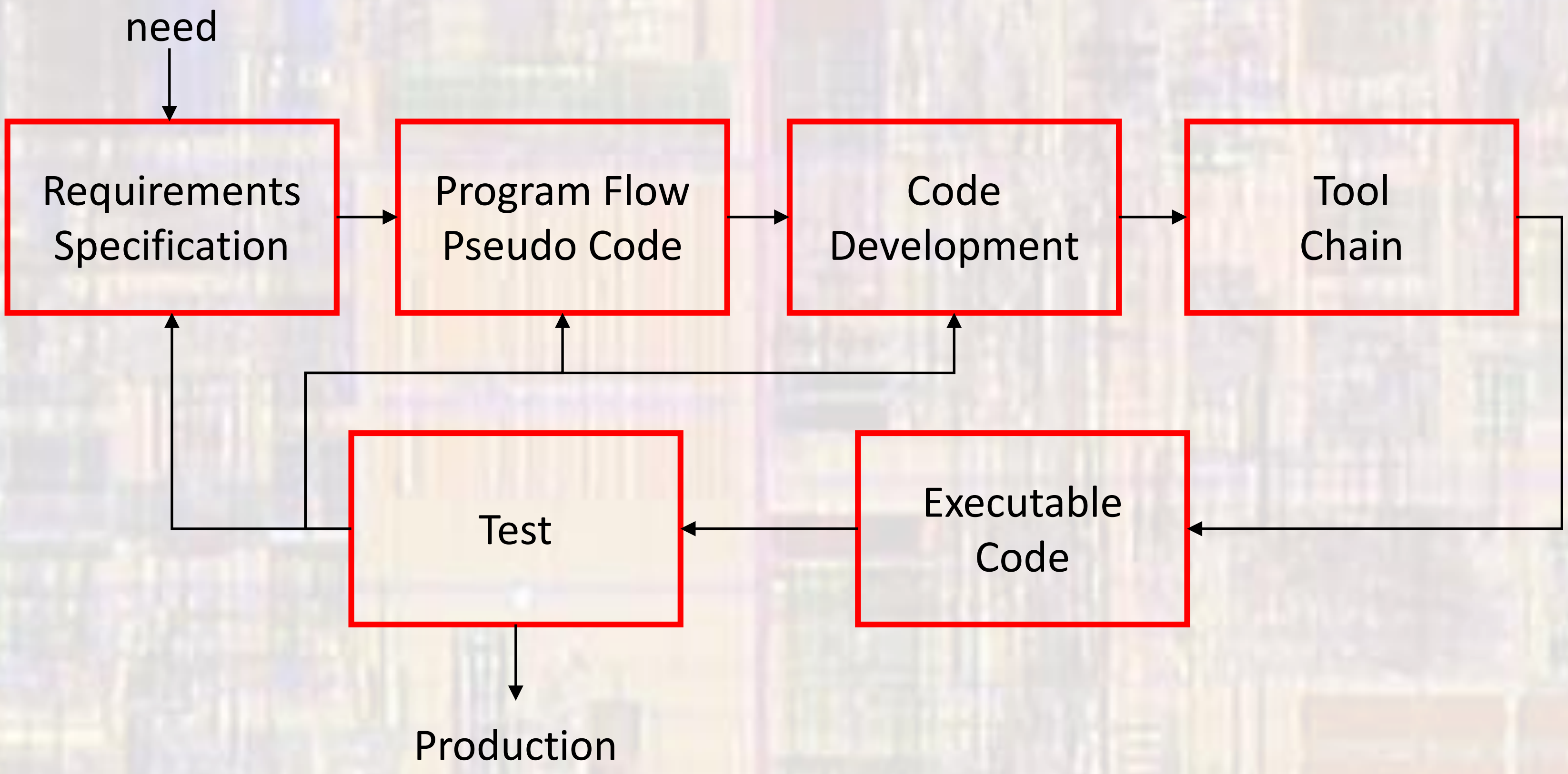
C Review I

Last updated 9/9/19

C Review I

- Development process



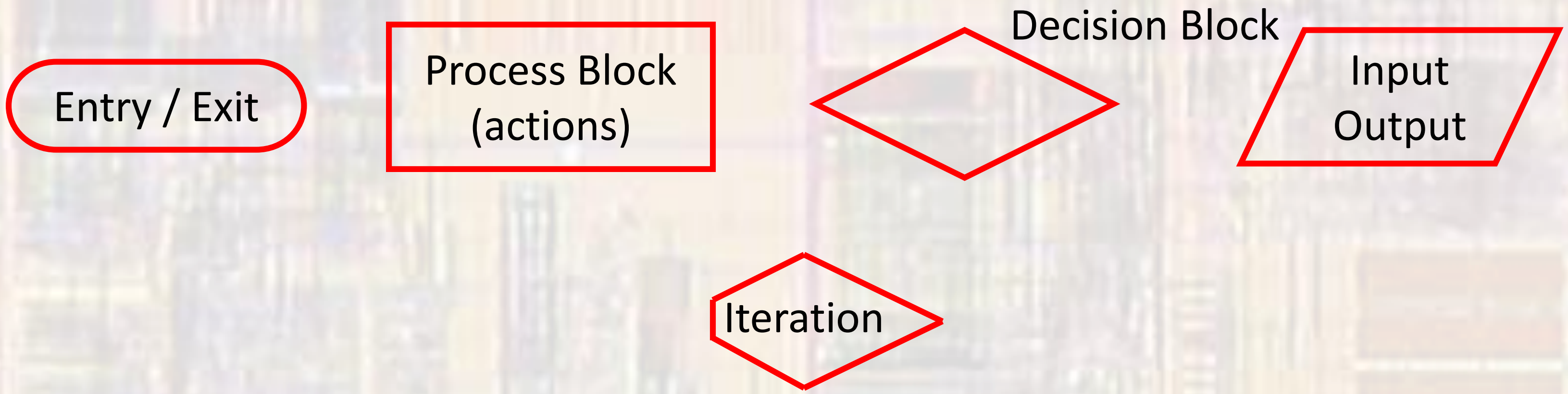
C Review I

- Requirements Specification
  - Identify what the system must do to solve the problem
    - Over specify:
      - Higher cost
      - Longer development time
    - Under specify
      - Don't solve problem
      - Customer does not accept the solution
  - No design solutions should be assumed at this stage
    - Maximize the design space

C Review I

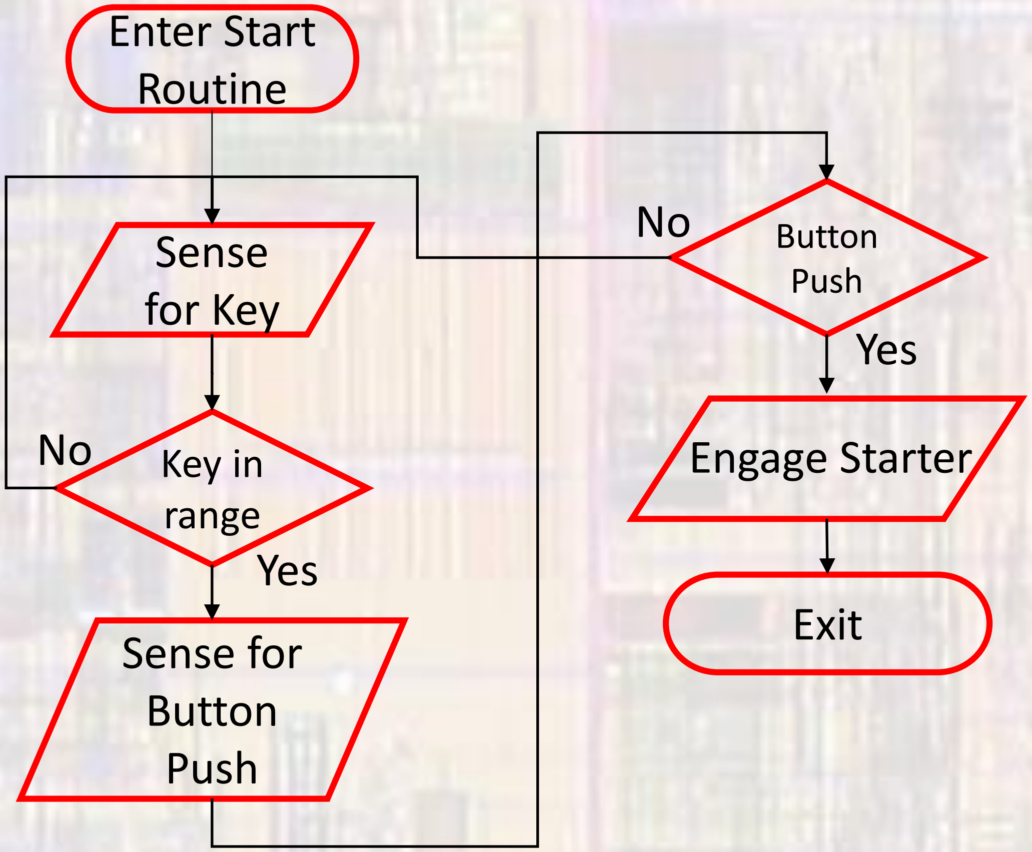
- Program Flow / Pseudo Code
  - Hierarchical system design
  - Up/Down sub-system design
  - Focus on general structure – not too specific

  - Basic flow diagram blocks



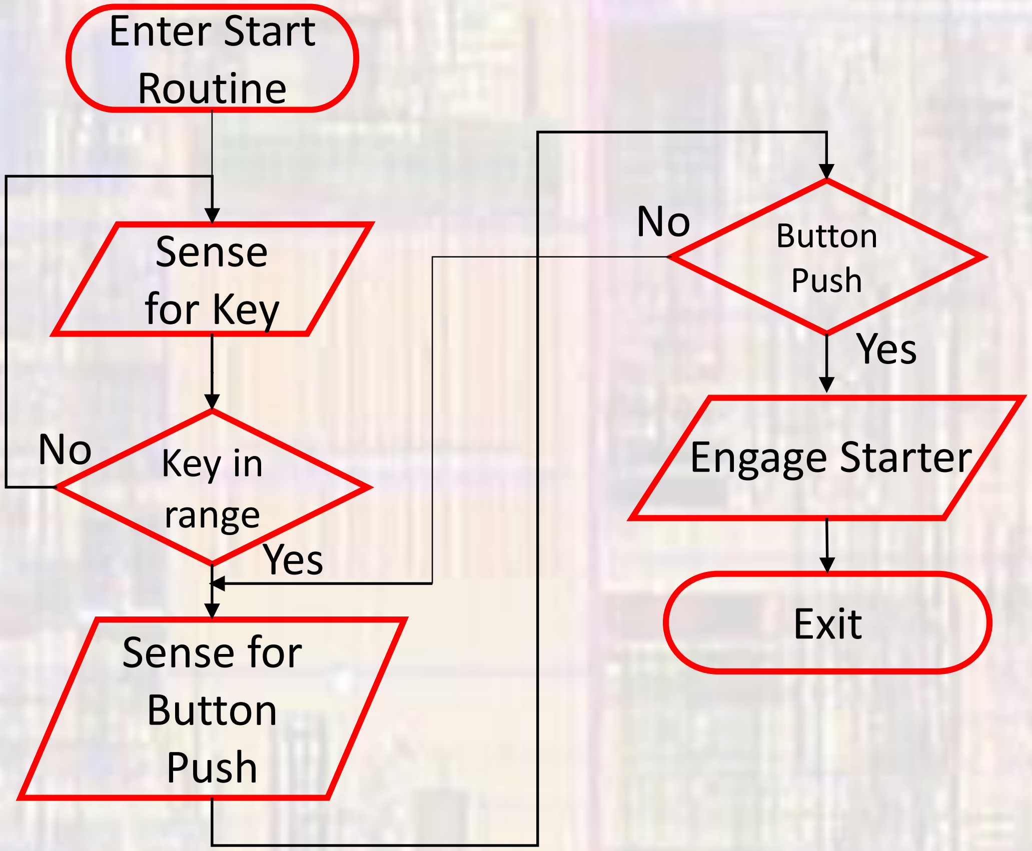
C Review I

- Program Flow Diagram – Keyless Start



C Review I

- Program Flow Diagram – Keyless Start



Why not this

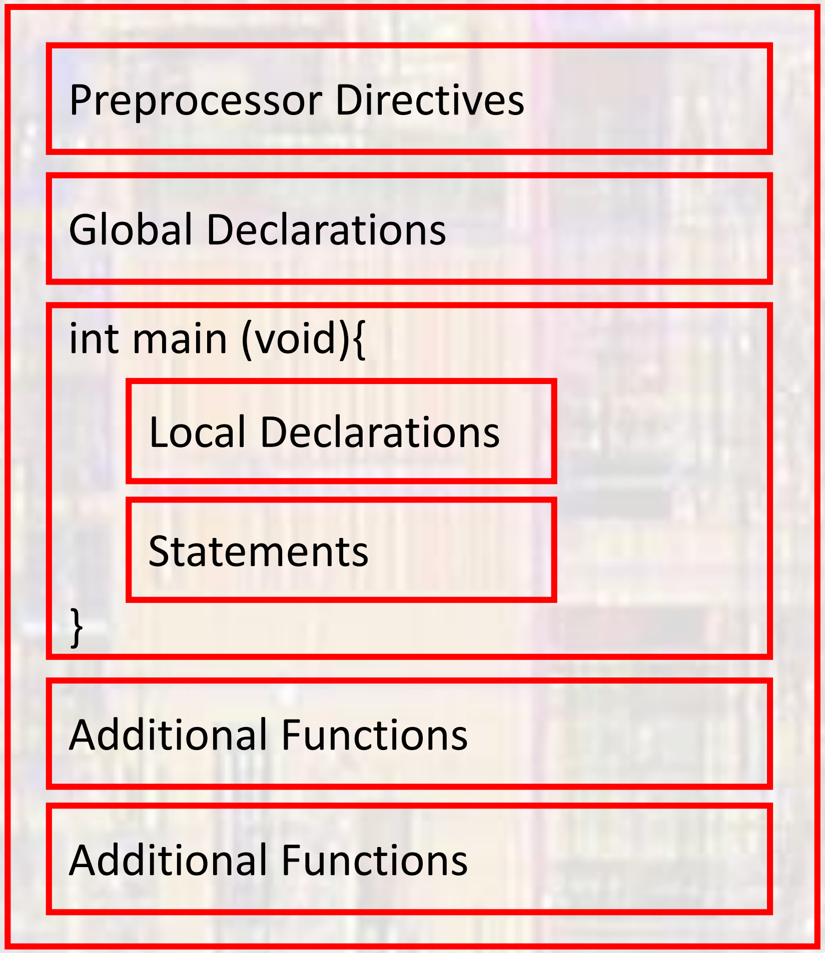
C Review I

- Pseudo Code – Keyless Start

```
In an infinite loop:

If Key present
    If button pushed
        engage starter
    end
end
```

C Review I

- C program – first look



C Review I

- Preprocessor directives
  - Provide information to the tool chain
    - Additional files to include
    - Name definitions
    - Constant definitions
    - Always start with a #

C Review I

- Preprocessor directives
  - Examples -

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

- Include the contents of library file stdio.h along with my code

```
#define PI 3.14159
```

- Everywhere I used PI, replace it with 3.14159

```
#define LEDMASK 0x02
```

- Identify which bit an LED is in an 8 bit word
- Allows changes in 1 place instead of all through the code

C Review I

- Global Declarations

  - Define variables that can be seen throughout the program
  - Should only define these when absolutely necessary
    - typedef, structures, shared peripheral values, interrupts

  - Examples

int age
- Define a global variable – age

float InterestRate = 0.012
- Define a global variable InterestRate and initialize it to 0.012

C Review I

- Main
  - Code section containing your top level program code
  - Program flow is controlled by main
  - Required
  - Can only be 1 main in your program (project)
    - Use “exclude from build” to allow multiple files with a main in your project
  - Local Declarations
    - Define variables that can be seen inside of main
  - Statements
    - The top level program code
      - Should be used for control, not calculations, I/O, …

C Review I

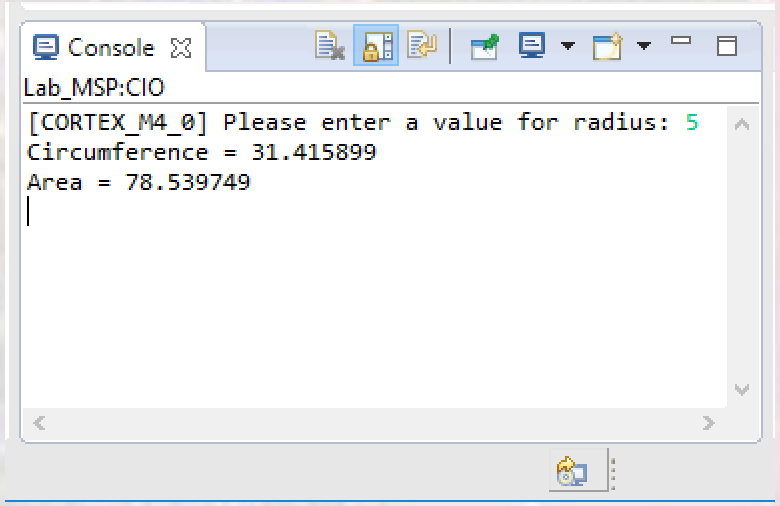
- Other functions

  - Functions are sections of code defined to do a specific task
  - They are called by main or other functions
  - Can take values in and provide values out

  - Good programming uses main for control and uses functions for getting things done

C Review I

- Simple MSP432 Program



```
[CORTEX_M4_0] Please enter a value for radius: 5
Circumference = 31.415899
Area = 78.539749
```

```
/*
 * circle_msp.c
 *
 *  Created on: Jul 23, 2018
 *      Author: johnsontimoj
 */
/////////////////////////
//
// example file for showing msp operation
//
// calculates the area and circumference of a circle
//
// inputs: radius
// outputs: area, circumference
//
///////////////////////////

// includes
#include "msp.h"
#include <stdio.h>
#define PI 3.14159

int main(void){

   //  Local variables
   float radius;
   float circumference;
   float area;

   //   Get input for radius
   printf("Please enter a value for radius: ");
   scanf("%f", &radius);

   //   Calculate circumference and area
   circumference = 2 * PI * radius;
   area = PI * radius * radius;

   // Output results
   printf("Circumference = %f\n", circumference);
   printf("Area = %f\n", area);

   return 0;
}
```

C Review I

- Simple MSP432 Program

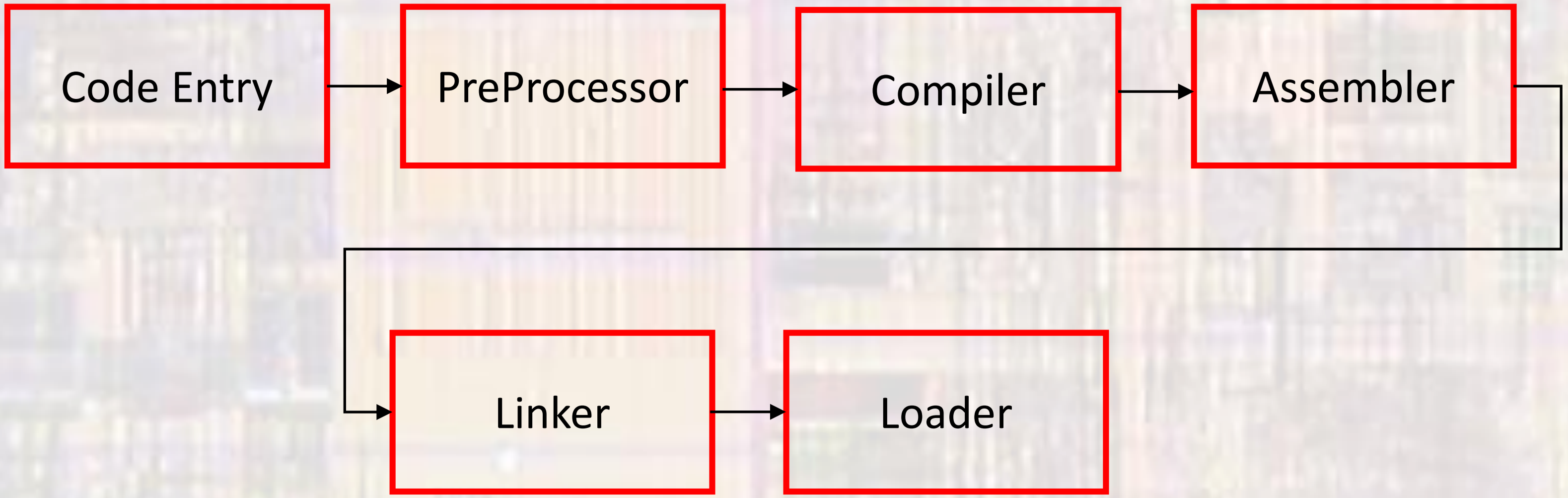
```c
/*
 * circle_msp_2.c
 *
 *  Created on: Jul 23, 2018
 *      Author: johnsontimoj
 */
/////////////////////////
//
// example file for showing msp operation
//
// calculates the area and circumference of a circle
// using functions
//
// inputs: radius
// outputs: area, circumference
//
///////////////////////////

// includes
#include "msp.h"
#include <stdio.h>
#define PI 3.14159

float get_radius(void);
float calc_circum(float r);
float calc_area(float r);
void pr_results(float c, float a);
```

```c
int main(void){

   //  Local variables
   float radius;
   float circumference;
   float area;

   //   Get input for radius
   radius = get_radius();

   //   Calculate circumference and area
   circumference = calc_circum(radius);
   area = calc_area(radius);

   // Output results
   pr_results(circumference, area);

   return 0;
}
```

```c
float get_radius(void){
    //   Get input for radius
    float rad;
    printf("Please enter a value for radius: ");
    scanf("%f", &rad);
    return rad;
}

float calc_circum(float r){
    return (2 * PI * r);
}

float calc_area(float r){
    return (PI * r * r);
}

void pr_results(float c, float a){
    printf("Circumference = %f\n", c);
    printf("Area = %f\n", a);
    return;
}
```

C Review I

- Tool Chain



C Review I

- Tool Chain
  - CodeEntry
    - filename.c
    - Text editor
    - Integrated Development Environment
      - Code Composer
      - Eclipse

Code Entry

```
a = a + 5;
```

C Review I

- Tool Chain
  - Preprocessor
    - Deals with any commands starting with #
    - Tells the tool chain to include additional libraries
    - Replaces any “defines” throughout the code
    - Expands macros throughout the code
    - Manages any conditional defines

$\boxed{\text{PreProcessor}}$

C Review I

- Tool Chain

Compiler

  - Compiler

    - Converts c-code to assembly language

    - Assembly language
      - Architecture specific programming language
      - Direct access to specific registers, commands, memory

```
ld    R2, 0x2000;          // load register R2 from memory location 0x2000
ldi   R3, 0x05;            // load R3 with 5
add R2, R3;                // add the values of R2 and R3 and store in R2
st     R2, 0x2000;         // store R2 (result) in memory location 0x2000
```

simplified

C Review I

- Tool Chain
  - Assembly Language

Compiler

```
void main(void)
{
    // create and print some variables and addresses

// Local Variables
uint32_t a = 1;
uint32_t b = 1;
uint32_t c;


    // Board Setup
WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer

c = a + b;
printf("%i", c);
}
```

```
main():
push      {r1, r2, r3, lr}
                  uint32_t a = 1;
movs      r0, #1
str       r0, [sp]
                  uint32_t b = 1;
movs      r0, #1
str       r0, [sp, #4]
                  WDTCTL = WDTPW | WDTHOLD;

ldr       r1, [pc, #0x1c]
mov.w     r0, #0x5a80
strh      r0, [r1]
                  c = a + b;
ldr       r0, [sp, #4]
ldr       r1, [sp]
adds      r0, r0, r1
str       r0, [sp, #8]
                  printf("%i", c);
ldr       r1, [sp, #8]
adr       r0, #4
bl        #0x2f18
}
```

C Review I

- Tool Chain

Assembler

- Assembler
  - Converts assembly language to machine language
  - Result is an object file (file.o)
  - Machine language
    - Part specific programming language
    - Binary representation that the processor understands

```
1011 1010 1001 1000;        // load register R2 from memory location 0x2000
1011 1110 1101 1010;        // load R3 with 5
1101 1010 1001 1000;        // add the values of R2 and R3 and store in R2
1001 1000 1011 1010;        // store R2 (result) in memory location 0x2000
```

simplified

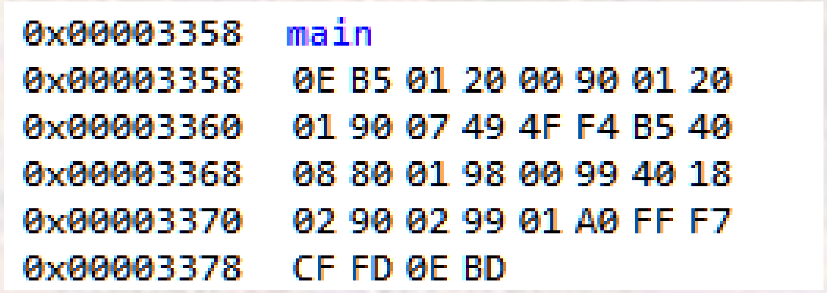
C Review I

- Tool Chain
  - Machine Language

Assembler

```
main():
push      {r1, r2, r3, lr}
              uint32_t a = 1;
movs      r0, #1
str       r0, [sp]
              uint32_t b = 1;
movs      r0, #1
str       r0, [sp, #4]
              WDTCTL = WDTPW | WDTHOLD;

ldr       r1, [pc, #0x1c]
mov.w     r0, #0x5a80
strh      r0, [r1]
              c = a + b;
ldr       r0, [sp, #4]
ldr       r1, [sp]
adds      r0, r0, r1
str       r0, [sp, #8]
              printf("%i", c);
ldr       r1, [sp, #8]
adr       r0, #4
bl        #0x2f18
}
```

Address Machine Code(hex)



```
0x00003358  main
0x00003358  0E B5 01 20 00 90 01 20
0x00003360  01 90 07 49 4F F4 B5 40
0x00003368  08 80 01 98 00 99 40 18
0x00003370  02 90 02 99 01 A0 FF F7
0x00003378  CF FD 0E BD
```

C Review I

- Tool Chain

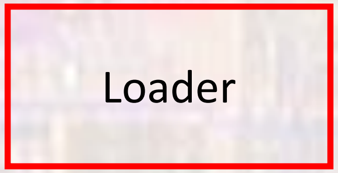
  - Linker
    - Combines the machine language code from your program with all included libraries
    - Configures all the code in memory
      - Aligns code segments
      - Makes connections where necessary (function calls)
      - Assigns variables spots in memory
    - Creates an executable file - file.out (file.exe for windows systems)

C Review I

- Tool Chain

  - Loader (programmer)

    - Creates whatever environment is necessary on the executing machine
    - Loads the executable program
    - Starts the program



C Review I

- Putting it all together

```
**** Build of configuration Debug for project Lab_MSP ****

"C:\\ti\\ccsv6\\utils\\bin\\gmake" -k all
'Building file: ../circle_msp.c'
'Invoking: MSP432 Compiler'
"C:/ti/ccsv6/tools/compiler/arm_15.12.3.LTS/bin/armcl" -mv7M4 --code_state=16 --float_support=FPv4SPD16 -me --
include_path="C:/ti/ccsv6/ccs_base/arm/include" --include_path="C:/ti/ccsv6/ccs_base/arm/include/CMSIS" --
include_path="C:/ti/ccsv6/tools/compiler/arm_15.12.3.LTS/include" --advice:power=all -g --gcc --define=__MSP432P401R__ --
define=TARGET_IS_MSP432P4XX --define=ccs --diag_warning=225 --diag_wrap=off --display_error_number --abi=eabi --preproc_with_compile --
preproc_dependency="circle_msp.d"  "../circle_msp.c"
'Finished building: ../circle_msp.c'
' '
'Building target: Lab_MSP.out'
'Invoking: MSP432 Linker'
"C:/ti/ccsv6/tools/compiler/arm_15.12.3.LTS/bin/armcl" -mv7M4 --code_state=16 --float_support=FPv4SPD16 -me --advice:power=all -g --gcc --
define=__MSP432P401R__ --define=TARGET_IS_MSP432P4XX --define=ccs --diag_warning=225 --diag_wrap=off --display_error_number --abi=eabi -
z -m"Lab_MSP.map" --stack_size=512 --heap_size=1024 -i"C:/ti/ccsv6/ccs_base/arm/include" -i"C:/ti/ccsv6/tools/compiler/arm_15.12.3.LTS/lib" -
i"C:/ti/ccsv6/tools/compiler/arm_15.12.3.LTS/include" --reread_libs --warn_sections --diag_wrap=off --display_error_number --
xml_link_info="Lab_MSP_linkInfo.xml" --rom_model -o "Lab_MSP.out" "./circle_msp.obj" "./startup_msp432p401r_ccs.obj"
"./system_msp432p401r.obj" "../msp432p401r.cmd"  -llibc.a
<Linking>
remark #10371-D: (ULP 1.1) Detected no uses of low power mode state changing instructions
remark #10372-D: (ULP 4.1) Detected uninitialized Port 1 in this project. Recommend initializing all unused ports to eliminate wasted current
consumption on unused pins.
…
'Finished building target: Lab_MSP.out'
' '

**** Build Finished ****
```

C Review I

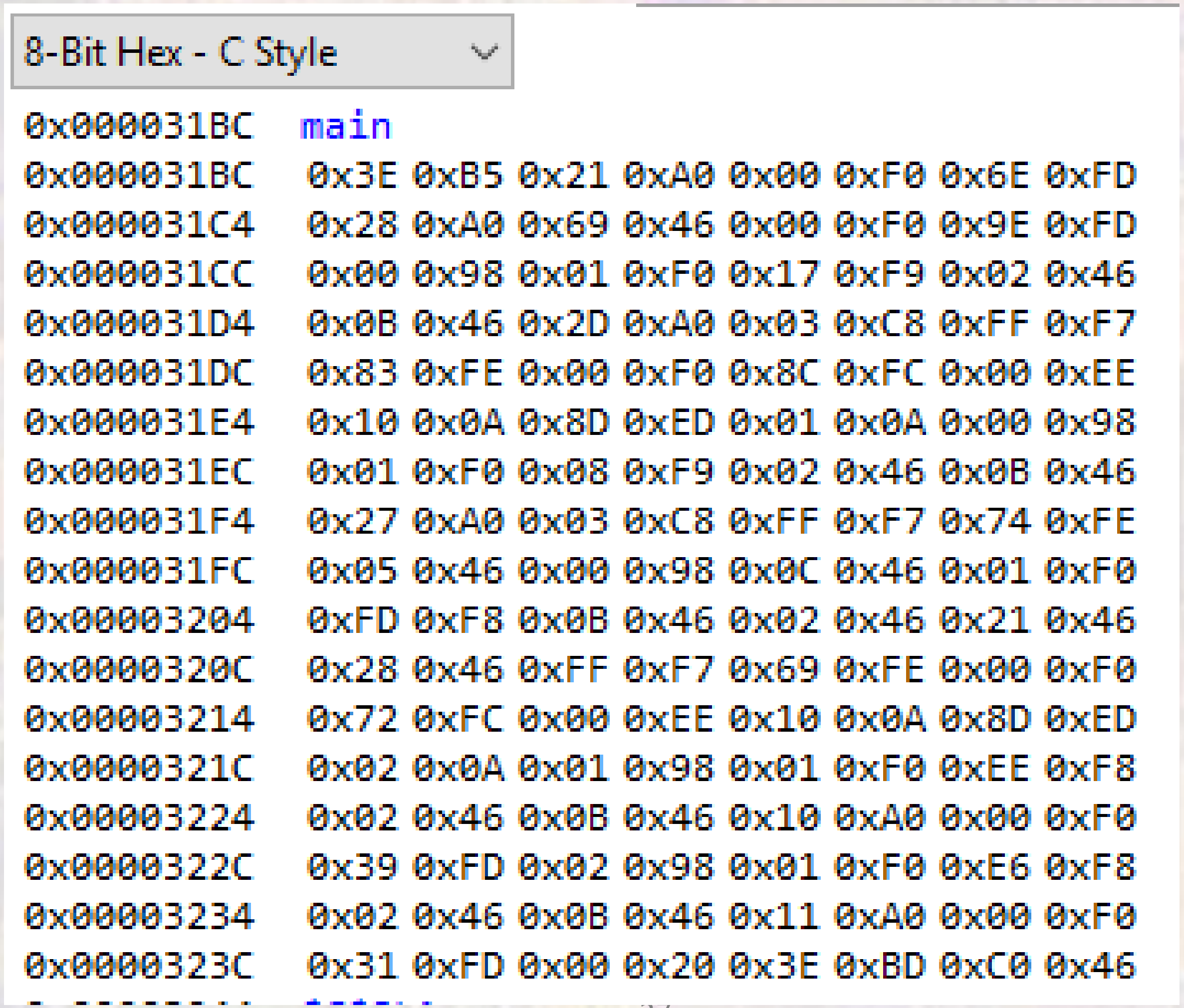
- Assembly for circle_msp.c

```
main():
000031bc:   B53E            push     {r1, r2, r3, r4, r5, lr}
31          printf("Please enter a value for radius: ");
000031be:   A021            adr      r0, #0x84
000031c0:   F000FD6E        bl       #0x3ca0
32          scanf("%f", &radius);
000031c4:   A028            adr      r0, #0xa0
000031c6:   4669            mov      r1, sp
000031c8:   F000FD9E        bl       #0x3d08
35          circumference = 2 * PI * radius;
000031cc:   9800            ldr      r0, [sp]
000031ce:   F001F917        bl       #0x4400
000031d2:   4602            mov      r2, r0
000031d4:   460B            mov      r3, r1
000031d6:   A02D            adr      r0, #0xb4
000031d8:   C803            ldm      r0, {r0, r1}
000031da:   F7FFFE83        bl       #0x2ee4
000031de:   F000FC8C        bl       #0x3afa
000031e2:   EE000A10        vmov     s0, r0
000031e6:   ED8D0A01        vstr     s0, [sp, #4]
```

```
36          area = PI * radius * radius;
000031ea:   9800            ldr      r0, [sp]
000031ec:   F001F908        bl       #0x4400
000031f0:   4602            mov      r2, r0
000031f2:   460B            mov      r3, r1
000031f4:   A027            adr      r0, #0x9c
000031f6:   C803            ldm      r0, {r0, r1}
000031f8:   F7FFFE74        bl       #0x2ee4
000031fc:   4605            mov      r5, r0
000031fe:   9800            ldr      r0, [sp]
00003200:   460C            mov      r4, r1
00003202:   F001F8FD        bl       #0x4400
00003206:   460B            mov      r3, r1
00003208:   4602            mov      r2, r0
0000320a:   4621            mov      r1, r4
0000320c:   4628            mov      r0, r5
0000320e:   F7FFFE69        bl       #0x2ee4
00003212:   F000FC72        bl       #0x3afa
00003216:   EE000A10        vmov     s0, r0
0000321a:   ED8D0A02        vstr     s0, [sp, #8]
39          printf("Circumference = %f\n", circumference);
0000321e:   9801            ldr      r0, [sp, #4]
00003220:   F001F8EE        bl       #0x4400
00003224:   4602            mov      r2, r0
00003226:   460B            mov      r3, r1
00003228:   A010            adr      r0, #0x40
0000322a:   F000FD39        bl       #0x3ca0
40          printf("Area = %f\n", area);
0000322e:   9802            ldr      r0, [sp, #8]
00003230:   F001F8E6        bl       #0x4400
00003234:   4602            mov      r2, r0
00003236:   460B            mov      r3, r1
00003238:   A011            adr      r0, #0x44
0000323a:   F000FD31        bl       #0x3ca0
42          return 0;
```

C Review I

- Machine Language for circle_msp.c



8-Bit Hex - C Style

```
0x000031BC   main
0x000031BC   0x3E 0xB5 0x21 0xA0 0x00 0xF0 0x6E 0xFD
0x000031C4   0x28 0xA0 0x69 0x46 0x00 0xF0 0x9E 0xFD
0x000031CC   0x00 0x98 0x01 0xF0 0x17 0xF9 0x02 0x46
0x000031D4   0x0B 0x46 0x2D 0xA0 0x03 0xC8 0xFF 0xF7
0x000031DC   0x83 0xFE 0x00 0xF0 0x8C 0xFC 0x00 0xEE
0x000031E4   0x10 0x0A 0x8D 0xED 0x01 0x0A 0x00 0x98
0x000031EC   0x01 0xF0 0x08 0xF9 0x02 0x46 0x0B 0x46
0x000031F4   0x27 0xA0 0x03 0xC8 0xFF 0xF7 0x74 0xFE
0x000031FC   0x05 0x46 0x00 0x98 0x0C 0x46 0x01 0xF0
0x00003204   0xFD 0xF8 0x0B 0x46 0x02 0x46 0x21 0x46
0x0000320C   0x28 0x46 0xFF 0xF7 0x69 0xFE 0x00 0xF0
0x00003214   0x72 0xFC 0x00 0xEE 0x10 0x0A 0x8D 0xED
0x0000321C   0x02 0x0A 0x01 0x98 0x01 0xF0 0xEE 0xF8
0x00003224   0x02 0x46 0x0B 0x46 0x10 0xA0 0x00 0xF0
0x0000322C   0x39 0xFD 0x02 0x98 0x01 0xF0 0xE6 0xF8
0x00003234   0x02 0x46 0x0B 0x46 0x11 0xA0 0x00 0xF0
0x0000323C   0x31 0xFD 0x00 0x20 0x3E 0xBD 0xC0 0x46
```

C Review I

- C Program Structure
  - A C program is composed of a series of functions
  - Main is the top level function in C
    - One and only one main function
    - Main may or may not call other functions

C Review I

- Function Purpose – simplified view

  - Receive zero or more pieces of data (parameters)
  - Operate on the data
  - Potentially have a side effect
  - Return one piece of data (return value)

  - Allow one piece of code to be reused with different inputs
  - Allows function libraries to reuse common code
    - # include <stdio.h>

C Review I

- User Defined Functions
  - Declaration
  - Definition
  - Call

```
// Function Declarations (prototypes)
int greeting(void);

int main(void){
   …
   greeting(void);      // function call
   return 0;
}

// Function Definition
int greeting(void){
   printf("Hello EE2920");
   return;
}
```

return type

C Review I

- User Defined Functions

```
// Function Declarations (prototypes)
void greeting(void);


int main(void){
   …
   greeting(void);
   return 0;
}


// Function Definition
void greeting(void){
   printf("Hello EE1910");
   return;
}
```

declaration must come before the first use of the function

If no data is passed to the function we can use (void) or ()

This function only has a side effect

Even if nothing is being returned we should include a return

C Review I

- User Defined Functions

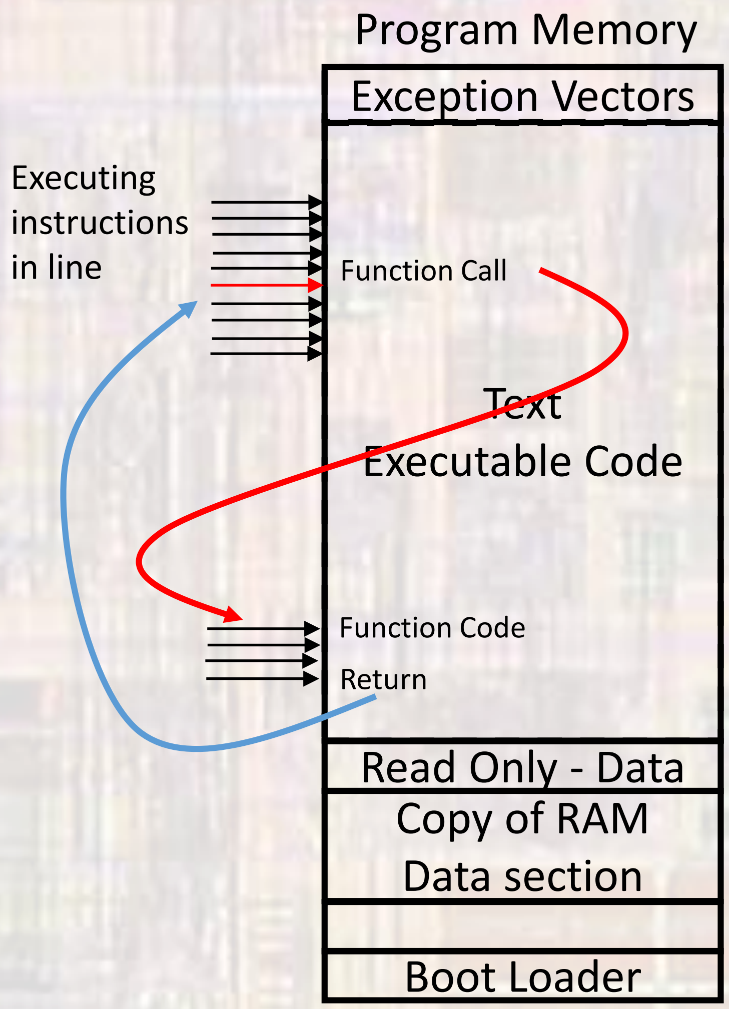
```
int vol(float length, float width, float height);
…

int main(void){
   float volume;
   // enter W, L, H
   …
  volume = vol(L, W, H);
  return 0;
}

int vol(float length, float width, float height){
  return (length * width * height);
}
```

Actual Parameters

Formal Parameters

C Review I

- Functions and Memory
  - Function call transfers execution to a separate section of code (function)
  - When done, the function returns to the next line of code
  - Multiple calls to the same function transfer execution to the same location
    - One copy of the function
    - Must be able to separate data from different calls



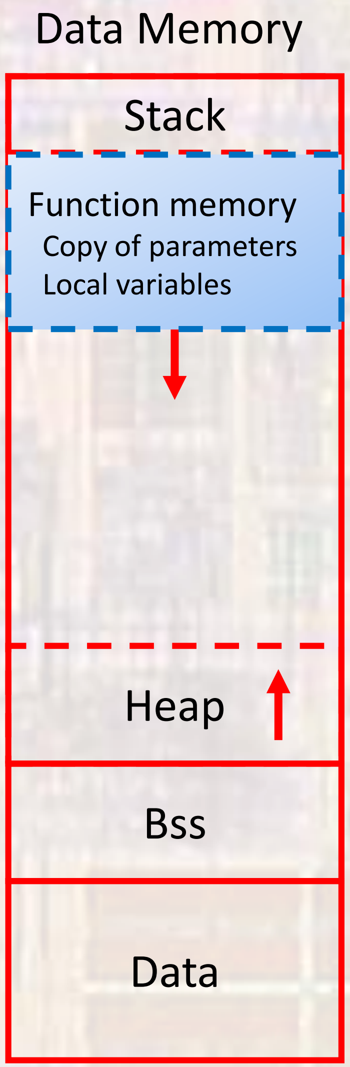
C Review I

- Functions and Memory

  - Function call creates a space in the stack
    - copy of the actual parameters
    - any function variables (local variables)

  - Function operates in this newly created space (scope)

  - When the function returns, the space is reclaimed (not necessarily erased but no longer available)



C Review I

- Functions and Memory

```
void update_acct(float balance, float int_rate);

int main(void){
   float bal;
   float ir;
    …
   printf(“balance is: %f”, bal);
   update_acct(bal, ir);
   printf(“balance is: %f”, bal);
   return 0;
}

void update_acct(float balance, float int_rate){
   balance = balance + balance * int_rate;
   printf(“balance is: %f”, balance);
   return;
}
```

with the original bal of 10,000 and a 10% ir – what does this print

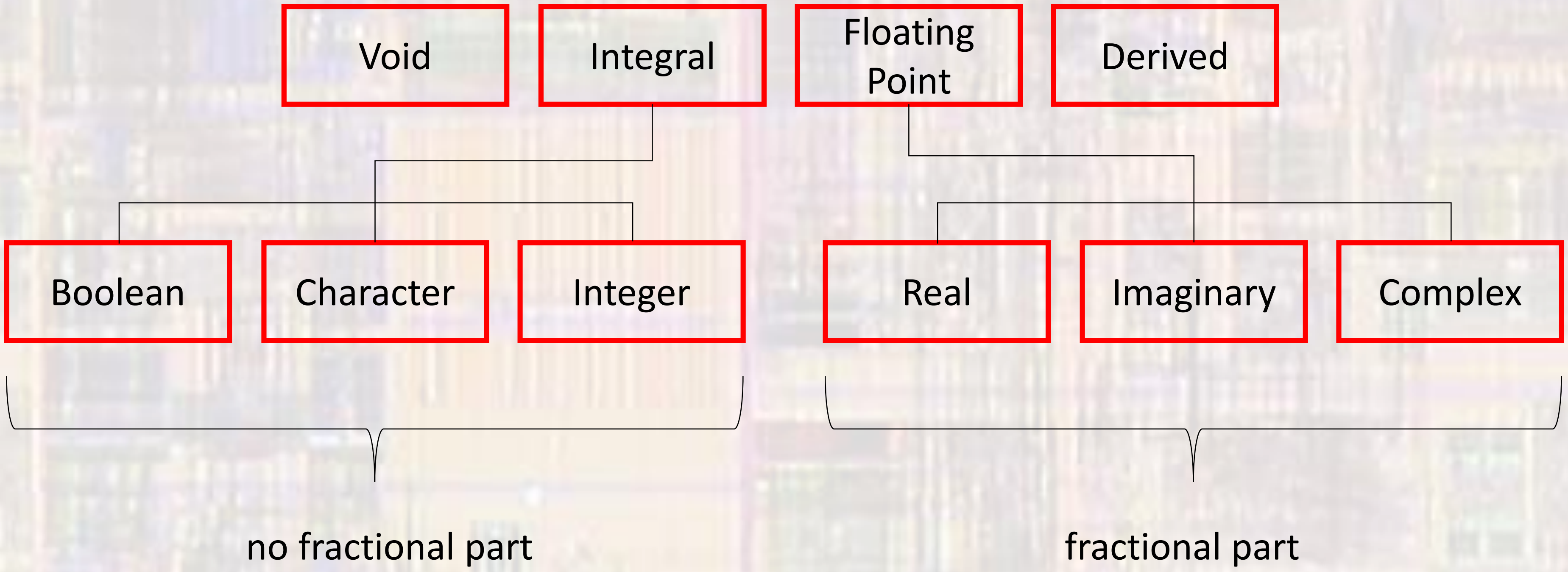
C Review I

- Functions and Memory

```
int main(void){
   float bal;
   float ir;
    …
   printf("balance is: %.2f\n", bal);
   update_acct(bal, ir);
   printf("balance is: %.2f\n", bal);
   return 0;
}

void update_acct(float balance, float int_rate){
   balance = balance + balance * int_rate;
   printf("balance is: %.2f\n", balance);
   return;
}
```

C Review I

- C Types



C Review I

- C Types - void

  - No values
  - No defined operations

  - Used when we want to indicate that nothing is here

  - Examples

```
MyFunction(void);
// call a function with no input parameters

void YourFunction(int val){    …
// indicate that a function returns nothing
```

C Review I

- C Types – char - character

  - ASCII - 128 values
    - a,b,c,1,2,3,$,%,*, …
    - English language characters
  - Unicode – millions of values

  - Stored in the computer as integers
  - Same operations as integers
  - Become characters when visualized
  - Require a single quote

  - Examples

```
char initial1 = ‘t’;
char initial2 = ‘j’;

printf(“%c%c”, initial1, initial2);
// print  - tj


printf(“%c”, (initial1 - initial2) );
// print – (line feed)
// 116 – 106 = 10 → linefeed
```

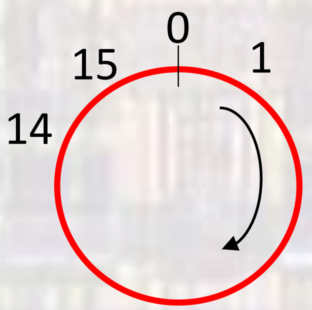
C Review I

- C Types – int - integer
  - Values are system dependent
    - integers only
    - 2, 4, 8 bytes
    - short int, int, long int, long long int
  - Operations
    - Arithmetic operations +, -, *, /
    - Comparison operations <, >, <=, >=, ==, !=
    - Bitwise operations ~, |, &, ^, <<, >>
  - Examples

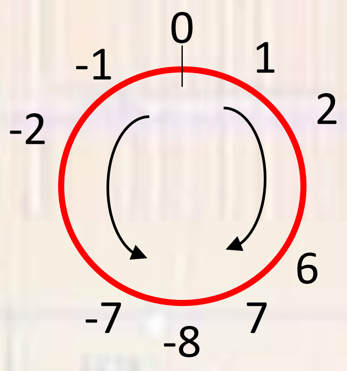
```
int aa, bb, cc;                    // declare 3 variables of type int
…
aa + bb;
// add aa to bb
```

C Review I

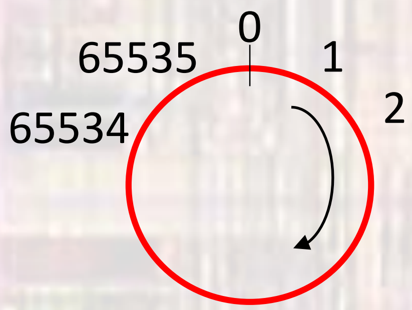
- C Types – int - integer
  - Special considerations with type int
    - Range is defined and limited
    - SIGNED and UNSIGNED variants



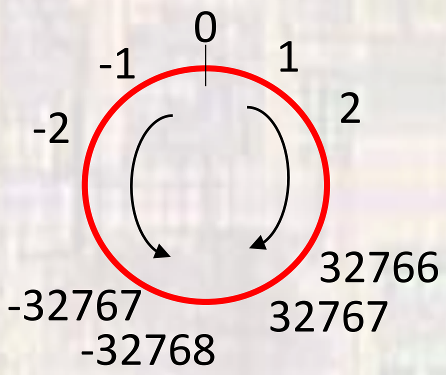
8 bit unsigned



8 bit signed



16 bit unsigned



16 bit signed

C Review I

- C Types – special integers
  - A special set of integers are defined for embedded systems
    - Designed to allow register/memory access
    - Not system dependent

#include <stdint.h>

signed char	**int8_t;**
unsigned char	**uint8_t;**
short	**int16_t;**
unsigned short	**uint16_t;**
int	**int32_t;**
unsigned	**uint32_t;**
long long	**int64_t;**
unsigned long long	**uint64_t;**

C Review I

- C Types – float – real
  - Values are system dependent
    - SIGNED
    - 4, 8 bytes – 1,8,23 - 1,11,52
    - float, double, long double
  - Operations
    - Arithmetic operations +, -, *, /
    - Comparison operations <, >, <=, >=, ==, !=
  - Examples

```
float aa, bb, cc;                    // declare 3 variables of type float
…
aa / bb;
// aa divided by bb
```

C Review I

- Variables

  - Symbolic representation for a value - name
  - Stored in memory (data)
  - Can be modified during execution

  - Since it requires space in memory it must have a type to tell the compiler how much space to reserve

C Review I

- Variable Declaration
  - Specify the type and name for a variable
  - Must be declared before it can be used

```
int foo;
float rate;
char initial1;

int var1, this, is, not, a, good, practice;

int AccountBalance;
int annual_interest_rate;
```

** Note: name length has no impact on compiled program size
focus on readable code

C Review I

- Variable Initialization

  - Variables are not initialized just by declaring them
    - They do not automatically have a value of 0
    - They may well have garbage values

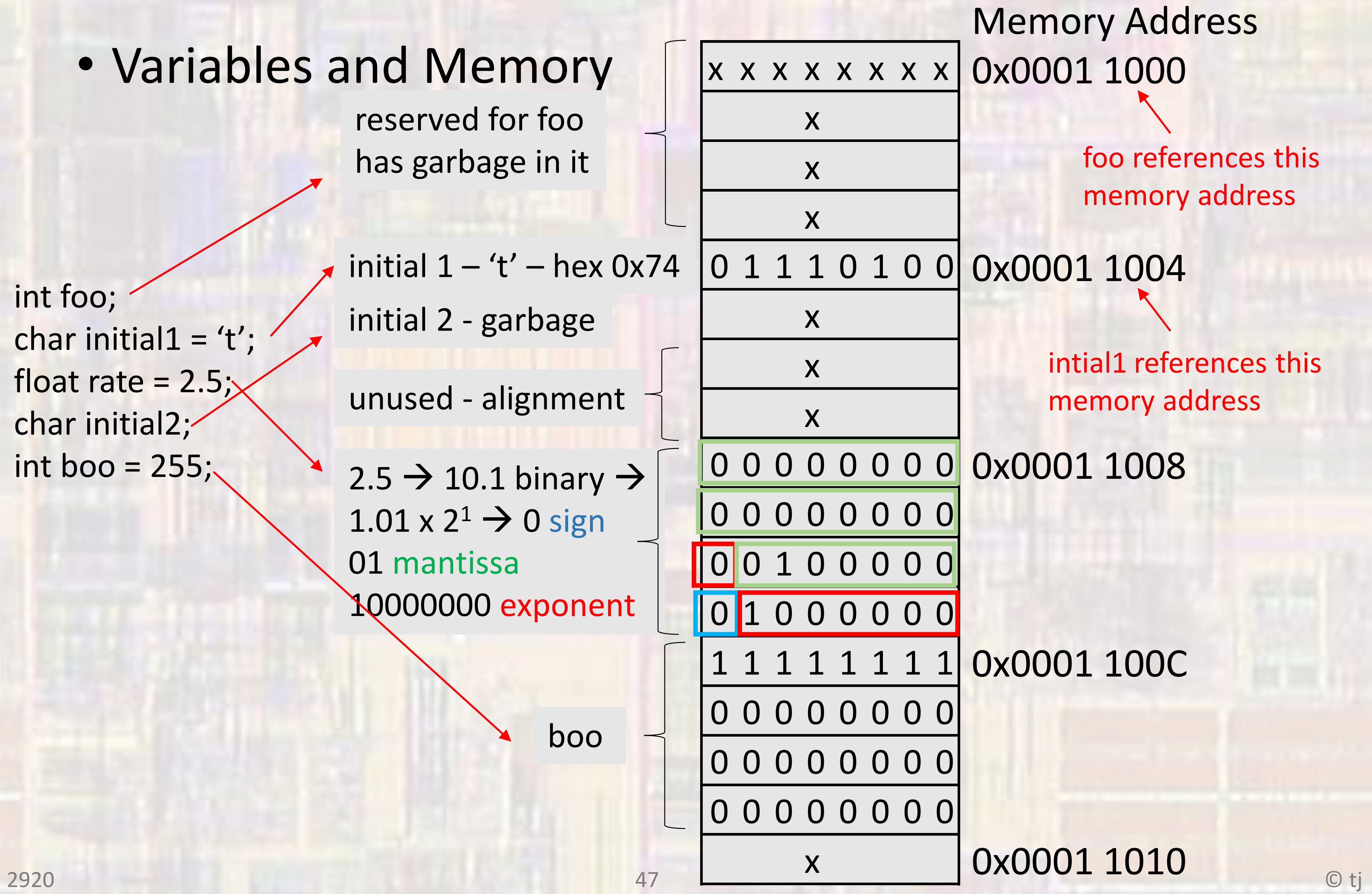
Nothing stops you from using an un-initialized variable

```
int foo = 23;                 int foo, boo = 23;

int count;                    int foo = 23, boo = 23;
count = 0;

char soo = 'A';               float pie = 3.14259;
```

C Review I

- Variables and Memory

```
int foo;
char initial1 = 't';
float rate = 2.5;
char initial2;
int boo = 255;
```



C Review I

- Constant
  - Symbolic representation for a value - name
  - Stored in memory (program)
  - Cannot be modified during execution
  - Since it requires space in memory it must have a type to tell the compiler how much space to reserve

```
const int foo = 12;
const float boo = 12.0;
const char goo = ‘T’;
```

C Review I

- Expressions
  - Sequence of Operators and Operands that reduce to a single value
    - Simple and Complex Expressions
    - Subject to Precedence and Associativity
    - Six categories

• Primary	One operand and no operators	foo, ‘a’
• Postfix	One operand followed by one operator	i++
• Prefix	One operator followed by one operand	--j
• Unary	One operator followed by one operand	-foo
• Binary	Operand operator operand	foo * boo
• Assignment	variable = expression	foo = boo * 2

C Review I

- Some expressions have a <u>Value</u> and a <u>Side Effect</u>

```
int j;
int x;
j = 5;

x = j++;

        Value:      x = 5
        Side Effect:  j = 6
```

C Review I

- Precedence
  - Order in which operators are evaluated
    - In math: * and / before + and –
    - 2/3+3*4 → ((2/3) + (3*4))
- Associativity
  - Order in which operators with the same precedence are evaluated
    - In math: left to right
    - 2 + 3 – 4 + 5 → (((2 + 3) – 4) + 5)

C Review I

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	| |	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

C Review I

- Expression Examples

a = 2, b=3, c=4

1 + 2 * 3 →

1 + 3 * 3 / 2 →

-b++ →

a += b *= c -= 3 →

--a * (1 + b) / 3 – c++ * b →

C Review I

- Bitwise Operators

  - ~ bitwise not
    - inverts the individual bits in a number
    - ~(1001 0110) → 0110 1001
  - | bitwise or
    - ORs the individual bits
    - (1001 1001) | (1010 1100) → 1011 1101
  - & bitwise and
    - ANDs the individual bits
    - (1001 1001) & (1010 1100) → 1000 1000
  - ^ bitwise xor
    - XORs the individual bits
    - (1001 1001) ^ (1010 1100) → 0011 0101

C Review I

- Bitwise Operators

  - >> bitwise shift right
    - shifts the individual bits in a number to the right
    - (1001 1001) >> 2 → 0010 0110   - unsigned

      OR

      → 1110 0110   - signed
  - << bitwise shift left
    - shifts the individual bits in a number to the left
    - (1001 1001) << 2 → 0110 0100   - unsigned or signed

C Review I

- Logical Operators

  - ! logical not
    - inverts the logical value
    - !true → false
    - !34 → false

  - || logical OR
    - evaluates both sides logically then does an OR
    - true || true → true
    - 0 || 0 → false
    - 34 || 32 → true

  - && logical AND
    - evaluates both sides logically then does an AND
    - true && true → true
    - 1 && 0 → false
    - 34 && 32 → true

C Review I

- Relational Operators

  - ==, <, >, <=, >=, !=
    - equals, LT, GT, LE, GE, not equal
    - evaluates the parameters by type

    - true == true → true
    - 34 == 32 → false
    - 34 != 32 → true
    - 5 >= 5 → true

C Review I

- Implicit Type Conversion
  - Type conversions done automatically by the compiler
  - Each type has a RANK

    bool < char

    < short < int < long < long long

    < float < double < long double

    complex types match the floating types

C Review I

- Implicit Type Conversion

times

int * float → float

1) int converted to float
2) multiplication
3) result is of type float

char + long int → long int

1) char converted to long int
2) addition
3) result is of type long int

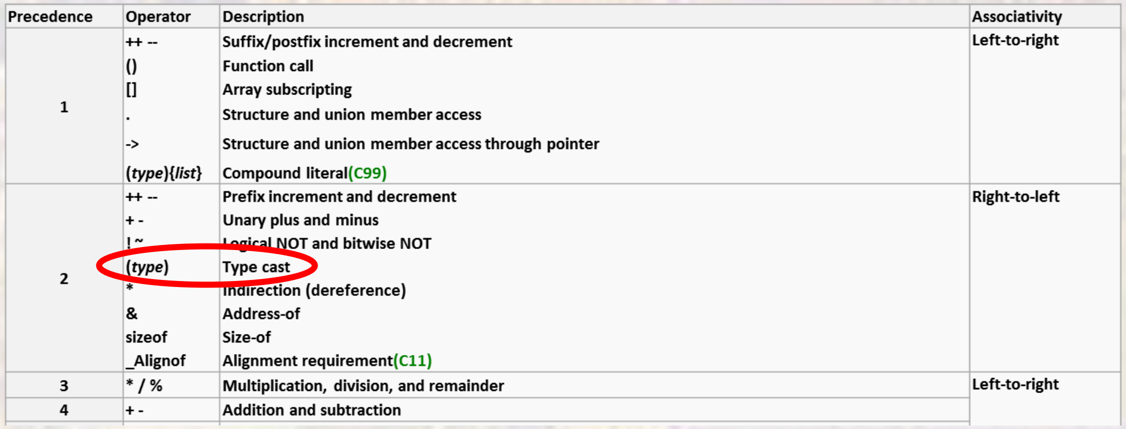
C Review I

- Explicit Type Conversion

  - Cast or casting
  - Force a type conversion

  - Use the unary operator “type cast”

(desired_type) var



Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	

C Review I

- Explicit Type Conversion

```
int a = 5;
int b = 2;
```

a/b	2
(float) a / b	5.0 / 2 → 5.0 / 2.0 → 2.5
a / (float) b	5 / 2.0 → 5.0 / 2.0 → 2.5
(float) (a/b)	(float) (5/2) → (float) 2 → 2.0

C Review I

- Assignment Type Conversion

  - Regardless of any implicit or explicit type conversions the assignment operator side effect cannot change the type of a variable

```
int a = 5;
float b = 12.0;
int c = 5;
a = b / c;
    c is promoted to type float
    right side is evaluated 12.0 / 5.0 → 2.4
    value is demoted to match the receiving variable (side effect)
    a = 2
```