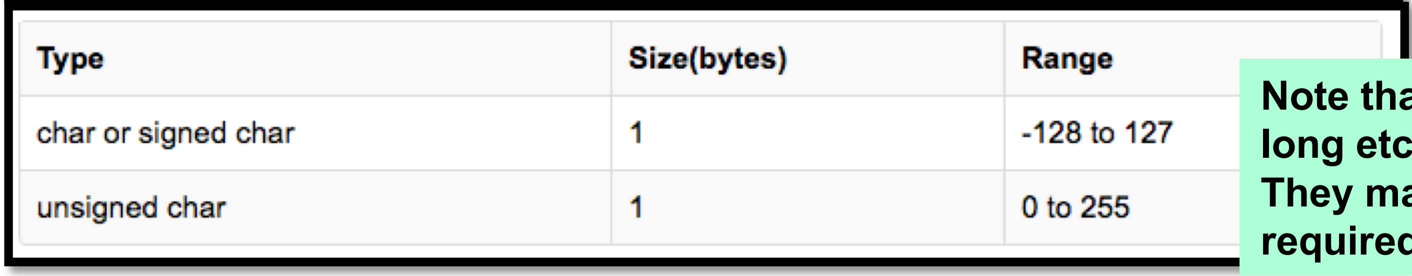
Types Specifiers (INT & FLOAT. CHAR, VOID)



Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295



Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932



Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

Void type

void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

Note that signed / unsigned / short / long etc. are called DATA modifiers! They manage the memory storage size required to store the variable

Integer

An **integer type** can be a decimal, octal, or hexadecimal constant.

A prefix specifies the base or radix:

- 0x or 0X for hexadecimal,
- 0 for octal,
- and nothing for decimal.

An integer type can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

```
85        /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30        /* int */
30u        /* unsigned int */
30l       /* long */
30ul       /* unsigned long */
```

```
#include <stdio.h>
#include <limits.h>
int main()
{
int d = 42;
int o = 051;
int x = 0x2b;
int X = 0X2A;
int b = 0b101010; // C++14

printf("%d, %d, %d, %d, %d", d, o, x, X, b);
/* Output: 42, 41, 43, 42, 42 */

   return 0;
}
```

C EXTERNAL Variable Data Type Examples

```
#include <stdio.h>

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {

   /* variable definition: */
   int a, b;
   int c;
   float f;

   /* actual initialization */
   a = 10;
   b = 20;

   c = a + b;
   printf("value of c : %d \n", c);

   f = 70.0/3.0;
   printf("value of f : %f \n", f);

   return 0;
}
```

Declare a variable at any place.

Defined in the main() function only

Type Qualifiers

- Key words applied to types making them *qualifies types*
  - *Const: Their values cannot change & stored in the program memory:* **`const`** `unsigned int x;`
  - Volatile*: Their values can change and located in the RAM:* **`volatile`** `int a = 5;`

Literal Constants

- A literal is a constant, but a constant is not a literal
  - `#define MAXINT 32767`
  - `const int MAXINT = 32767;`
  - Constants are labels that represent a literal
  - **Literals** are values, often assigned to symbolic constants and variables

- Literals or Literal Constants
  - Are "hard coded" values
  - May be numbers, characters, or strings
  - May be represented in a number of formats (decimal, hexadecimal, binary, character, etc.)
  - Always represent the same value (5 always represents the quantity five)

Defining Constants - Example

There are two simple ways in C to define constants –

•Using **#define** preprocessor.

•Using **const** keyword.

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

  return 0;
}
```

```
#include <stdio.h>

int main() {
   const int  LENGTH = 10;
   const int  WIDTH = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

Note that it is a good programming practice to define constants in CAPITALS.

Literal Constants

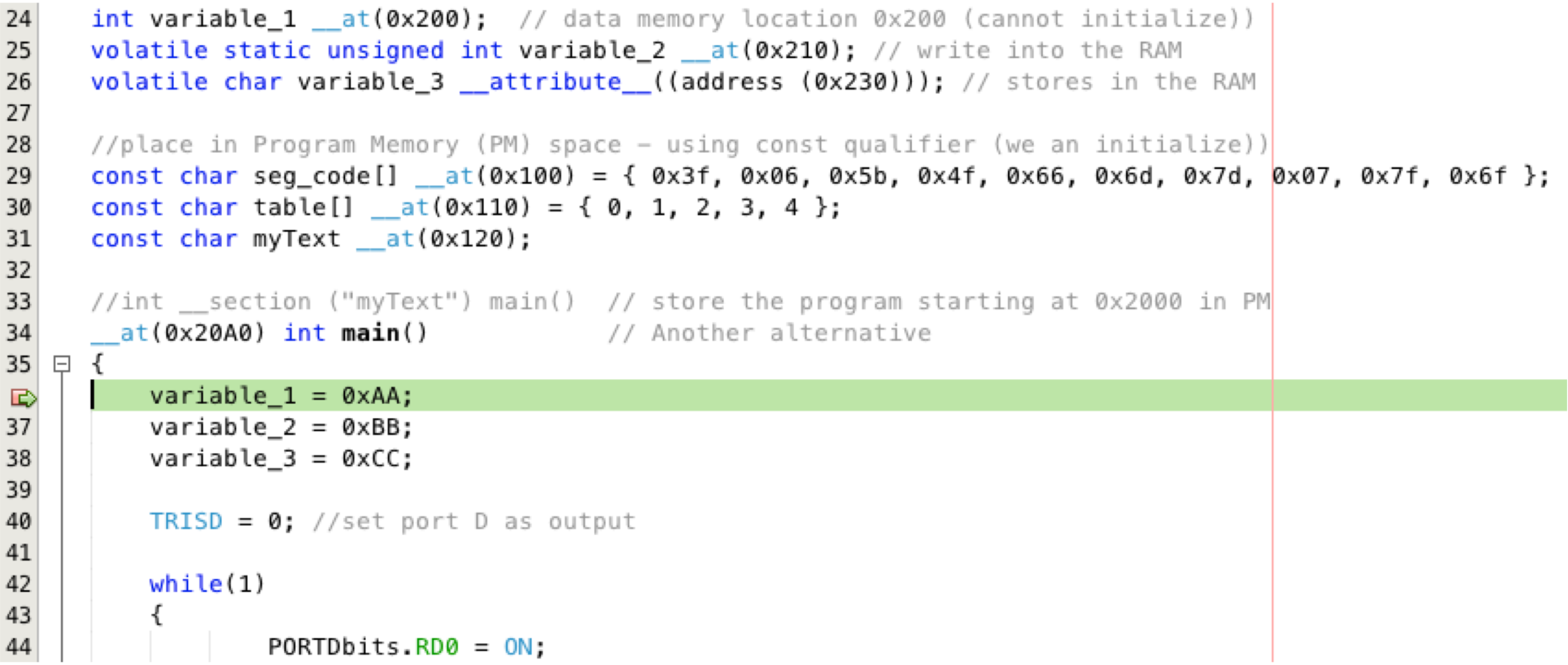
Example

```
unsigned int a;
unsigned int c;
#define b 2

void main(void)
{
   a = 5;
   c = a + b;
   printf("a=%d, b=%d, c=%d\n", a, b, c);
}
```

Literal

Literal

Type Qualifiers in XC8



```
int variable_1 __at(0x200);  // data memory location 0x200 (cannot initialize))
volatile static unsigned int variable_2 __at(0x210); // write into the RAM
volatile char variable_3 __attribute__((address (0x230))); // stores in the RAM

//place in Program Memory (PM) space - using const qualifier (we an initialize))
const char seg_code[] __at(0x100) = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f };
const char table[] __at(0x110) = { 0, 1, 2, 3, 4 };
const char myText __at(0x120);

//int __section ("myText") main()  // store the program starting at 0x2000 in PM
__at(0x20A0) int main()            // Another alternative
{
    variable_1 = 0xAA;
    variable_2 = 0xBB;
    variable_3 = 0xCC;

    TRISD = 0; //set port D as output

    while(1)
    {
            PORTDbits.RD0 = ON;
```

Function Data Type & Function Declaration

```
// function declaration
int func();

int main() {

   // function call
   int i = func();
}

// function definition
int func() {
   return 0;
}
```

For function declaration we can provide a function name at the time of its declaration and its actual definition can be given anywhere else.

Note the function type is INT

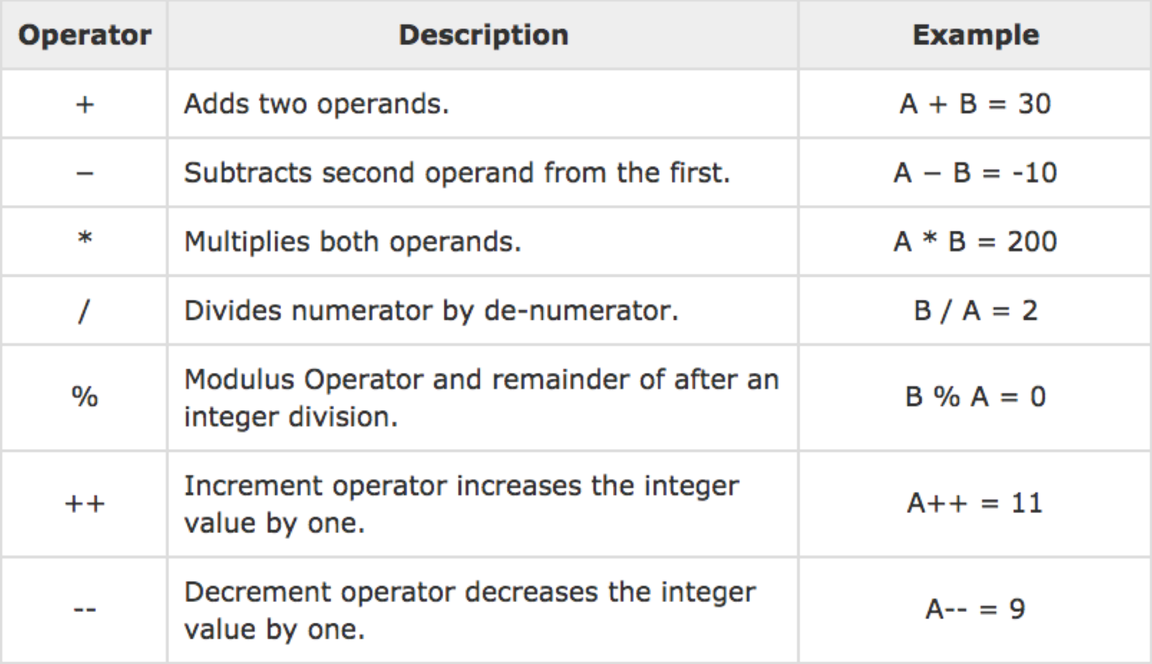
Let’s Talk About Operators….

C- Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators

- •Arithmetic Operators
- •Relational Operators
- •Logical Operators
- •Bitwise Operators
- •Assignment Operators
- •Misc Operators

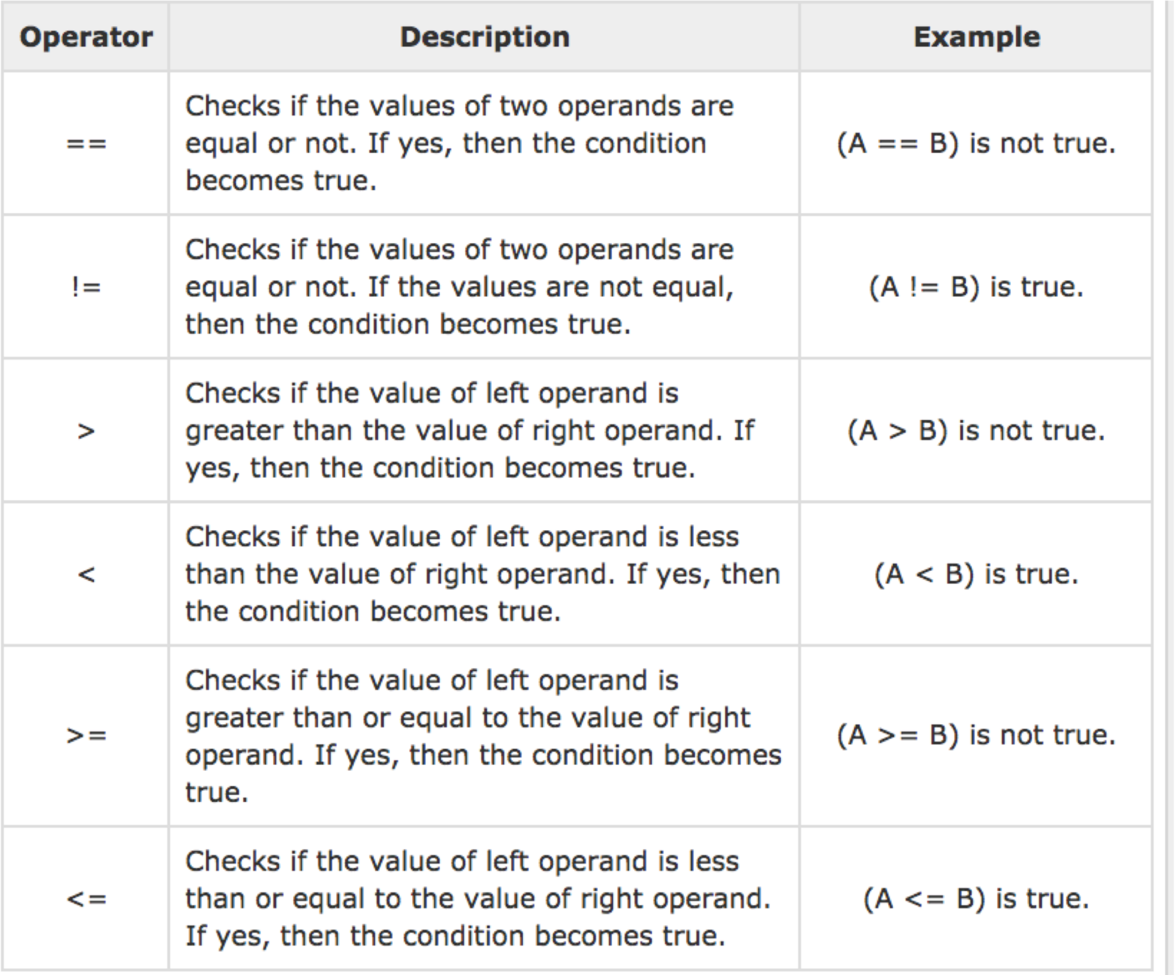
Arithmetic Operators



Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

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

C- Operators
Relational Operator

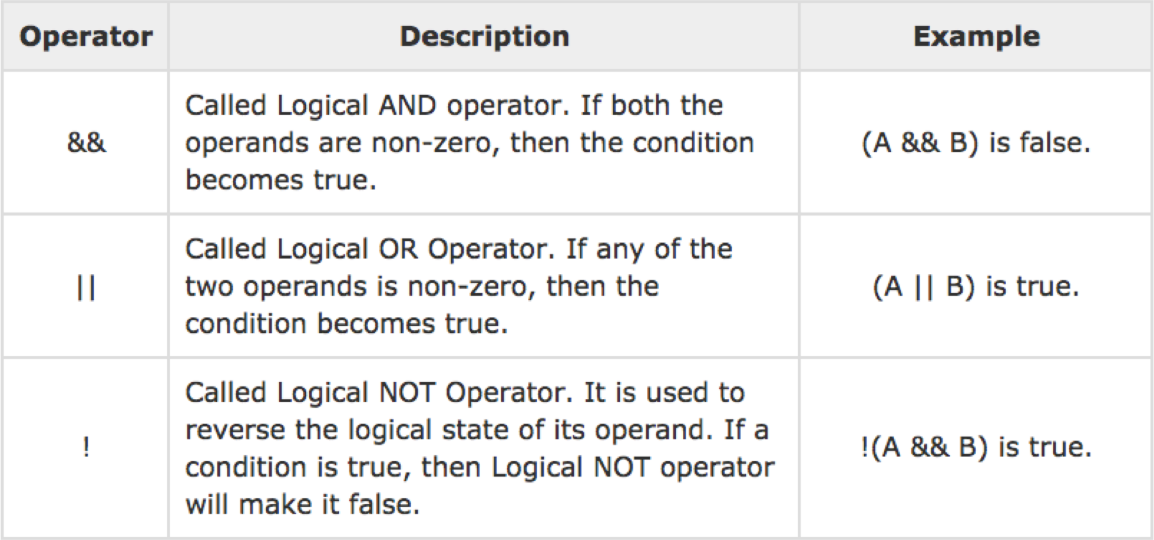


Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the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

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

C- Operators
Logical Operator

```
#include <stdio.h>

main() {

  int a = 5;
  int b = 20;
  int c ;

  if ( a && b ) {
    printf("Line 1 - Condition is true\n" );
  }

  if ( a || b ) {
    printf("Line 2 - Condition is true\n" );
  }

  /* lets change the value of  a and b */
  a = 0;
  b = 10;

  if ( a && b ) {
    printf("Line 3 - Condition is true\n" );
  } else {
    printf("Line 3 - Condition is not true\n" );
  }

  if ( !(a && b) ) {
    printf("Line 4 - Condition is true\n" );
  }

}
```

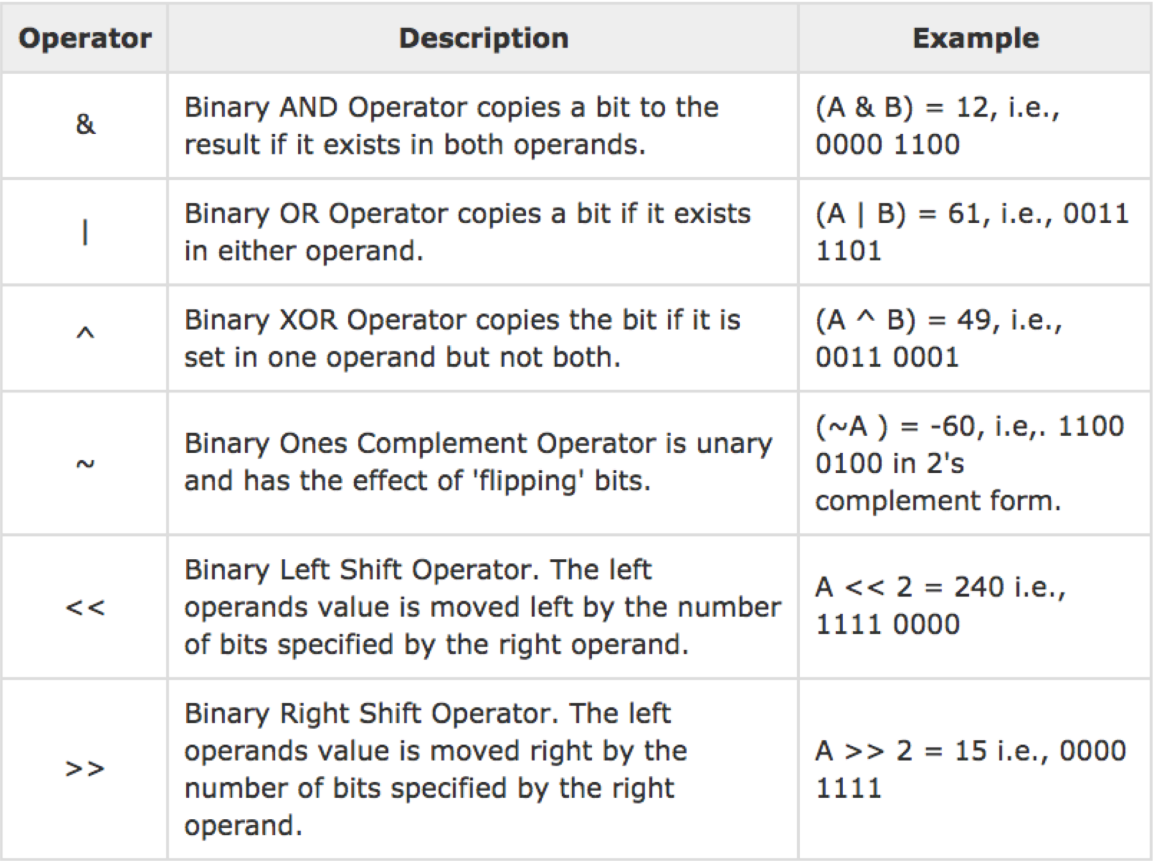


Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

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

C- Operators
Bitwise Operators

```
#include <stdio.h>

main() {

   unsigned int a = 60;/* 60 = 0011 1100 */
   unsigned int b = 13;/* 13 = 0000 1101 */
   int c = 0;

   c = a & b;        /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;        /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;        /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;           /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;       /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;       /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```



Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

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

Bitwise Operators -
Shift Operations

- Basic idea
  - Right shift (>>) & Left shift (<<)
- Example: if ch contains the bit pattern 11100101, then ch >> 1 will produce the result 01110010, and ch >> 2 will produce 00111001.
- Note – In C Code:

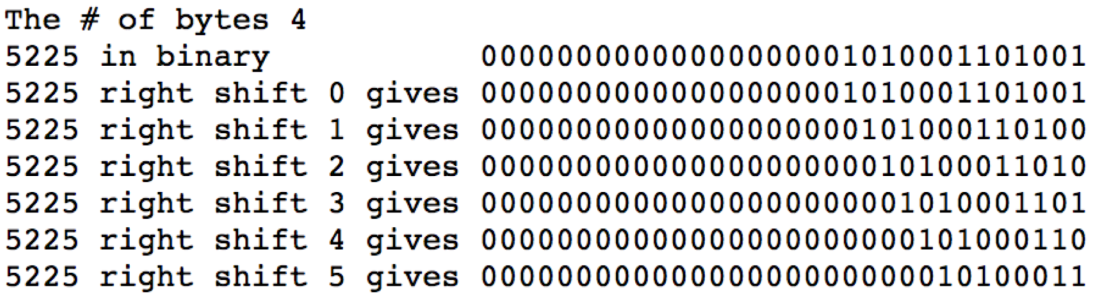
```
i = 14; // Bit pattern 00001110
j = i >> 1; // here we have the bit pattern shifted by 1 thus we get 00000111 = 7 which is 14/2
```

Bitwise Operators -

Shift Operations

```
#include <stdio.h>
void showbits(unsigned int x) // convert to binary
{
   int i;
   for(i=(sizeof(int)*8)-1; i>=0; i--)// (number of bytes * 8)-1
          //1u = unsigned value 1 is shifted
          (x&(1u<<i))?putchar('1'):putchar('0');

   printf("\n");
}
```

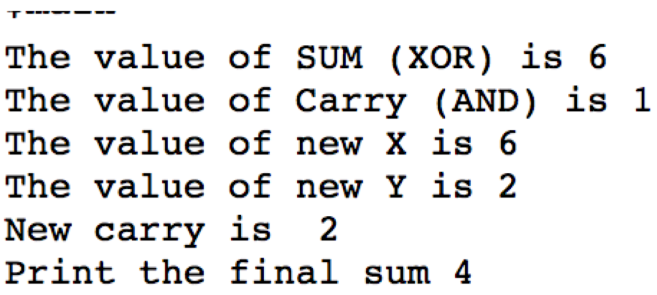
```
int main()
{
   printf("The # of bytes %d \n", sizeof(int)); // 4 bytes in the int
   int j = 5225, m, n;
   printf("%d in binary \t\t ", j);
   /* assume we have a function that prints a binary string when given
      a decimal integer
    */
   showbits(j);

   /* the loop for right shift operation */
   for ( m = 0; m <= 5; m++ ) {
      n = j >> m;
      printf("%d right shift %d gives ", j, m);
      showbits(n);
   }
   return 0;
}
```



```
The # of bytes 4
5225 in binary             00000000000000000001010001101001
5225 right shift 0 gives 00000000000000000001010001101001
5225 right shift 1 gives 00000000000000000000101000110100
5225 right shift 2 gives 00000000000000000000010100011010
5225 right shift 3 gives 00000000000000000000001010001101
5225 right shift 4 gives 00000000000000000000000101000110
5225 right shift 5 gives 00000000000000000000000010100011
```

Bitwise Operators - Shift Operations

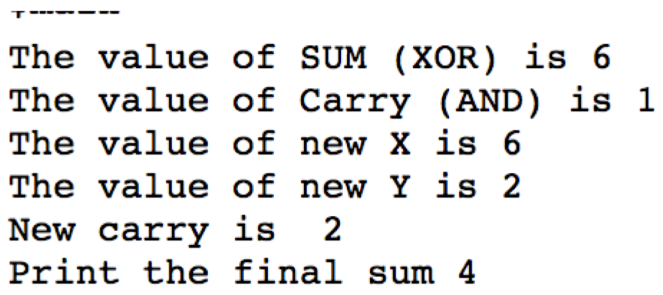
```
#include <stdio.h>

int main()
{
  unsigned int x = 5, y = 3, sum, carry;
  sum = x ^ y; // x XOR y
  printf("The value of SUM (XOR) is %d\n", sum);
  carry = x & y; // x AND y
  printf("The value of Carry (AND) is %d\n", carry);

  x ^= y; // x = x XOR y
  printf("The value of new X is %d\n", x);
  y &= x; // y = x AND y
  printf("The value of new Y is %d\n", y);

  x = 3, y = 1; // reinitialize
  while (!(carry & 2)) {
    carry = carry << 1; // left shift the carry
    x = sum; // initialize x as sum
    y = carry; // initialize y as carry
    sum = x ^ y; // sum is calculated
    carry = x & y; /* carry is calculated, the loop condition is
                      evaluated and the process is repeated until
                      carry is equal to 0.*/
   printf("New carry is  %u\n", carry);

  }
  printf("Print the final sum %u\n", sum); // the program will print 4
  return 0;
}
```



```
The value of SUM (XOR) is 6
The value of Carry (AND) is 1
The value of new X is 6
The value of new Y is 2
New carry is  2
Print the final sum 4
```

Bitwise Operators - Shift Operations

```
#include <stdio.h>

int main()
{
  unsigned int x = 5, y = 3, sum, carry;
  sum = x ^ y; // x XOR y
  printf("The value of SUM (XOR) is %d\n", sum);
  carry = x & y; // x AND y
  printf("The value of Carry (AND) is %d\n", carry);

  x ^= y; // x = x XOR y
  printf("The value of new X is %d\n", x);
  y &=
  printf

  x = 3
  while
    ca
    x =
    y = carry; // initialize y as carry
    sum = x ^ y; // sum is calculated
    carry = x & y; /* carry is calculated, the loop condition is
                  evaluated and the process is repeated until
                  carry is equal to 0.*/
   printf("New carry is  %u\n", carry);

  }
  printf("Print the final sum %u\n", sum); // the program will print 4
  return 0;
}
```

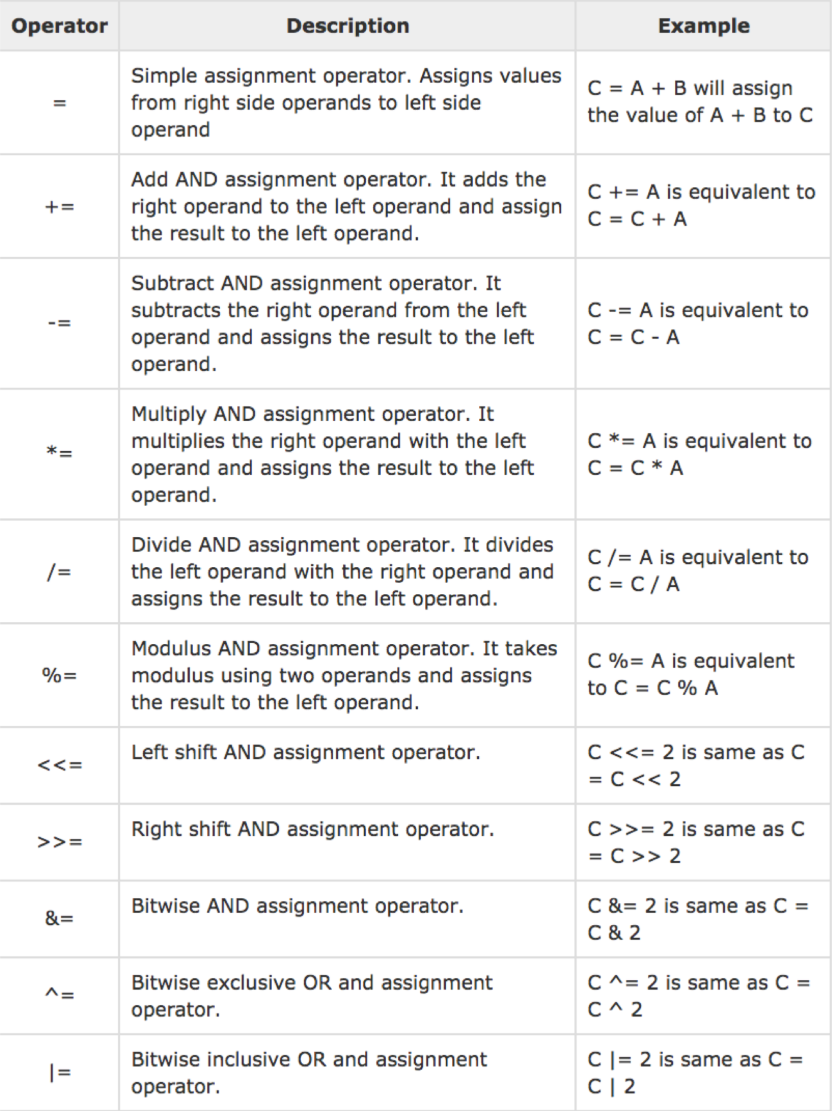


```
The value of SUM (XOR) is 6
The value of Carry (AND) is 1
The value of new X is 6
The value of new Y is 2
New carry is  2
Print the final sum 4
```

Answer the following questions:

- What is the difference between x ^=y and x=x^y?
- What is the significance of **while (!(carry & 2))**
- What is the significance of **while (carry !=0)**
- If y = 6 and x = 2 what will be the new value of y and y after **y |= x** ?

Assignment Operators

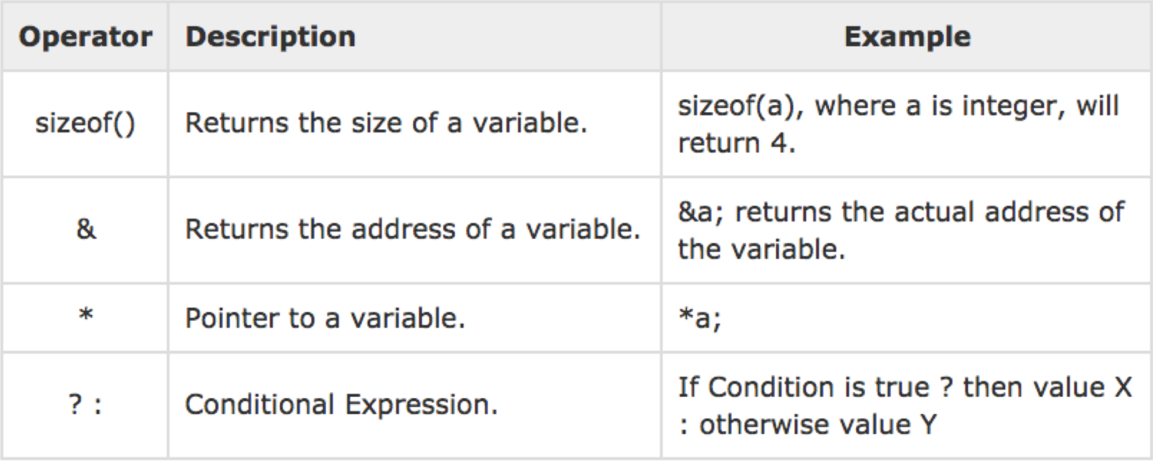


Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

```
#include <stdio.h>
main() {
  int a = 21;
  int c ;
  c =  a;
  printf("Line 1 - =  Operator Example, Value of c = %d\n", c );
  c +=  a;
  printf("Line 2 - += Operator Example, Value of c = %d\n", c );
  c -=  a;
  printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
  c *=  a;
  printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
  c /=  a;
  printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
  c  = 200;
  c %=  a;
  printf("Line 6 - %= Operator Example, Value of c = %d\n", c );
  c <<=  2;
  printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );
  c >>=  2;
  printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
  c &=  2;
  printf("Line 9 - &= Operator Example, Value of c = %d\n", c );
  c ^=  2;
  printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );
  c |=  2;
  printf("Line 11 - |= Operator Example, Value of c = %d\n", c );
}
```

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

C- Operators
Misc Operators



Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

```
#include <stdio.h>

main() {

   int a = 4;
   short b;
   double c;
   int* ptr;

   /* example of sizeof operator */
   printf("Line 1 - Size of variable a = %d\n", sizeof(a) );
   printf("Line 2 - Size of variable b = %d\n", sizeof(b) );
   printf("Line 3 - Size of variable c= %d\n", sizeof(c) );

   /* example of & and * operators */
   ptr = &a;    /* 'ptr' now contains the address of 'a'*/
   printf("value of a is  %d\n", a);
   printf("*ptr is %d.\n", *ptr);

   /* example of ternary operator */
   a = 10;
   b = (a == 1) ? 20: 30;
   printf( "Value of b is %d\n", b );

   b = (a == 10) ? 20: 30;
   printf( "Value of b is %d\n", b );
}
```

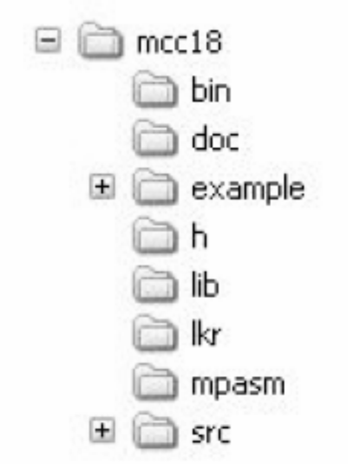
```
Line 1 - Size of variable a = 4
Line 2 - Size of variable b = 2
Line 3 - Size of variable c= 8 value of a
is 4 *ptr is 4.
Value of b is 30
Value of b is 20
```

Using IDE

MPLAB C18/XC8 Directory Structure

- MPLAB C18/XC8 can be installed anywhere on the PC. Its default installation directory is the C:\mcC18/XC8 folder.
- MPLAB IDE should be installed on the PC prior to installing MPLAB C18/XC8.

MPLAB® C18 DIRECTORY STRUCTURE



```
⊟ mcc18
     bin
     doc
  ⊞  example
     h
     lib
     lkr
     mpasm
  ⊞  src
```

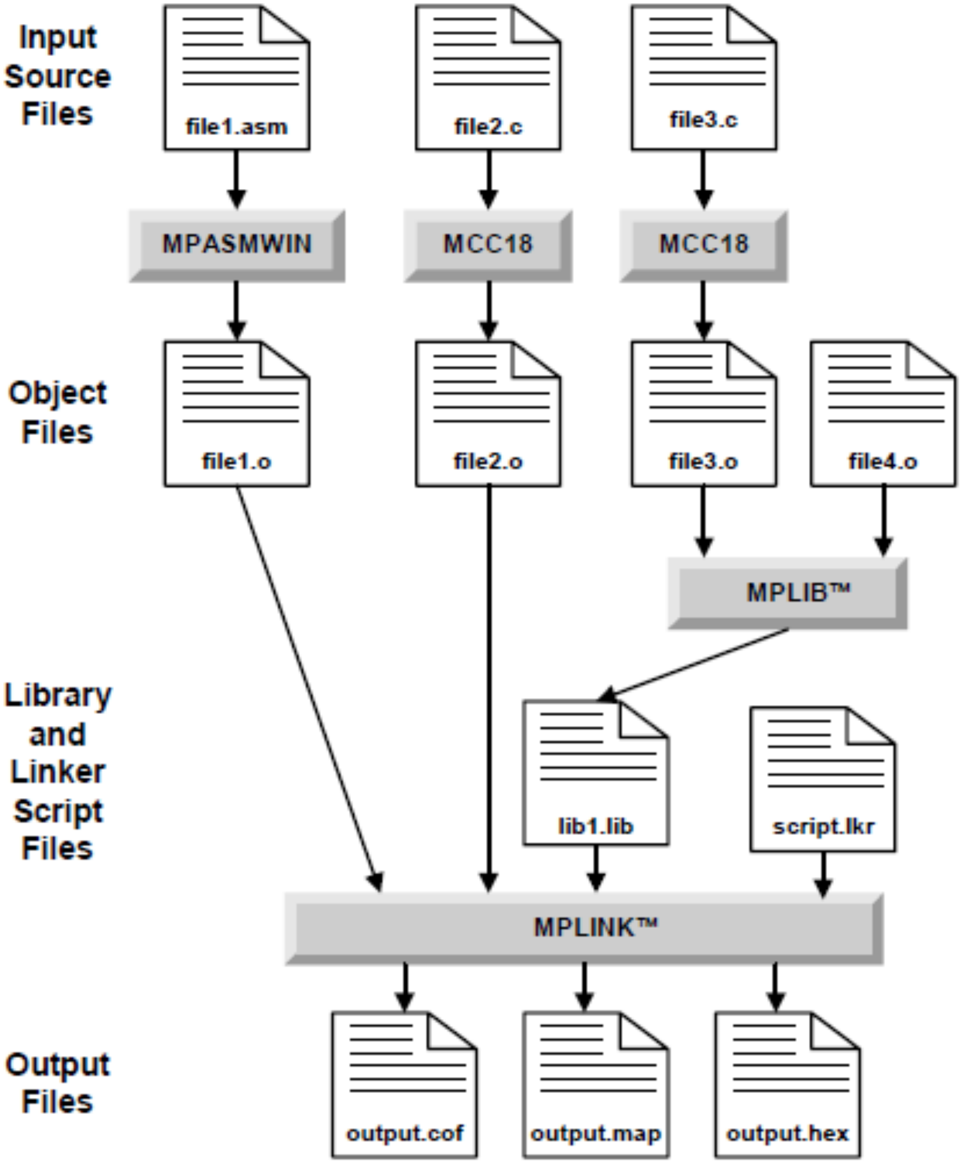
h: **Contains the header files for the standard C library and the processor-specific libraries for the supported PICmicro® MCUs.**

lib: **Contains the standard C library (clib.lib or clib_e.lib), the processor-specific libraries (p18*xxxx*.lib or p18*xxxx*_e.lib, where *xxxx* is the specific device number) and the start-up modules (c018.o, c018_e.o, c018i.o, c018i_e.o, c018iz.o, c018iz_e.o).**

lkr: **Contains the linker script files for use with MPLAB C18/XC8.**

mpasm: **Contains the MPASM assembler and the assembly header files for the devices supported by MPLAB C18/XC8 (p18xxxx.inc).**

LANGUAGE TOOLS EXECUTION FLOW



Input Source Files

file1.asm

file2.c

file3.c

MPASMWIN

MCC18

MCC18

Object Files

file1.o

file2.o

file3.o

file4.o

MPLIB™

Library and Linker Script Files

lib1.lib

script.lkr

MPLINK™

Output Files

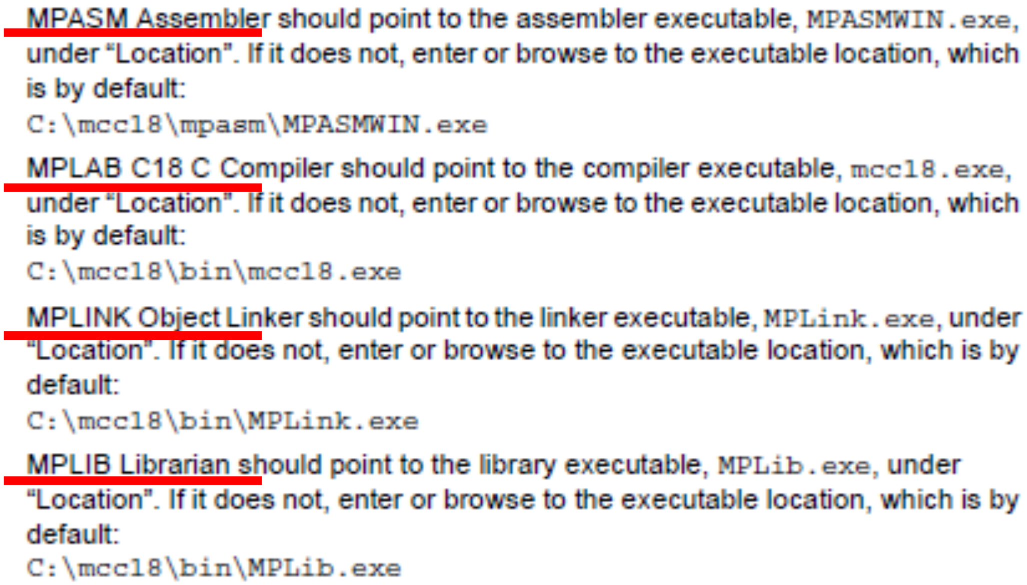
output.cof

output.map

output.hex

Installation Notes for MCC 18

- Make sure executable file locations are properly assigned

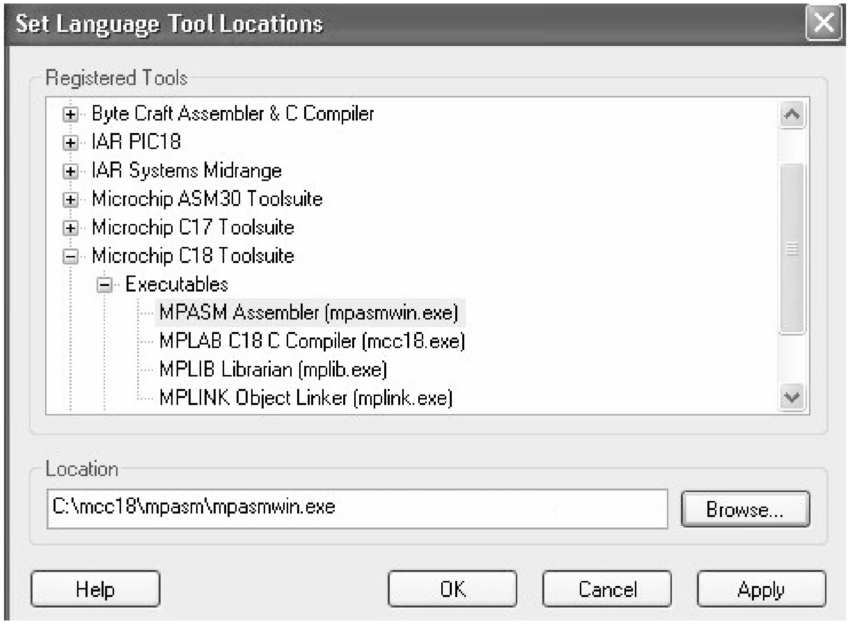


MPASM Assembler should point to the assembler executable, `MPASMWIN.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\mpasm\MPASMWIN.exe`

MPLAB C18 C Compiler should point to the compiler executable, `mcc18.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\mcc18.exe`

MPLINK Object Linker should point to the linker executable, `MPLink.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\MPLink.exe`

MPLIB Librarian should point to the library executable, `MPLib.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\MPLib.exe`



Set Language Tool Locations

Registered Tools

- Byte Craft Assembler & C Compiler
- IAR PIC18
- IAR Systems Midrange
- Microchip ASM30 Toolsuite
- Microchip C17 Toolsuite
- Microchip C18 Toolsuite
  - Executables
    - MPASM Assembler (mpasmwin.exe)
    - MPLAB C18 C Compiler (mcc18.exe)
    - MPLIB Librarian (mplib.exe)
    - MPLINK Object Linker (mplink.exe)

Location: C:\mcc18\mpasm\mpasmwin.exe

Setting up the IDE in MPLAB

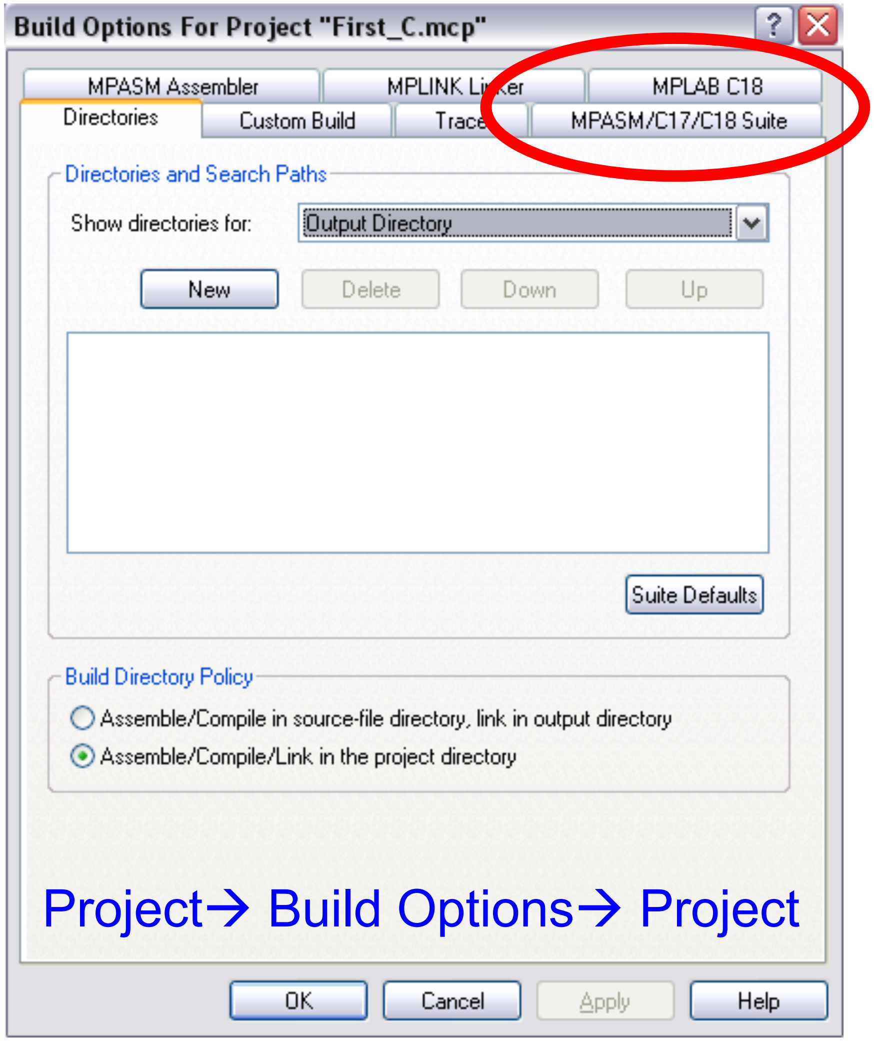
1. Under the “Configure” label on the menu bar, select “Select Device” from the dropdown menu and choose the microcontroller for the project.

2. Under the “Project” label on the menu bar, select “Project Wizard” from the dropdown menu and again select the microcontroller for the project.

3. In Step 2 of the project wizard, select the “Microchip C18/XC8 Toolsuite” and click next. The paths are all correct of the C18/XC8 compiler is installed properly.

4. Enter a name for the project and a directory and then click on next.

Installing C18/XC8 Compiler



Build Options For Project "First_C.mcp"

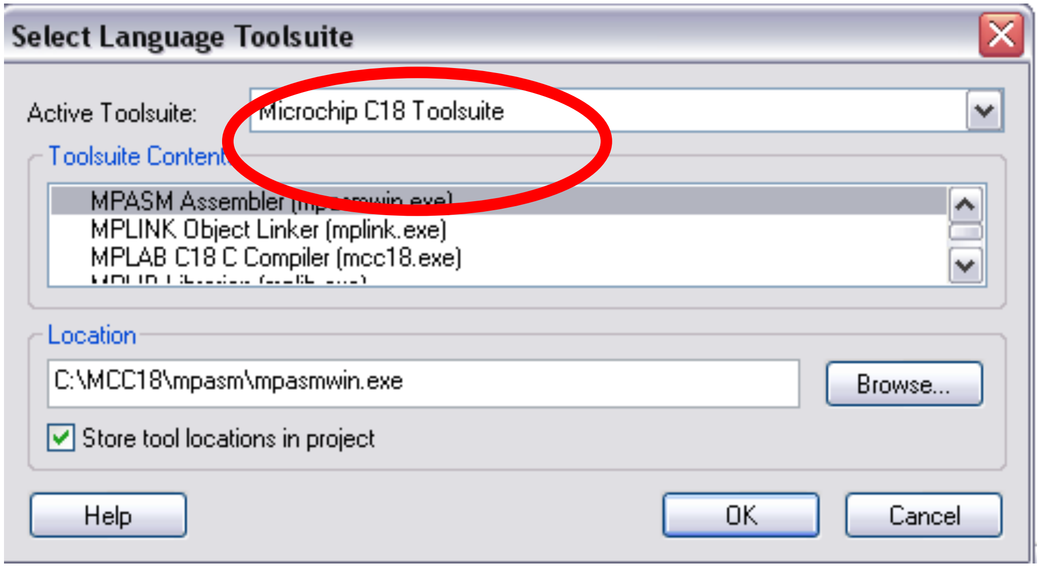
Show directories for: Output Directory

○ Assemble/Compile in source-file directory, link in output directory

● Assemble/Compile/Link in the project directory

Project→ Build Options→ Project

Project→ Select Language Toolsuite



Select Language Toolsuite

Active Toolsuite: Microchip C18 Toolsuite

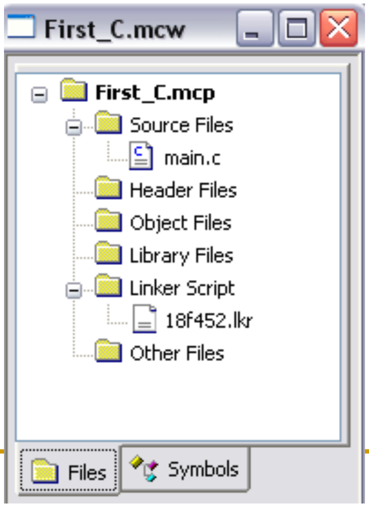
Toolsuite Contents

MPASM Assembler (mpasmwin.exe)
MPLINK Object Linker (mplink.exe)
MPLAB C18 C Compiler (mcc18.exe)
MPLIB Librarian (mplib.exe)

Location

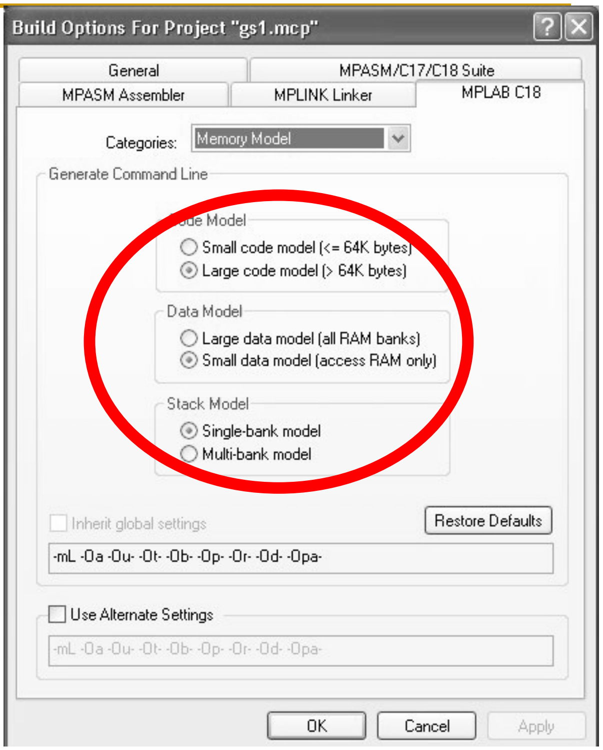
C:\MCC18\mpasm\mpasmwin.exe

☑ Store tool locations in project



Program Examples

- Make and Build the project
- If you are using standard outputs:
  - Select *Debugger>Settings* and click on the **Uart1 IO** tab. The box marked
  - **Enable Uart1 IO** should be checked, and the **Output** should be set to **Window**
- **Select large code model**



Categories: Memory Model

Generate Command Line

de Model
○ Small code model (<= 64K bytes)
● Large code model (> 64K bytes)

Data Model
○ Large data model (all RAM banks)
● Small data model (access RAM only)

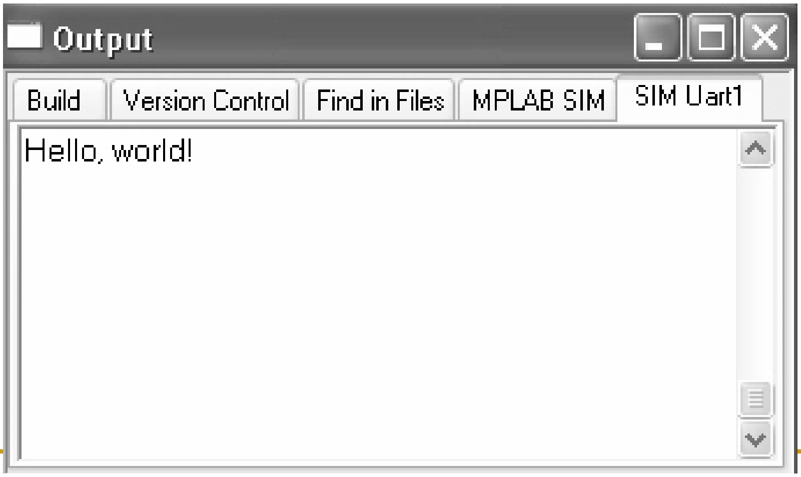
Stack Model
● Single-bank model
○ Multi-bank model

□ Inherit global settings

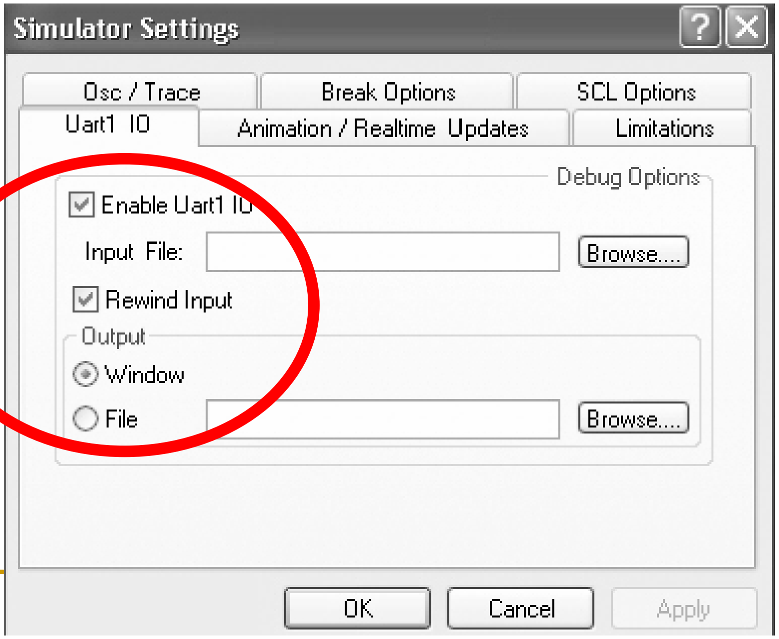
```
-mL -Oa -Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-
```

□ Use Alternate Settings

```
-mL -Oa -Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-
```



```
Hello, world!
```



☑ Enable Uart1 IO

Input File:

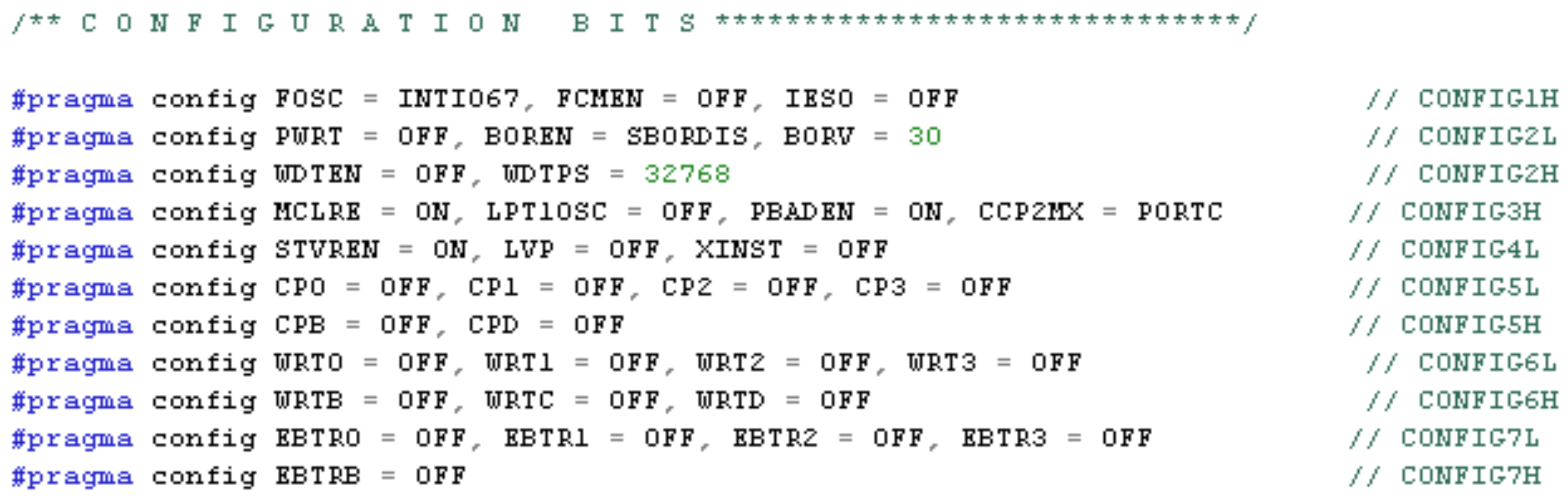
☑ Rewind Input

Output
● Window
○ File

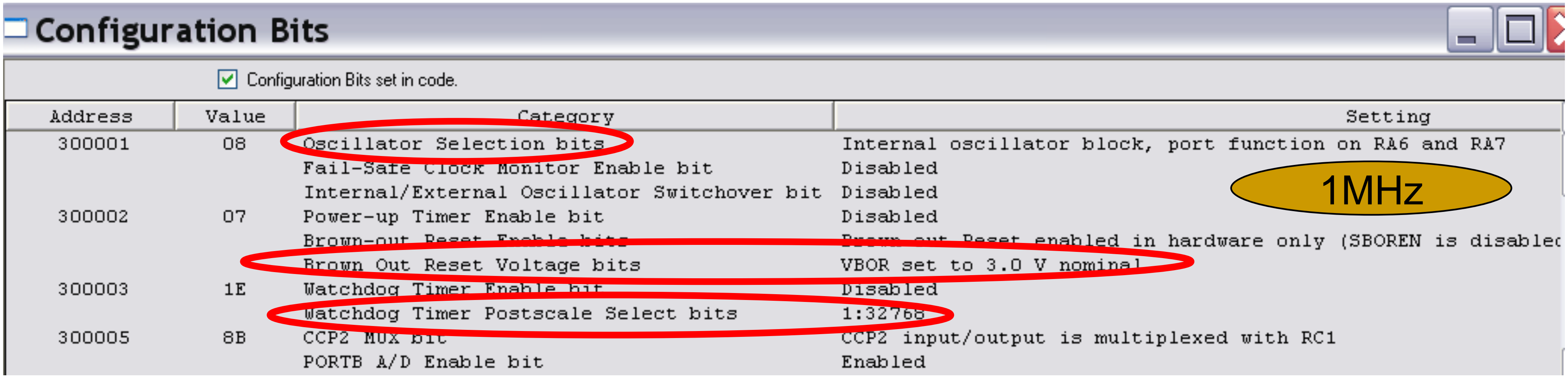
Configuration Bits

Modified for PIC18/XC8F4xK20

#pragma is used to declare directive;
Config directive allows configuring MCU operating modes; device dependent



```
/** C O N F I G U R A T I O N   B I T S ******************************/

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF                      // CONFIG1H
#pragma config PWRT = OFF, BOREN = SBORDIS, BORV = 30                       // CONFIG2L
#pragma config WDTEN = OFF, WDTPS = 32768                                   // CONFIG2H
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = ON, CCP2MX = PORTC      // CONFIG3H
#pragma config STVREN = ON, LVP = OFF, XINST = OFF                         // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF                  // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                                        // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF               // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                           // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF          // CONFIG7L
#pragma config EBTRB = OFF                                                 // CONFIG7H
```



☑ Configuration Bits set in code.

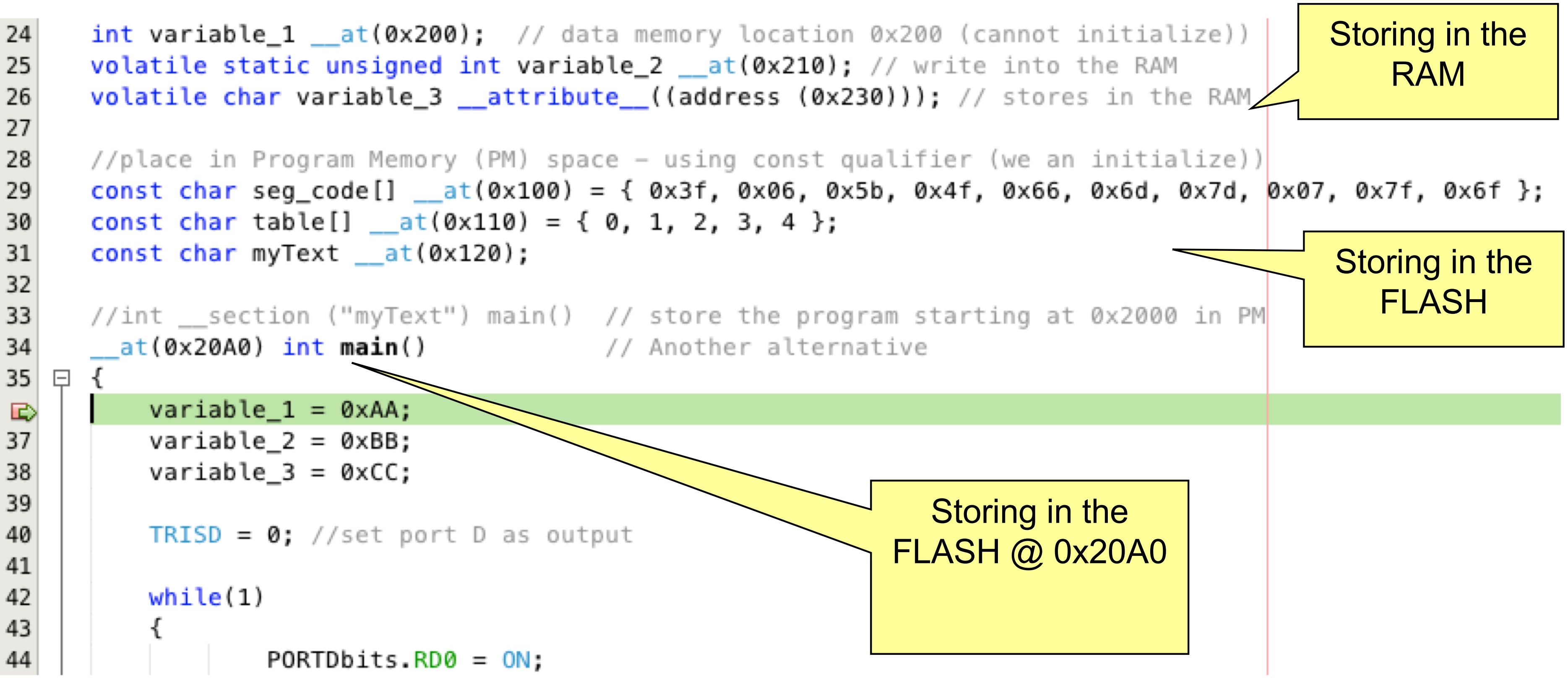
Address	Value	Category	Setting
300001	08	Oscillator Selection bits	Internal oscillator block, port function on RA6 and RA7
		Fail-Safe Clock Monitor Enable bit	Disabled
		Internal/External Oscillator Switchover bit	Disabled
300002	07	Power-up Timer Enable bit	Disabled
		Brown-out Reset Enable bits	Brown-out Reset enabled in hardware only (SBOREN is disablec
		Brown Out Reset Voltage bits	VBOR set to 3.0 V nominal
300003	1E	Watchdog Timer Enable bit	Disabled
		Watchdog Timer Postscale Select bits	1:32768
300005	8B	CCP2 MUX bit	CCP2 input/output is multiplexed with RC1
		PORTB A/D Enable bit	Enabled

1MHz

Type Qualifiers in XC8

```
__EEPROM_DATA(0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07);
```

Storing in the EEPROM



```
int variable_1 __at(0x200);  // data memory location 0x200 (cannot initialize))
volatile static unsigned int variable_2 __at(0x210); // write into the RAM
volatile char variable_3 __attribute__((address (0x230))); // stores in the RAM

//place in Program Memory (PM) space - using const qualifier (we an initialize))
const char seg_code[] __at(0x100) = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f };
const char table[] __at(0x110) = { 0, 1, 2, 3, 4 };
const char myText __at(0x120);

//int __section ("myText") main()  // store the program starting at 0x2000 in PM
__at(0x20A0) int main()            // Another alternative
{
    variable_1 = 0xAA;
    variable_2 = 0xBB;
    variable_3 = 0xCC;

    TRISD = 0; //set port D as output

    while(1)
    {
        PORTDbits.RD0 = ON;
```

Storing in the RAM

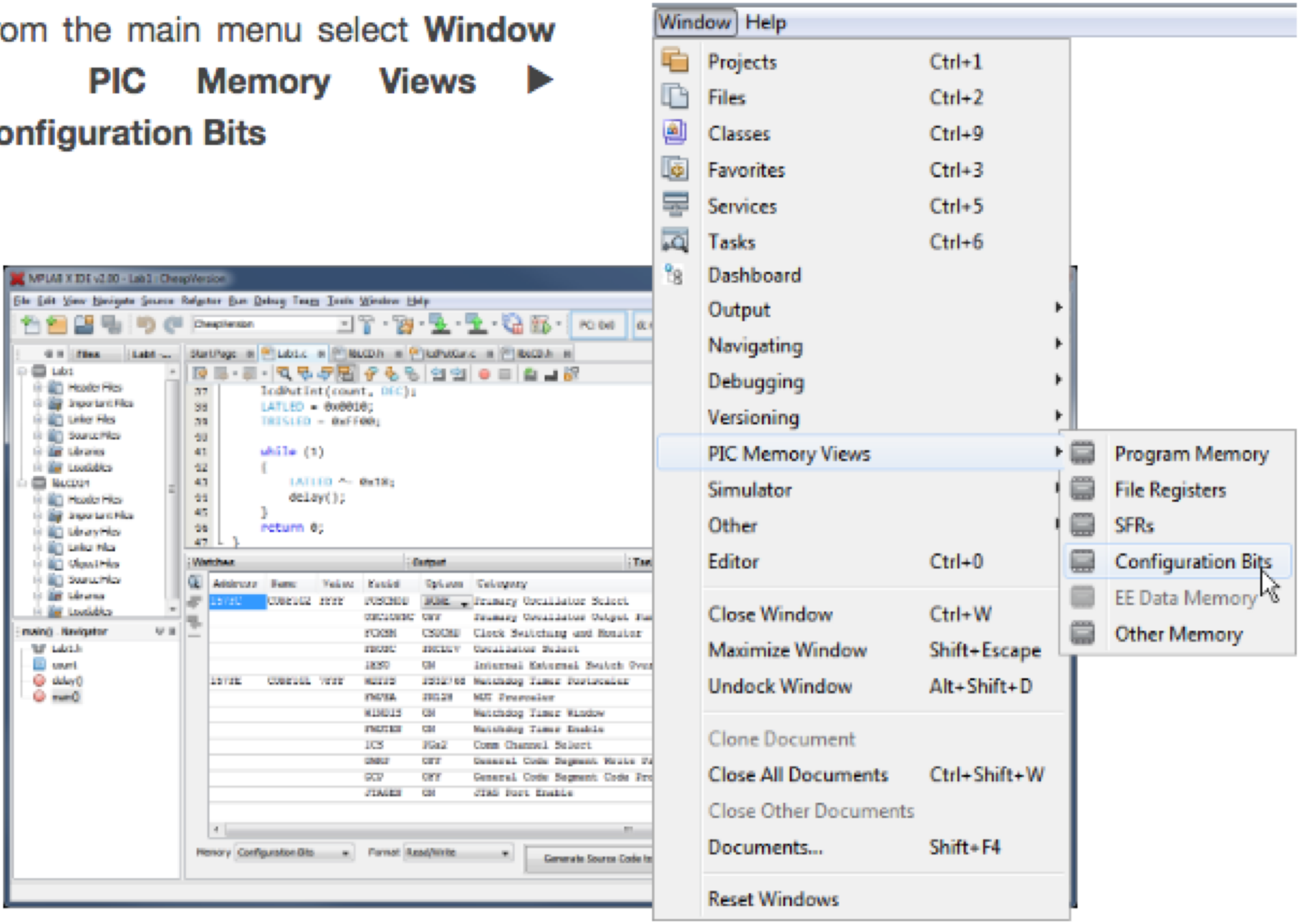
Storing in the FLASH

Storing in the FLASH @ 0x20A0

Viewing Bit Configurations

How to display the Configuration Bits window

From the main menu select **Window ▶ PIC Memory Views ▶ Configuration Bits**



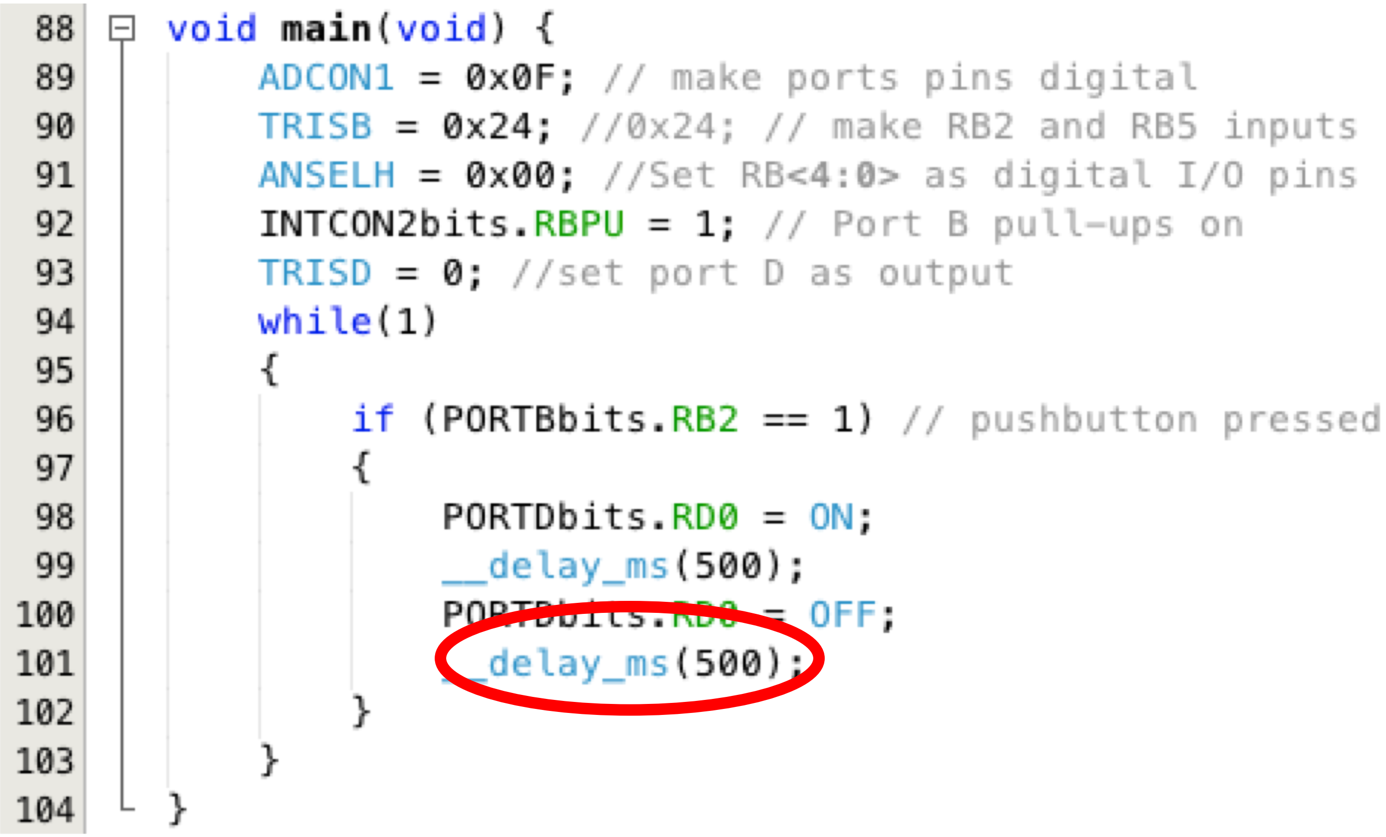
```
lcdPutInt(count, DEC);
LATLED = 0x0010;
TRISLED = 0xFF00;

while (1)
{
    LATLED ^= 0x10;
    delay();
}
return 0;
}
```

Address	Name	Value	Field	Option	Category
157FC	CONFIG2	FFFF	POSCMOD	NONE	Primary Oscillator Select
			OSCIOFNC	OFF	Primary Oscillator Output Fun
			FCKSM	CSDCMD	Clock Switching and Monitor
			FNOSC	FRCDIV	Oscillator Select
			IESO	ON	Internal External Switch Over
157FE	CONFIG1	7FFF	WDTPS	PS32768	Watchdog Timer Postscaler
			FWPSA	PR128	WDT Prescaler
			WINDIS	ON	Watchdog Timer Window
			FWDTEN	ON	Watchdog Timer Enable
			ICS	PGx2	Comm Channel Select
			GWRP	OFF	General Code Segment Write Pr
			GCP	OFF	General Code Segment Code Pro
			JTAGEN	ON	JTAG Port Enable

Memory: Configuration Bits Format: Read/Write

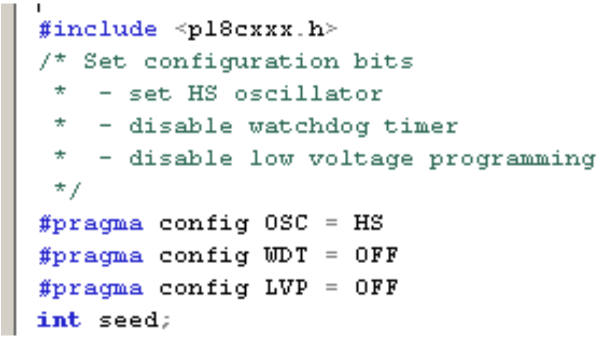
Time Delay Functions in XC8

#include <xc.h>
#include <time.h>

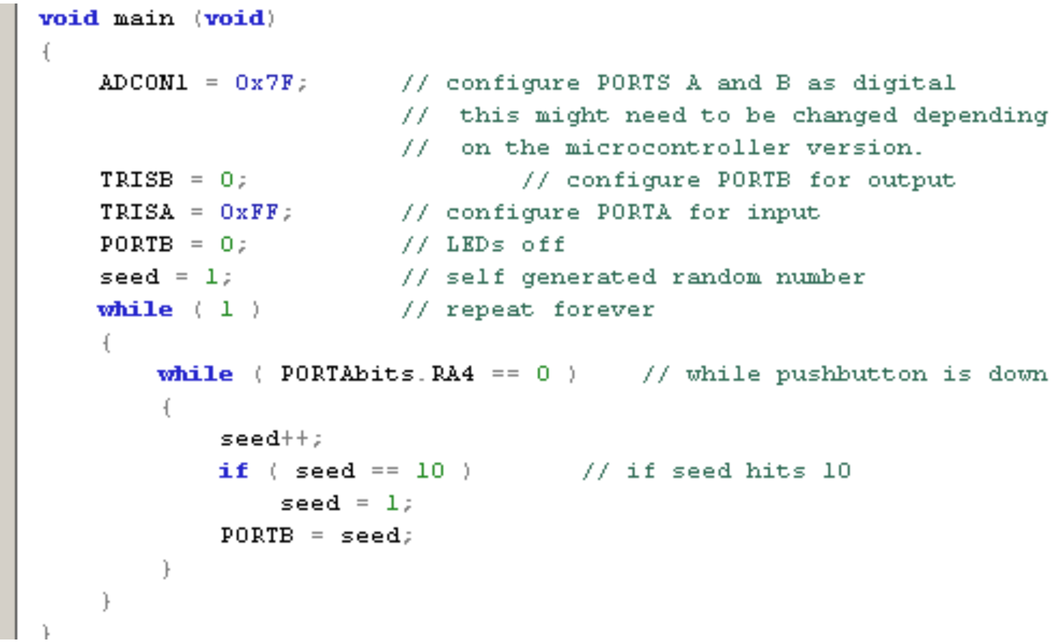


```
void main(void) {
    ADCON1 = 0x0F; // make ports pins digital
    TRISB = 0x24; //0x24; // make RB2 and RB5 inputs
    ANSELH = 0x00; //Set RB<4:0> as digital I/O pins
    INTCON2bits.RBPU = 1; // Port B pull-ups on
    TRISD = 0; //set port D as output
    while(1)
    {
        if (PORTBbits.RB2 == 1) // pushbutton pressed
        {
            PORTDbits.RD0 = ON;
            __delay_ms(500);
            PORTDbits.RD0 = OFF;
            __delay_ms(500);
        }
    }
}
```

Random Number Generator in XC8



```
#include <p18cxxx.h>
/* Set configuration bits
 *   - set HS oscillator
 *   - disable watchdog timer
 *   - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
int seed;
```



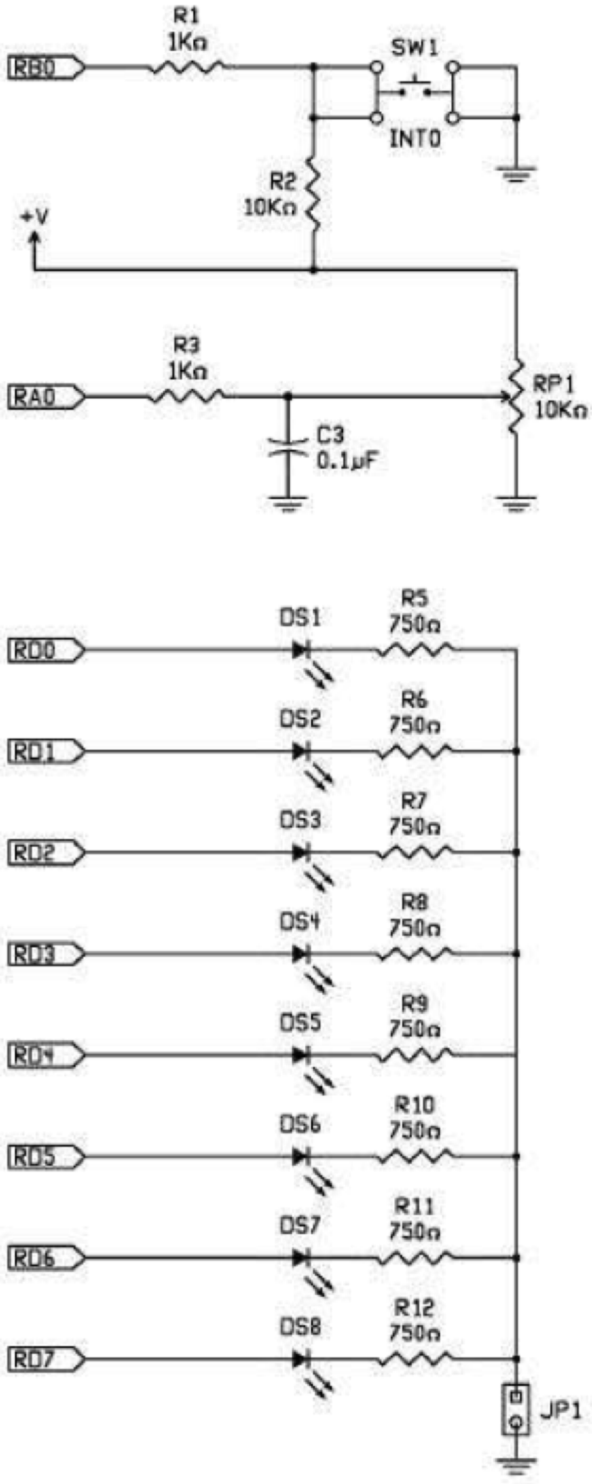
```
void main (void)
{
    ADCON1 = 0x7F;          // configure PORTS A and B as digital
                            //  this might need to be changed depending
                            //  on the microcontroller version.
    TRISB = 0;                      // configure PORTB for output
    TRISA = 0xFF;           // configure PORTA for input
    PORTB = 0;              // LEDs off
    seed = 1;               // self generated random number
    while ( 1 )             // repeat forever
    {
        while ( PORTAbits.RA4 == 0 )    // while pushbutton is down
        {
            seed++;
            if ( seed == 10 )       // if seed hits 10
                seed = 1;
            PORTB = seed;
        }
    }
}
```

Display a random number when RA4 is pressed
Random number will be between 0-9

Random Number Generator

Modified for PIC18/XC8F4xK20

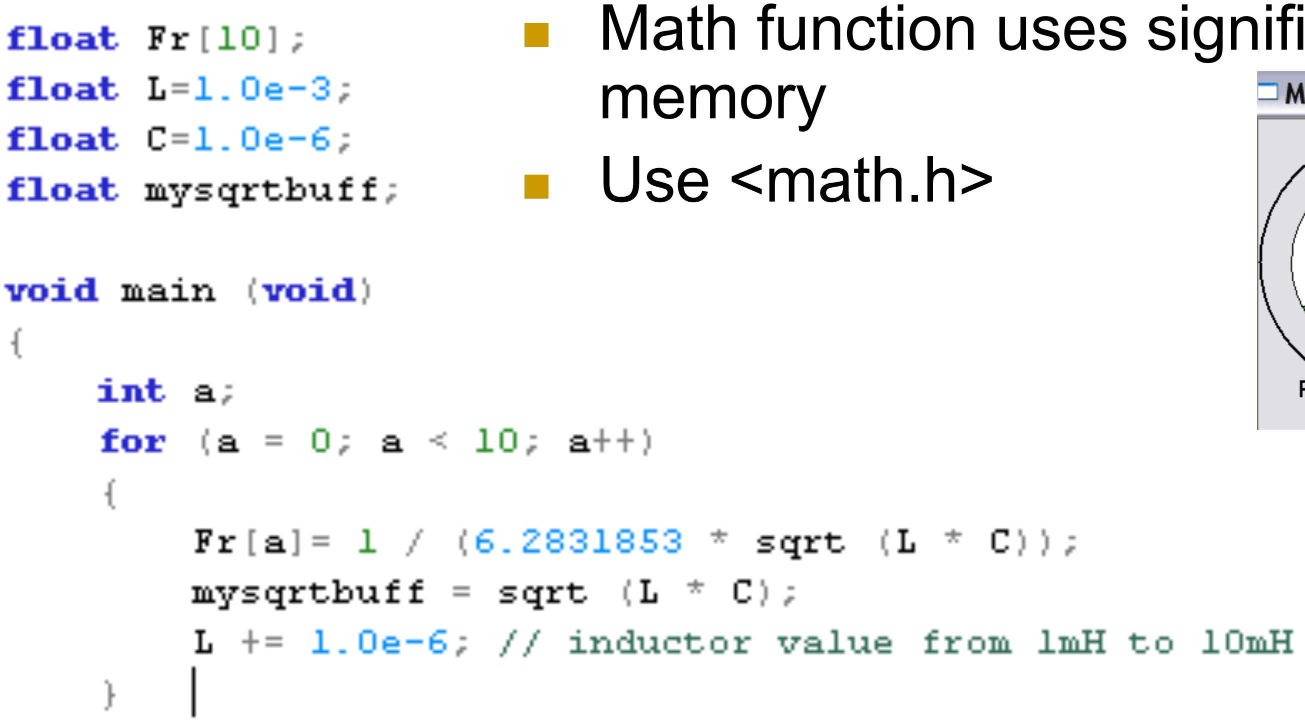
```
int  seed;
void main (void)
{
     TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)
     INTCON2bits.RBPU = 0; // enable PORTB internal pullups
     WPUBbits.WPUB0 = 1; // enable pull up on RB0
     ANSELH = 0x00; // AN8-12 are digital inputs (AN12 on RB0)
     TRISBbits.TRISB0 = 1; // PORTB bit 0
                                    // (connected to switch) is input (1)

     TRISB=0xFF;
     PORTD=0;
     seed = 1;
     while (1)
     {
             while (PORTBbits.RB0 == 0)
             {
                          seed++;
                          if (seed == 10)
                                       seed = 1;
                          PORTD = seed;
             }
     }
}
```



We can also use RAND()

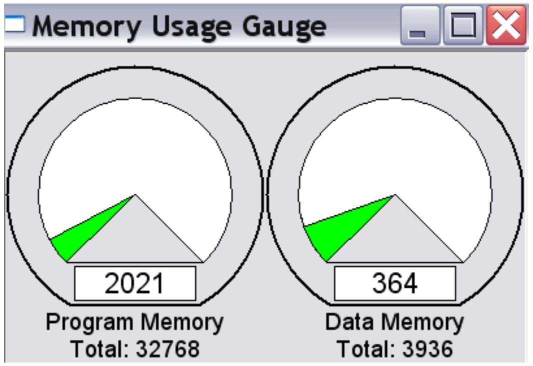
```
#include <p18cxxx.h>
#include <delays.h>
#include <stdlib.h>
/* Set configuration bits
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
void main (void)
{
        ADCON1 = 0x7F;                   // configure PORTS A and B as digital
                                         //        this might need to be changed depending
                                         //        on the microcontroller version.
        TRISB = 0;                       // configure PORTB for output
        PORTB = 0;                       // LEDs off
        srand(1);                // Sets the seed of the random number
        while ( 1 )              // repeat forever
        {
                Delay10KTCYx(50);        // wait 1/2 second
                PORTB = rand();          // display a random number
                                         // rand() returns a random value

        }
}
```

Example of using Math Functions



```
float Fr[10];
float L=1.0e-3;
float C=1.0e-6;
float mysqrtbuff;

void main (void)
{
    int a;
    for (a = 0; a < 10; a++)
    {
        Fr[a]= 1 / (6.2831853 * sqrt (L * C));
        mysqrtbuff = sqrt (L * C);
        L += 1.0e-6; // inductor value from 1mH to 10mH
    }
```

- Math function uses significant amount of memory
- Use <math.h>

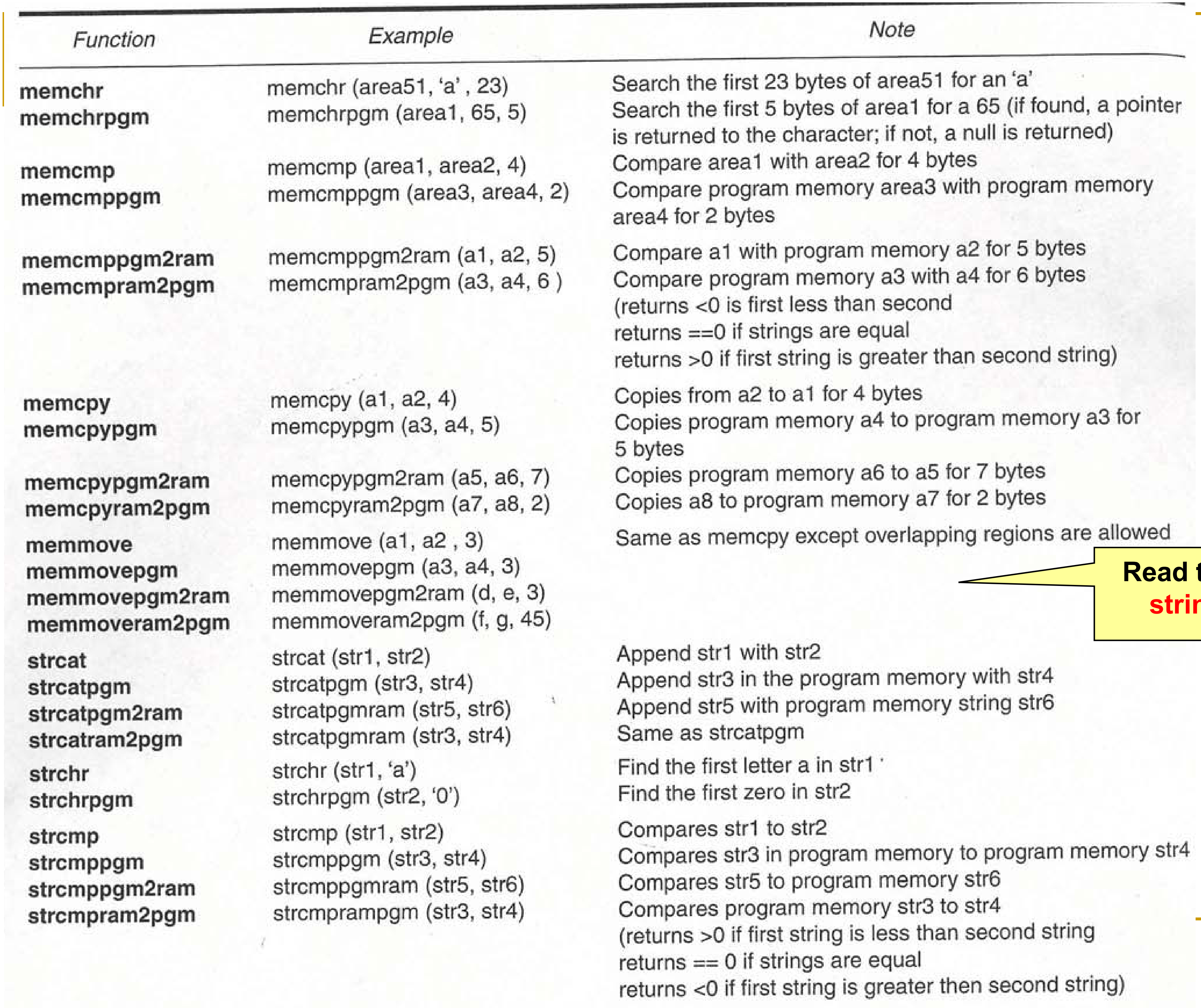


Memory Usage with math.h

Common Conversion Functions in <stdlib.h>

Function	Example	Note
atob	atob(buffer)	Converts the number from string form in buffer; returned as a byte signed number (+127 to -128)
atof	atof(buffer)	Converts the number from string form in buffer; returned as a floating point number
atoi	atoi(buffer)	Converts the number from string form in buffer; returned as a 16-bit signed integer (+ 32,767 to -32, 768)
atol	atol(buffer)	Converts the number form string format in buffer; returned as a 32-bit signed integer (+ 2, 147, 483, 647 to – 2, 417, 483, 648)
btoa	btoa(num, buffer)	Converts the signed byte into a string stored at buffer
itoa	itoa(num, buffer)	Converts the signed 16-bit integer to a string stored at buffer
Itol	itol(num, buffer)	Converts the 32-bit signed integer to a string stored at buffer
rand	rand()	Returns a 16 bit random number (0 to 32, 767)
srand	srand(seed)	Sets the seed values to 16-bit integer seed
tolower	tolower(letter)	Converts byte-sized character letter from uppercase; returned as lowercase
toupper	toupper(letter)	Converts byte-sized character letter from lowercase, returns uppercase
ultoa	ultoa(num, buffer)	Same as itol, except num is unsigned

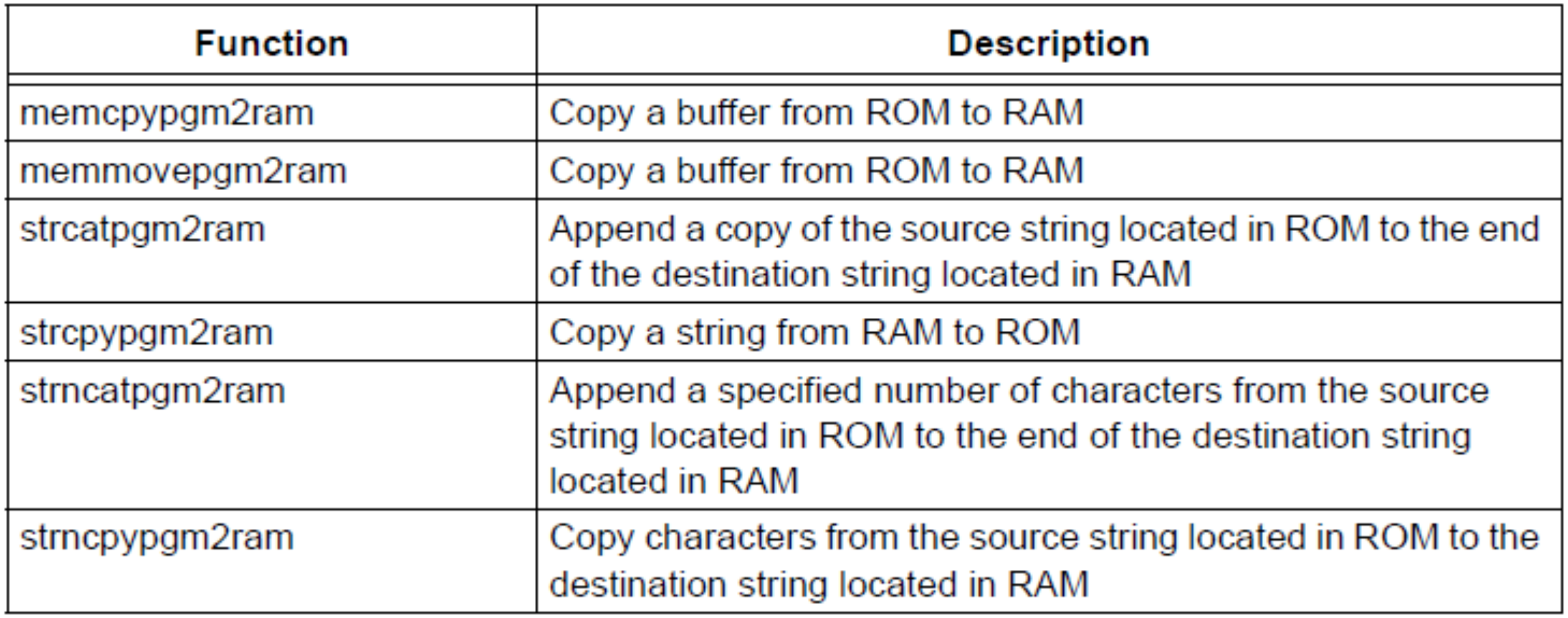
The C Library Reference Guide http://www.acm.uiuc.edu/webmonkeys/book/c_guide/



Function	*Example*	*Note*
memchr **memchrpgm**	memchr (area51, ‘a’ , 23) memchrpgm (area1, 65, 5)	Search the first 23 bytes of area51 for an ‘a’ Search the first 5 bytes of area1 for a 65 (if found, a pointer is returned to the character; if not, a null is returned)
memcmp **memcmppgm**	memcmp (area1, area2, 4) memcmppgm (area3, area4, 2)	Compare area1 with area2 for 4 bytes Compare program memory area3 with program memory area4 for 2 bytes
memcmppgm2ram **memcmpram2pgm**	memcmppgm2ram (a1, a2, 5) memcmpram2pgm (a3, a4, 6)	Compare a1 with program memory a2 for 5 bytes Compare program memory a3 with a4 for 6 bytes (returns <0 is first less than second returns ==0 if strings are equal returns >0 if first string is greater than second string)
memcpy **memcpypgm**	memcpy (a1, a2, 4) memcpypgm (a3, a4, 5)	Copies from a2 to a1 for 4 bytes Copies program memory a4 to program memory a3 for 5 bytes
memcpypgm2ram **memcpyram2pgm**	memcpypgm2ram (a5, a6, 7) memcpyram2pgm (a7, a8, 2)	Copies program memory a6 to a5 for 7 bytes Copies a8 to program memory a7 for 2 bytes
memmove **memmovepgm** **memmovepgm2ram** **memmoveram2pgm**	memmove (a1, a2 , 3) memmovepgm (a3, a4, 3) memmovepgm2ram (d, e, 3) memmoveram2pgm (f, g, 45)	Same as memcpy except overlapping regions are allowed
strcat **strcatpgm** **strcatpgm2ram** **strcatram2pgm**	strcat (str1, str2) strcatpgm (str3, str4) strcatpgmram (str5, str6) strcatpgmram (str3, str4)	Append str1 with str2 Append str3 in the program memory with str4 Append str5 with program memory string str6 Same as strcatpgm
strchr **strchrpgm**	strchr (str1, ‘a’) strchrpgm (str2, ‘0’)	Find the first letter a in str1 Find the first zero in str2
strcmp **strcmppgm** **strcmppgm2ram** **strcmpram2pgm**	strcmp (str1, str2) strcmppgm (str3, str4) strcmppgmram (str5, str6) strcmprampgm (str3, str4)	Compares str1 to str2 Compares str3 in program memory to program memory str4 Compares str5 to program memory str6 Compares program memory str3 to str4 (returns >0 if first string is less than second string returns == 0 if strings are equal returns <0 if first string is greater then second string)

Read these!
string.h

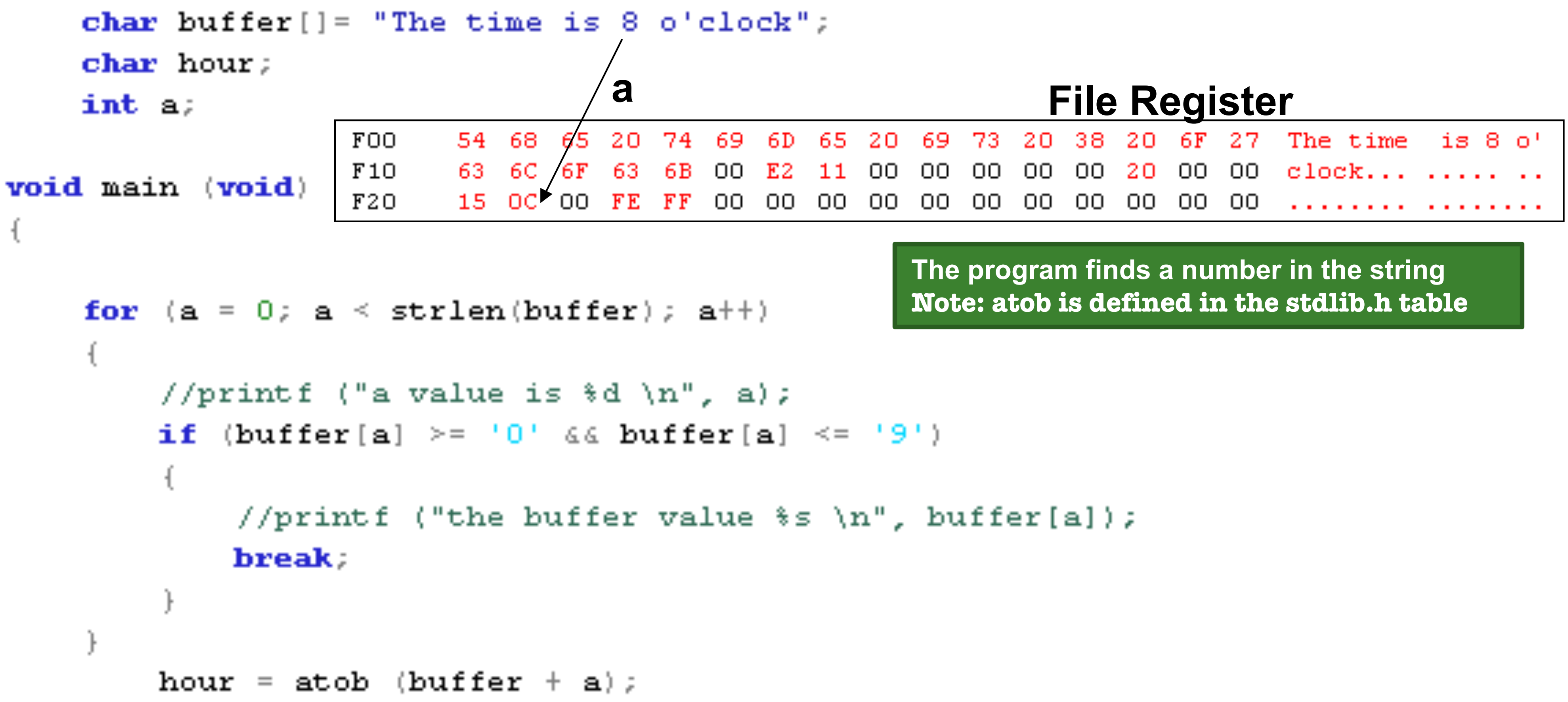
Copy data from program memory to data memory



Function	Description
memcpypgm2ram	Copy a buffer from ROM to RAM
memmovepgm2ram	Copy a buffer from ROM to RAM
strcatpgm2ram	Append a copy of the source string located in ROM to the end of the destination string located in RAM
strcpypgm2ram	Copy a string from RAM to ROM
strncatpgm2ram	Append a specified number of characters from the source string located in ROM to the end of the destination string located in RAM
strncpypgm2ram	Copy characters from the source string located in ROM to the destination string located in RAM

Example of <string.h> and <stlib.h>

- Using strlen() and atob()



```
  char buffer[]= "The time is 8 o'clock";
  char hour;
  int a;

void main (void)
{

  for (a = 0; a < strlen(buffer); a++)
  {
      //printf ("a value is %d \n", a);
      if (buffer[a] >= '0' && buffer[a] <= '9')
      {
          //printf ("the buffer value %s \n", buffer[a]);
          break;
      }
  }
      hour = atob (buffer + a);
```

a

File Register

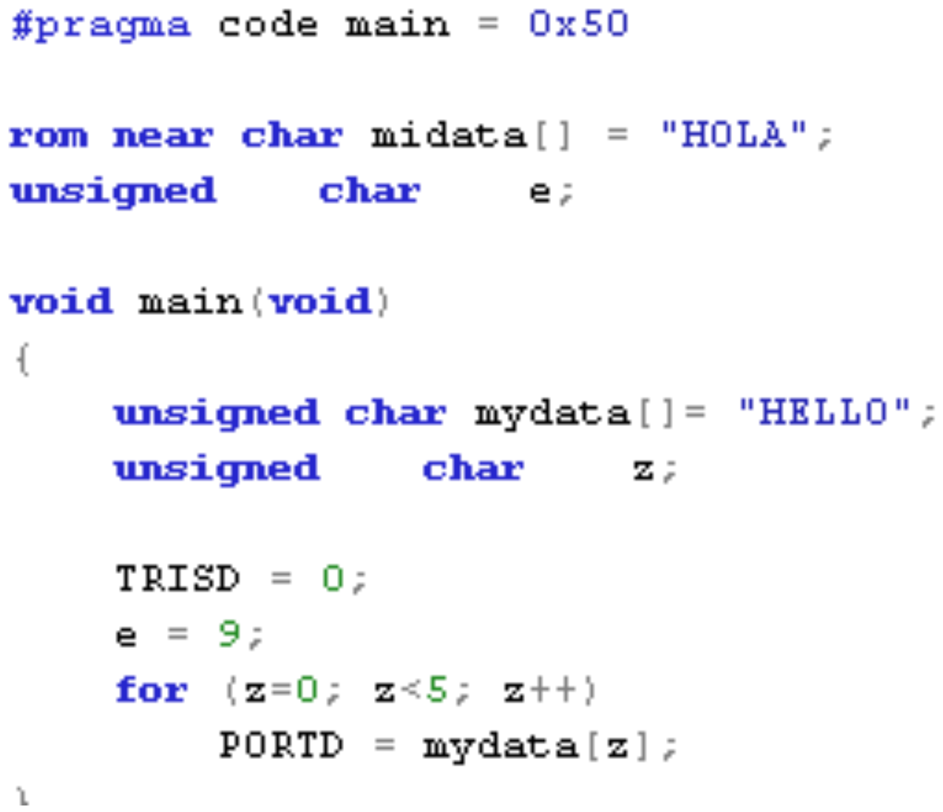
```
F00   54 68 65 20 74 69 6D 65 20 69 73 20 38 20 6F 27  The time  is 8 o'
F10   63 6C 6F 63 6B 00 E2 11 00 00 00 00 00 20 00 00  clock... ..... ..
F20   15 0C 00 FE FF 00 00 00 00 00 00 00 00 00 00     ........ ........
```

The program finds a number in the string
Note: atob is defined in the stdlib.h table

Understanding Data Storage Using C18/XC8 C compiler-Example

Answer the following questions (LAB):

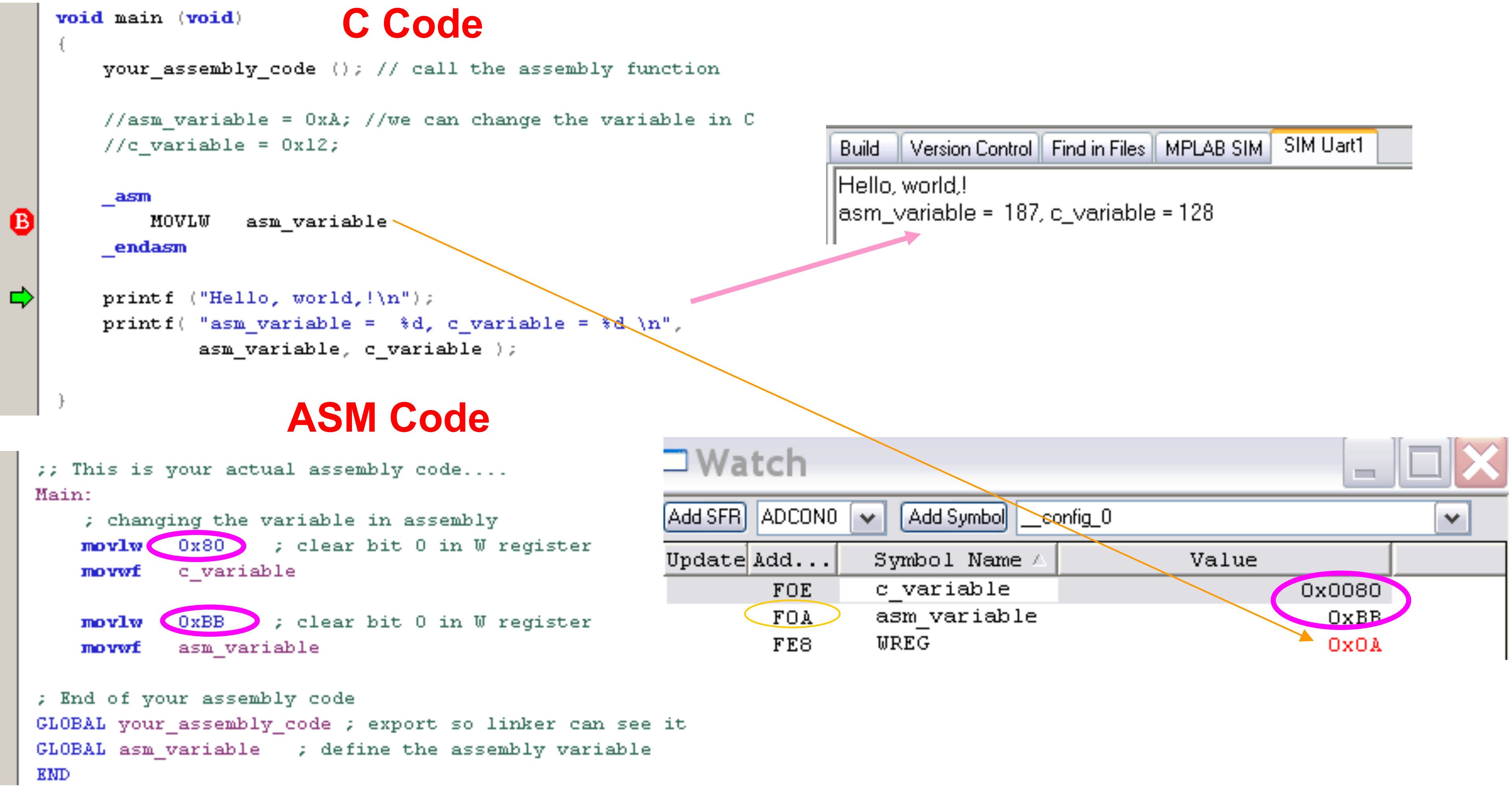
1- where is mydata stored? Which register?
2- Where is Z variable located at?
3- Where is e variable located at?
4- where is midata?
5- where does the main program start at?



```
#pragma code main = 0x50

rom near char midata[] = "HOLA";
unsigned    char    e;

void main(void)
{
    unsigned char mydata[]= "HELLO";
    unsigned    char    z;

    TRISD = 0;
    e = 9;
    for (z=0; z<5; z++)
        PORTD = mydata[z];
}
```

Program: Second_C

Passing Parameters Between C and ASM Codes



C Code

```
void main (void)
{
    your_assembly_code (); // call the assembly function

    //asm_variable = 0xA; //we can change the variable in C
    //c_variable = 0x12;

    _asm
        MOVLW    asm_variable
    _endasm

    printf ("Hello, world,!\n");
    printf( "asm_variable =  %d, c_variable = %d \n",
            asm_variable, c_variable );

}
```

```
Hello, world,!
asm_variable =  187, c_variable = 128
```

ASM Code

```
;; This is your actual assembly code....
Main:
    ; changing the variable in assembly
    movlw   0x80    ; clear bit 0 in W register
    movwf   c_variable

    movlw   0xBB    ; clear bit 0 in W register
    movwf   asm_variable

; End of your assembly code
GLOBAL your_assembly_code ; export so linker can see it
GLOBAL asm_variable    ; define the assembly variable
END
```

Update	Add...	Symbol Name	Value
	F0E	c_variable	0x0080
	F0A	asm_variable	0xBB
	FE8	WREG	0x0A

(Refer to Example Code: passing_parameters.c)

References

- Microchip.com
- Brey chapter 5
- Huang