

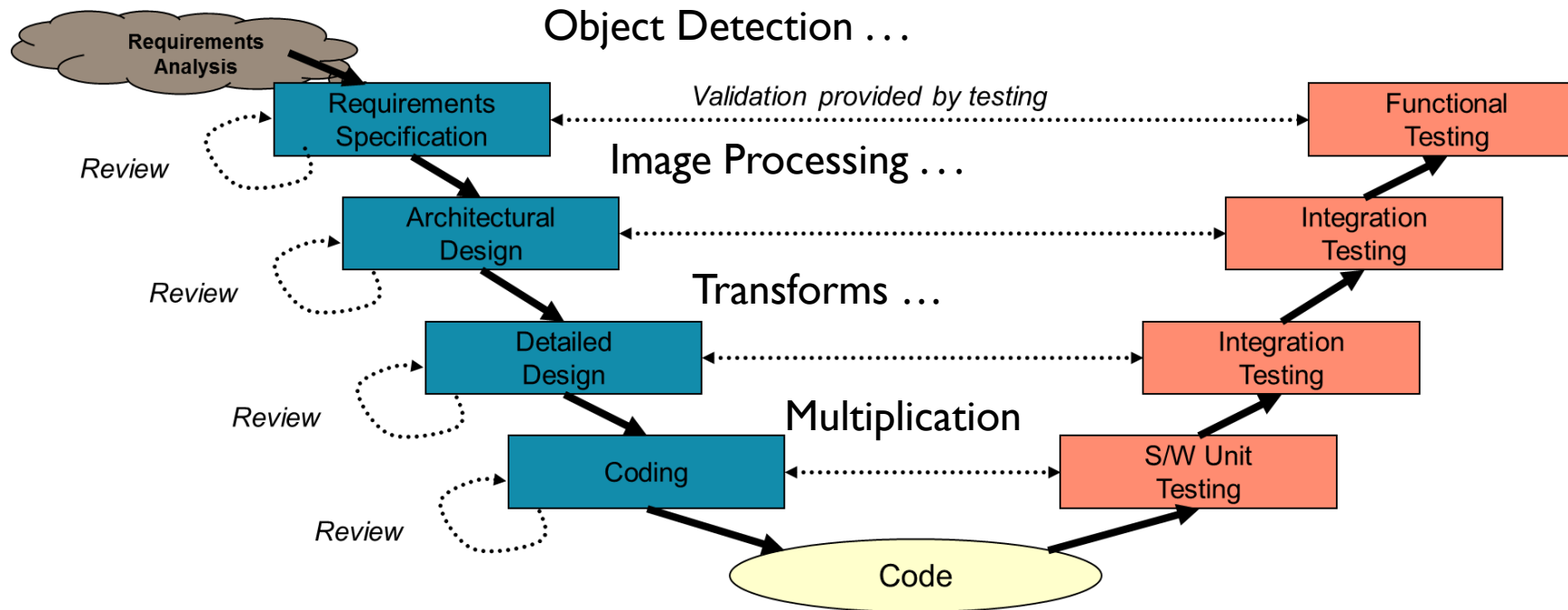
CHW 469 : Embedded Systems

Instructor:

Dr. Ahmed Shalaby <http://bu.edu.eg/staff/ahmedshalaby14#>

<https://piazza.com/fci.bu.edu.eg/spring2017/chw469/home>

V Model Overview



Assignment no. 3

kindly read the following paper [[Software Engineering for Space Exploration](#)]. In short, one paper only (2 pages), write an essay mention your opinion about the topic.

Notes:

- you will deliver your report on Sunday (at lecture time).
- you can work in a group but the group is only two students.
- you may need to read more - paper references or external resources.
- at lecture time, there will be a discussion regarding the topic, be ready to present the topic and discuss it.

Introduction to C Programming

- **Compiler** is system software **converts a high-level** language program (human readable format) into **object code** (machine readable format).
gcc, visual studio.
- **Assembler** is system software **converts an assembly language** program (human readable format) into **object code** (machine readable format).
- **Linker** builds software system by connecting (linking) **software components**.
- **Loader** places the **object code in memory**. In an embedded system, the loader programs object code into flash ROM.
- **Debugger** is a set of **hardware and software tools** we use to **verify system is operating correctly**. The two important aspects of a good debugger are **control** and **observability**.

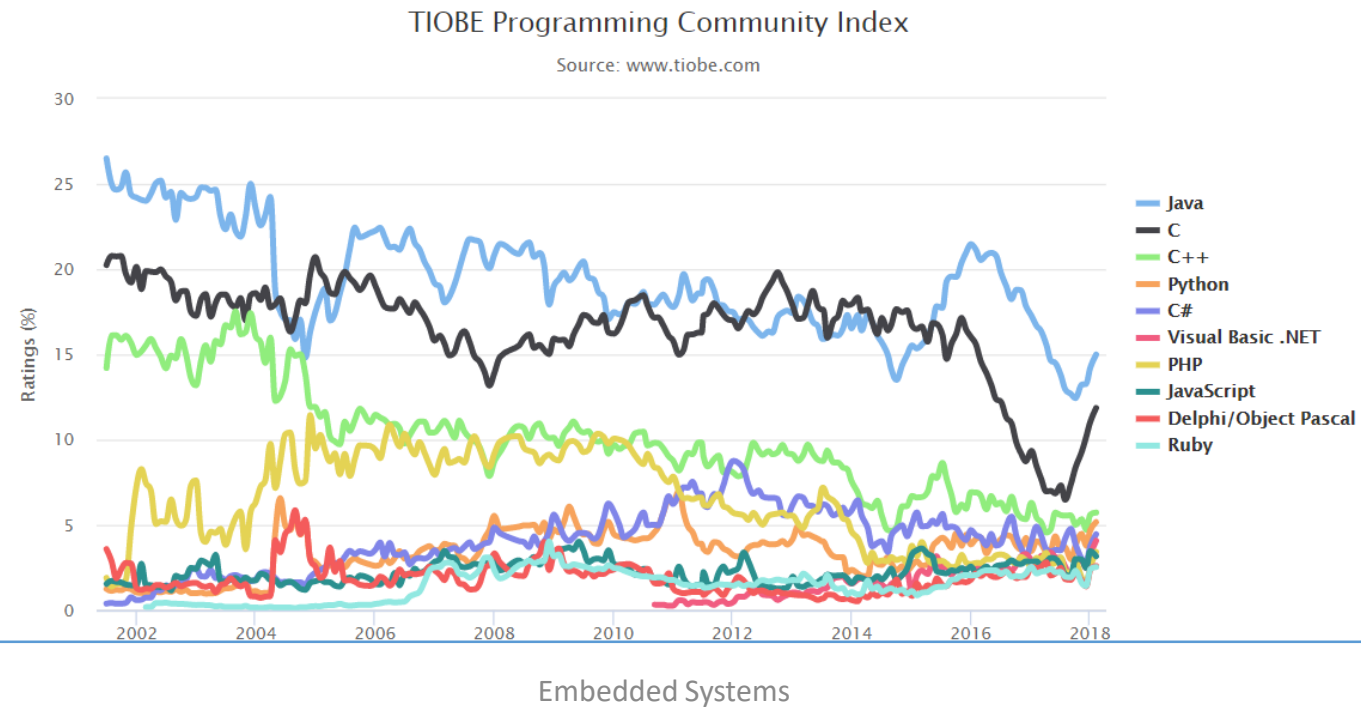
C code (**z = x+y;**) →
Assembly code (**ADD R2,R1,R0**) →
Machine code (**0xEB010200**)

Introduction to C Programming – What?

- C is a **general-purpose** programming language initially developed by **Dennis Ritchie** between 1969 and 1973 while at AT&T Bell Labs.
- In most programming languages the column position and line number affect the meaning. On the contrary, **C is a free field language**. Except for preprocessor lines (that begin with #), **spaces, tabs and line breaks** have the same meaning.
- C was invented to write an operating system - **UNIX**.
- C is a successor of **B language**.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

Introduction to C Programming – Why ?

- Why C language ?
 - last ten years, **ranked one or two** - high-level languages. (**popular**)
 - C is the **most common** language for embedded systems. It is **not tied** to any particular hardware or system.
 - C is **efficient** programming language.
 - C is **high/mid** level language.



Introduction to C Programming – Why ?

- C is much more flexible than **other high-level programming languages**:
 - C is a **structured** language.
 - C is a **relatively small** language.
 - C has very **loose data typing**.
 - C easily supports **low-level bit-wise** data manipulation.
 - C is sometimes referred to as a “**high-level assembly language**”.
- When **compared to assembly language** programming:
 - Code written in C can be more **reliable**.
 - Code written in C can be more **scalable**.
 - Code written in C can be more **portable** between different platforms.
 - Code written in C can be **easier to maintain**.
 - Code written in C can be **more productive**.

Embedded C Programming !

Main characteristics of an Embedded programming environment:

- Limited **ROM**.
- Limited **RAM**.
- Limited **stack space**.
- **Hardware oriented** programming.
- **Critical timing** (Interrupt Service Routines, tasks, ...).
- Many **different pointer** kinds (far / near / rom / uni / paged / ...).
- **Special keywords** and tokens (@, interrupt, tiny, ...).

Introduction to C Programming – How ?

- Variables and Data Types.
- Operators and Hardware Manipulation.
- Program Flow Control.
- Advanced Types, Constants and Expressions.
- Arrays and Pointer Basics.
- Functions.
- Structures and Unions.
- Scheduling Techniques.
- Arrays of Pointers.
- Declarations.
- Preprocessor.
- Real-Time Operating Systems.

Introduction to C Programming– Program

- C Program is divided into **four sections**.
- Every C program has a main, and execution begins at the top of this main.

```
//**** 0. Documentation Section
// This program calculates the area of square shaped rooms
// Author: Ramesh Yerraballi & Jon Valvano
// Date: 6/28/2013
//
// 1. Pre-processor Directives Section
#include <stdio.h> // Diamond braces for sys lib: Standard I/O
#include "uart.h" // Quotes for user lib: UART lib
#define SIZE 10 // SIZE is found as a token, it is replaced with the 10
// 2. Global Declarations section
// 3. Subroutines Section
// MAIN: Mandatory routine for a C program to be executable
int main(void) {
    UART_Init(); // call subroutine to initialize the uart
    printf("This program calculates areas of square-shaped rooms\n");
}
```

Introduction to C Programming – Keywords

Standard ANSI C recognizes the following keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

__asm Specify a function is written in assembly code

C Programming / Basic Syntax

- A C program consists of **various tokens** and a token is either a keyword, an identifier, a constant, a string literal, or a symbol.
- Comments are like helping text in your C program and they are ignored by the compiler. `/* This is a comment */`
- A C identifier is a name used to **identify a variable, function, or any other user-defined item**.
- A C identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).
- C does not allow punctuation characters such as @, \$, and % within identifiers.
- C is a **case-sensitive programming** language.

C Programming / Punctuation

- Punctuation marks (semicolons, colons, commas, apostrophes, quotation marks, braces, brackets, and parentheses)

<i>Punctuation</i>	<i>Meaning</i>
;	End of statement
:	Defines a label
,	Separates elements of a list
()	Start and end of a parameter list
{ }	Start and stop of a compound statement
[]	Start and stop of a array index
" "	Start and stop of a string
' '	Start and stop of a character constant

C Programming / Storage Classes

- A storage class **defines the scope (visibility)** and life-time of variables and/or functions within a C.

Storage Class	Description
auto	auto storage class is the default storage class for all local variables.
register	register storage class is used to define local variables that should be stored in a register instead of RAM
static	static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.
extern	extern storage class is used to give a reference of a global variable that is visible to ALL the program files.

C Programming / Variables and Expressions

- Local variables contain **temporary information** that is accessible only within a narrow scope.
- In C, **local variable** must be **declared immediately after a brace {** that begins a compound statement. Unlike **globals**, which are said to be **static**, **locals are created dynamically** when their block is entered, and they cease to exist when control leaves the block.
- Although two **global** variables **cannot use the same name**, a local variable of one block can use the same name as a local variable in another block.
- **Constants** refer to **fixed values** that the program may not alter during its execution.
- Constants can be of any of the **basic data types** like an integer constant.
- **void** represents the **absence of type**.

C Programming / Variables and Expressions

Data type	Precision	Range
unsigned char	8-bit unsigned	0 to +255
signed char	8-bit signed	-128 to +127
unsigned int	compiler-dependent – 16 or 32 bits	
int	compiler-dependent– 16 or 32 bits	
unsigned short	16-bit unsigned	0 to +65535
short	16-bit signed	-32768 to +32767
unsigned long	unsigned 32-bit	0 to 4294967295L
long	signed 32-bit	-2147483648L to 2147483647L

C Programming / Variables and Expressions

```
int main(void) {  
    unsigned long side; // room wall meters  
    unsigned long area; // size squared meters  
    UART_Init();      // call subroutine to initialize the uart  
    side = 3;  
    area = side*side;  
    printf("\nArea of the room with side of %ld m is %ld sqr m\n",side,area);  
}
```

C Programming / 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

C Programming / Arithmetic Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
+	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.	$A-- = 9$

Given that **A** holds 10 and **B** holds 20

C Programming / Relational Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
<code>==</code>	Checks if the values of two operands are equal . If yes, then the condition becomes true.	(A == B) is not true.
<code>!=</code>	Checks if the values of two operands are not equal . If the values are not equal, then the condition becomes true.	(A != B) is true.
<code>></code>	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.
<code><</code>	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.
<code>>=</code>	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.
<code><=</code>	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.

Given that **A** holds 10 and **B** holds 20

C Programming / Logical Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
&&	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.

Given that **A** holds 1 and **B** holds 0

C Programming / Bitwise Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
&	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) = -61, i.e., 1100 0011
<<	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

Given that **A** = 0011_1100 (60) and **B** = 0000_1101(13)

C Programming / Assignment Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
=	Simple assignment operator . Assigns values from right side operands to left side operand	$C = A + B$ assigns 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 $C = C \% A$

C Programming / Assignment Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
<code><<=</code>	Left shift AND assignment operator.	<code>C <<= 2</code> is same as <code>C = C << 2</code>
<code>>>=</code>	Right shift AND assignment operator.	<code>C >>= 2</code> is same as <code>C = C >> 2</code>
<code>&=</code>	Bitwise AND assignment operator.	<code>C &= 2</code> is same as <code>C = C & 2</code>
<code>^=</code>	Bitwise exclusive OR and assignment operator.	<code>C ^= 2</code> is same as <code>C = C ^ 2</code>
<code> =</code>	Bitwise inclusive OR and assignment operator.	<code>C = 2</code> is same as <code>C = C 2</code>

C Programming / Misc Operators

<i>Operation</i>	<i>Description</i>	<i>Example</i>
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

C Programming / Precedence - Priority

<i>Precedence</i>	<i>Operators</i>	<i>Associativity</i>
highest	() [] . -> ++(postfix) --(postfix)	left to right
	++(prefix) --(prefix) !~ sizeof (type) +(unary) - (unary) &(address) *(dereference)	right to left
	* / %	left to right
	+ -	left to right
	<< >>	left to right
	< <= > >=	left to right
	== !=	left to right
	&	left to right
	^	left to right
		left to right
	&&	left to right
		left to right
	? :	right to left
	= += -= *= /= %= <<= >>= = &= ^=	right to left
lowest	,	left to right

C Programming / Operators

```
void main(void)
{
    long x,y,z;
    x=1; y=2;
    z = x+4*y;
    x++;
    y--;
    x = y<<2;
    z = y>>2;
    y += 2;
}
```

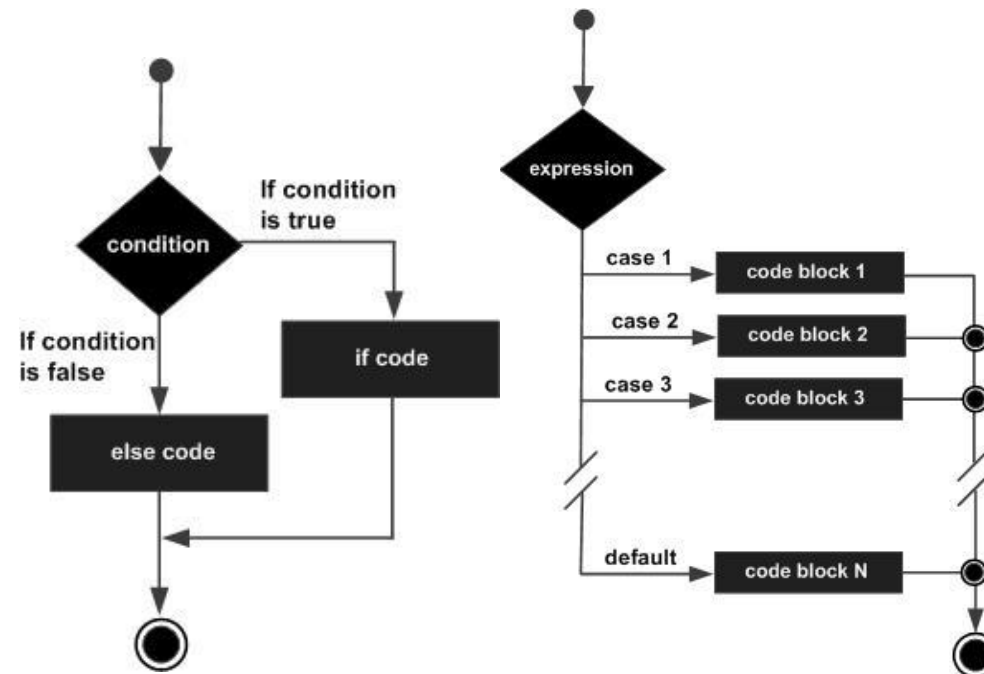
What is the value of x, y and z ?

C Programming / Operators

```
void main(void)
{
    long x,y,z; // Three local variables
    x=1; y=2; // set the values of x and y
    z = x+4*y; // arithmetic operation z = 1+8 = 9
    x++; // same as x=x+1; x = 2
    y--; // same as y=y-1; y = 1
    x = y<<2; // left shift same as x=4*y; x = 4
    z = y>>2; // right shift same as x=y/4; y = 1
    y += 2; // same as y=y+2; y = 3
}
```

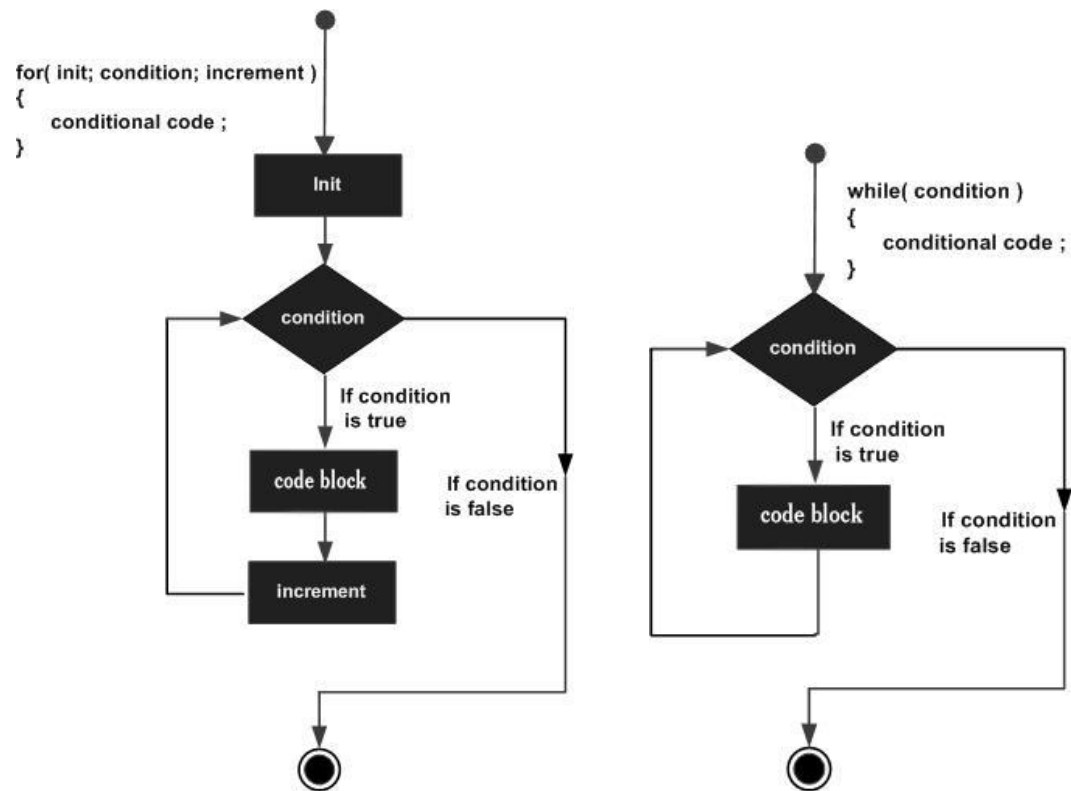
C Programming / Conditional Branching

- Decision making structures specifies one or more conditions to be evaluated or tested, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Conditional statements :
 - if statement
 - if...else statement
 - nested if statements
 - switch statement
 - nested switch statements
 - ? Operator



C Programming / Loops

- A loop statement allows execution of a statement or group of statements **multiple times**.
- Loop statements :
 - while loop.
 - for loop.
 - do ... while loop.
 - nested loop.



C Programming / Loop control

- Loop control statements **change execution from its normal sequence.**
When execution leaves a scope,
- Control statements :
 - **break** statement: **terminates the loop** or switch statement and transfers execution to the statement immediately following the loop or switch.
 - **continue** statement: causes the loop to skip the remainder of its body and **immediately retest its condition** prior to reiterating.
 - **goto** statement: **transfers control** to the labeled statement.

C Programming / Functions

- A function is a **sequence of operations** that can be invoked from other places within the software.
- A function can have zero or one output parameter.
- C programmer to distinguish the two terms **declaration** and **definition**.
- A **function declaration** (prototype) specifies its **name**, its **input parameters** and its **output parameter**.
- A **function definition** specifies the **exact sequence of operations** to execute when it is called.
- A function definition **generates object code**, which are machine instructions to be loaded into memory.
- An **object must be declared or defined** before it can be used in a statement.
- Actually the **preprocessor performs** the first pass through the program that handles the preprocessor directives.
- A **top-down approach** is to first declare a function, use the function, and lastly define the function.

C Programming / Functions

- Every C program **has at least one function**, which is **main()**, and all the most trivial programs can define additional functions.
- The C standard library provides numerous **built-in functions** your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.
- The main component of the function are:
 - Name:
The actual name of the function. **The function name and the parameter list** together constitute the **function signature**.
 - Parameters:
A parameter is like a **placeholder**. When a function is invoked, you **pass a value to the parameter**.
 - Body:
a **collection of statements** that define what the function does.
 - Return Type:
Function **may return a value**. The return type is the data type of this **value**.

C Programming / Functions

```
unsigned long Calc_Area(unsigned long s);
int main(void) {
    unsigned long side; // room wall meters
    unsigned long area; // size squared meters
    UART_Init(); // call subroutine to initialize the uart
    printf("This program calculates areas of square-shaped rooms\n");
    side = 3;
    area = Calc_Area(side);
    printf("\nArea of the room with side of %ld m is %ld sqr m\n",side,area);
    side = side+2;
    area = Calc_Area(side);
    printf("\nArea of the room with side of %ld m is %ld sqr m\n",side,area);
}
// Calculates area
// Input: side of a room (unsigned long) in meters
// Output: area of the room (unsigned long) in square meters
unsigned long Calc_Area(unsigned long s) {
    unsigned long result;
    result = s*s;
    return(result); }
```

Introduction to C Programming – Keywords

Standard ANSI C recognizes the following keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

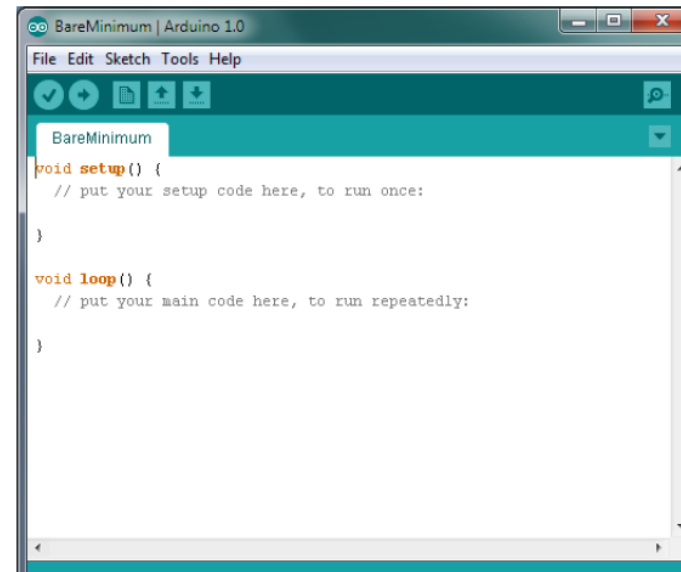
- The volatile keyword **disables compiler optimization**, forcing the compiler to **fetch a new value each time**.
- volatile is used when **defining I/O ports** because the value of ports can change outside of software action. volatile is used when sharing a **global variable** between the main program and an **interrupt service routine**.

The “*Super Loop*” Software Architecture

- The main difference between the **embedded systems** and **desktop computer systems** in programming is:
 - The vast majority of embedded systems are required to **run only one program**.
 - This program **start running** when the microcontroller is **powered up**, and will **stop running** when the **power is off**.

```
void main(void)
{
  X_Init(); // initialize function X

  while(1) // 'for ever' (Super Loop)
  {
    X(); // Run function X()
  }
}
```

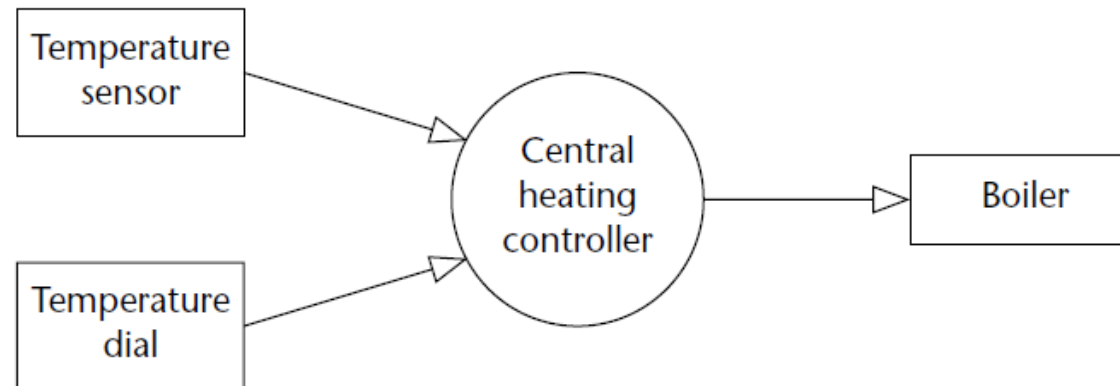


Ex.1 – Flashing LED

```
void main(void)
{
    LED_FLASH_Init();
    while(1)
    {
        // Change the LED state (OFF to ON, or vice versa)
        LED_FLASH_Change_State();
        // Delay for *approx* 1000 ms
        DELAY_LOOP_Wait(1000);
    }
}
```

Ex.2 - Central Heating Controller

- Develop a microcontroller-based control system to be used as part of the central-heating system in a building. The system consists of:
 - The gas-fired boiler (which we wish to control).
 - The sensor (measuring room temperature).
 - The temperature dial (through which the desired temperature is specified) – user interface.
 - The controller itself.



Ex.2 - Central Heating Controller (Cont...)

```
/* Framework for a central heating system using a Super Loop. [Compiles and
runs but does nothing useful */
] #include "Cen_Heat.h"
/*-----*/
void main(void)
{
    C_HEAT_Init(); // Init the system
    while(1) // 'for ever' (Super Loop)
    {
        // Find out what temperature the user requires (via the user interface)
        C_HEAT_Get_Required_Temperature();
        // Find out what the current room temperature is (via temperature sensor)
        C_HEAT_Get_Actual_Temperature();
        // Adjust the gas burner, as required
        C_HEAT_Control_Boiler();
    }
} /*--- END OF FILE -----*/
```

MDK Introduction

- The **Keil Microcontroller Development Kit (MDK)** helps you to create embedded applications for **ARM Cortex-M** processor-based devices. MDK is a powerful, yet easy to learn and use development system.
- The MDK-Core is **based on the genuine Keil μ Vision IDE/Debugger** with leading support for Cortex-M processor-based microcontroller devices.
- [Getting Started with MDK Create Applications with \$\mu\$ Vision® for ARM® Cortex®-M Microcontrollers](#)

