Variable Name

Variable name can be any set of letters and numbers of a length up to 256 characters. Following constrains must be respected:

- Do not use any reserved keyword in C like (void, include, int)
- Do not use space or any special character inside variable name except "_".
- Do not start with a number

Correct variable names	M n Values	m_name counter name1	name2 min_value
Wrong variables names	Min value max>name void	5names printf	min-value

Comments

- Sometimes programmers need to add some notes beside their code. Those notes are very important to describe the code and to clarify complex operation. Notes or comments can be added in two ways as shown in the following example.

```
        double temprature;

        //Supply the temprature in Fahrenheit
        printf("Enter the temprature in Fahrenheit : \r\n");
        scanf("%lf", &temprature);

        /*Convert temprature from
        Fahrenheit to Celsius */
        temprature = (temprature - 32.0) * 5.0/9.0;

        //prints the
        //result
        printf("The temprature in Celsius is %lf\r\n",
                                        temprature);
}
```

Data Types

- **User Defined**
  - enum
  - typedef
- **Derived**
  - Arrays
  - structure
  - union
  - pointer
- **Primitive/Basic Types**
  - **Real Values**
  - **Integer Values**
    - **unsigned**
    - **signed**

Integer Values

Data Type	Major Type	Size (Bytes)	Precision	Range
Char	Integer	1	1	–128 to 127
unsigned char	Integer	1	1	0 to 255
Short	Integer	2	1	–32,768 to 32,767
unsigned short	Integer	2	1	0 to 65,535
*int	Integer	4	1	–2,147,483,648 to 2,147,483,647
*unsigned int	Integer	4	1	0 to 4,294,967,295
Long	Integer	4	1	–2,147,483,648 to 2,147,483,647
unsigned long	Integer	4	1	0 to 4,294,967,295
long long		8		–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long		8		0 to 18,446,744,073,709,551,615

Unsigned Integer

$$0 >>> (2^{size\_in\_bits} - 1)$$

Signed Integer

$$-(2^{(size\_in\_bits-1)}) >>> +(2^{(size\_in\_bits-1)} - 1)$$

For example “char”

For example if the **Unsigned char** value uses one byte (8 bits) to hold the numeric value:

- The minimum value is $(00000000)_2 = (0)_{10}$.
- The maximum value is $(11111111)_2 = (2^8\text{-}1)_{10} = (255)_{10}$.

Another example if the **Signed char** value uses one byte (8 bits) to hold the numeric
value. If Tow‟s Complement method is used to represent the negative values:

- The minimum value is $(10000000)_2 = -(10000000)_2 = -(2^7)_{10} = -(128)_{10}$.
- The maximum value is $(01111111)_2 = (2^7\text{-}1)_{10} = (127)_{10}$.

For example “char”

For example if the **Unsigned short** value uses one byte (16 bits) to hold the numeric value:

- The minimum value is $(0000000000000000)_2 = (0)_{10}$.
- The maximum value is $(1111111111111111)_2 = (65535)_{10}$.

Another example if the **Signed Integer** value uses four byte (32 bits) to hold the numeric
value. If Tow"s Complement method is used to represent the negative values:

- The minimum value is $(80000000)_{16} = - (7FFFFFFF)_{16} = - (2^{31})_{10} = - (2147483648)_{10}$.
- The maximum value is $(FFFFFFFF)_{16} = (2^{31}-1)_{10} = (2147483647)_{10}$.

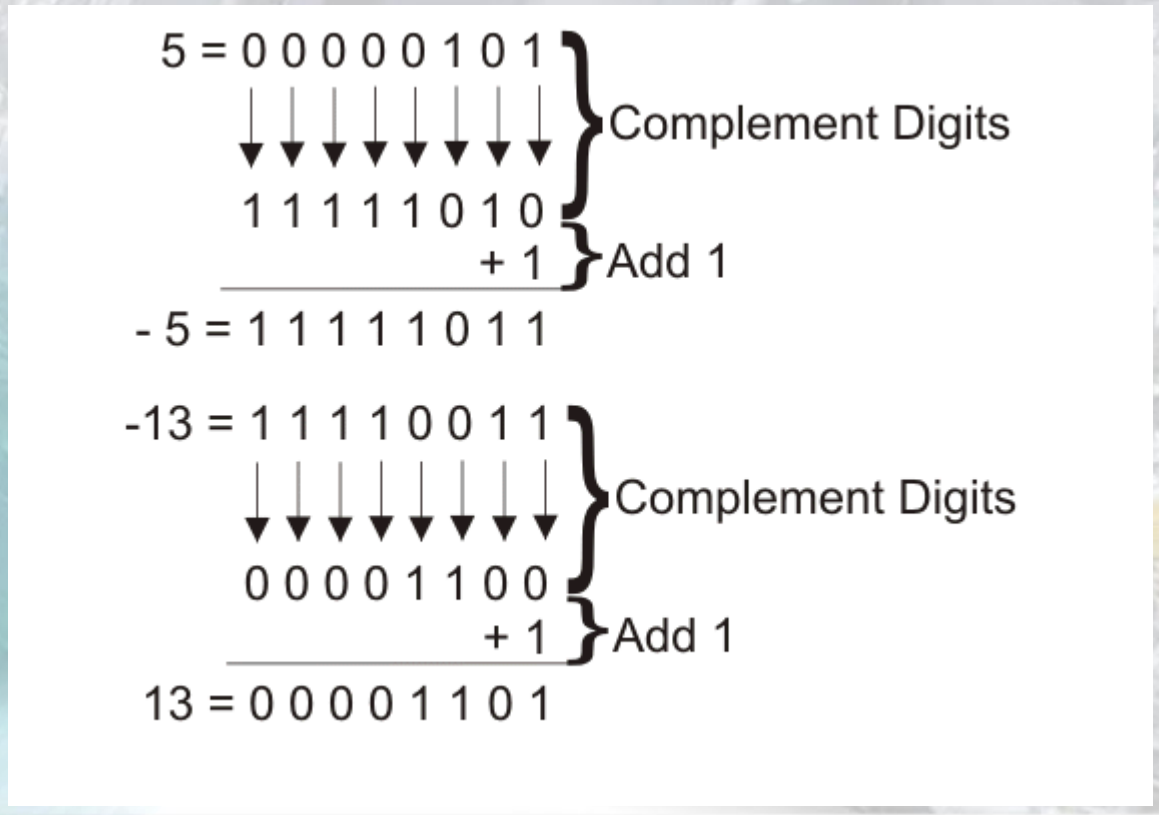
Note

- **int**, this data type called the machine dependent data type, which means its size and range
  vary from a machine type to another machine type (EX: in 8 bit computers **int** is 1 byte, in
  16 bit computers **int** is 2 bytes, in 32 bit computers **int** is 4 bytes, in 64 bit computers **int** is 8
  bytes).
  Know that (32 Bits computers) means the principle data unit size in those computers are 32
  bit, which mean the computer is designed and optimized to process 32 bit values

Floating-Point Types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

2's Complement



bool is a data type ?

```c
#include <stdio.h>
#include <stdbool.h>
 int main()
{
bool a = true;
return 0;
}
```


bool

- Traditionally, there was no boolean type in C. However, C99 defines a standard boolean type under <sdbool.h> header file. A boolean type can take one of two values, either true or false. For example:

```c
#include <stdio.h>
#include <stdbool.h>
 int main()
{
bool a = true;
return 0;
}
```

quiz: ☺ what is the output

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 15, 2018
 *      Author: keroles
 */

#include "stdio.h"
int main ()
{
    int x = 5 ;
    float y = 2.0 ;
if (x/y == 2)
    printf (" int/float >>> int \n"  ) ;
else if (x/y == 2.5)
    printf (" int/float >>> float \n"  ) ;

return 0 ;
}
```

quiz: ☺ what is the output

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 15, 2018
 *      Author: keroles
 */

#include "stdio.h"
int main ()
{
    int x = 5 ;
    float y = 2.0 ;
if (x/y == 2)
    printf (" int/float >>> int \n"  ) ;
else if (x/y == 2.5)
    printf (" int/float >>> float \n"  ) ;

return 0 ;
}
```

Problems | Tasks | Console | Properties | AVR Device Explorer

<terminated> (exit value: 0) EmbedXpro_C_1.exe [C/C++ Application] C:\Users\kkhalil\av

```
int/float >>> float
```

Integer and Float Conversions

- In order to effectively develop C programs, it will be necessary to understand the rules that are used for the implicit conversion of floating point and integer values in C. These are mentioned below. Note them carefully.
  - An arithmetic operation between an integer and integer always yields an integer result.
  - An operation between a real and real always yields a real result.
  - An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

- Here in the first assignment statement though the expression's value is a float (3.5) it cannot be stored in i since it is an int.
- In such a case the float is demoted to an int and then its value is stored. Hence what gets stored in i is 3.
- Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and then stored in b, since b being a float variable cannot hold anything except a float value.

```
int i ;
float b ;
i = 3.5 ;
b = 30 ;
```


Type Conversion in Assignments

- in the assignment statement some operands are **int**s whereas others are **float**s. As we know, during evaluation of the expression
- the **int**s would be promoted to **float**s and the result of the expression would be a **float**. But when this **float** value is assigned to **s** it is again demoted to an **int** and then stored in **s**.

```
float a, b, c ;
int s ;
s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;
```


Quiz ☺ What is the output

k is an integer variable and a is a real variable.

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9		a = 2 / 9	
k = 2.0 / 9		a = 2.0 / 9	
k = 2 / 9.0		a = 2 / 9.0	
k = 2.0 / 9.0		a = 2.0 / 9.0	
k = 9 / 2		a = 9 / 2	
k = 9.0 / 2		a = 9.0 / 2	
k = 9 / 2.0		a = 9 / 2.0	
k = 9.0 / 2.0		a = 9.0 / 2.0	

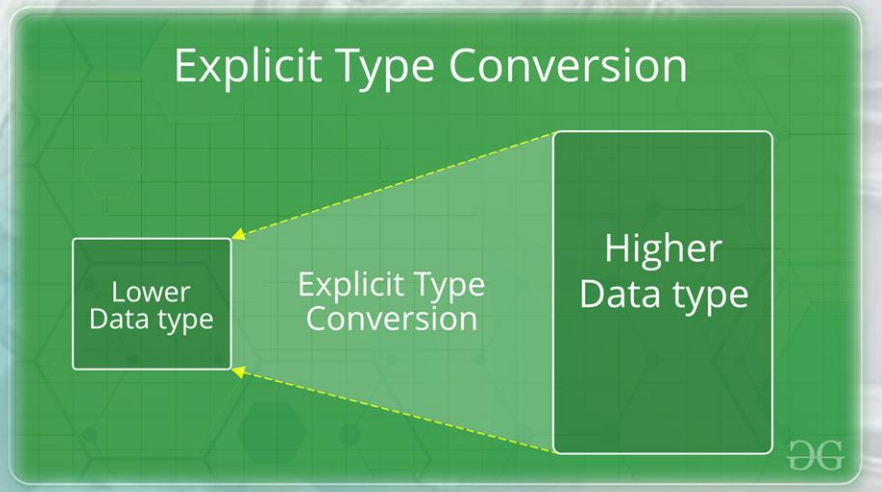
Quiz ☺ What is the output

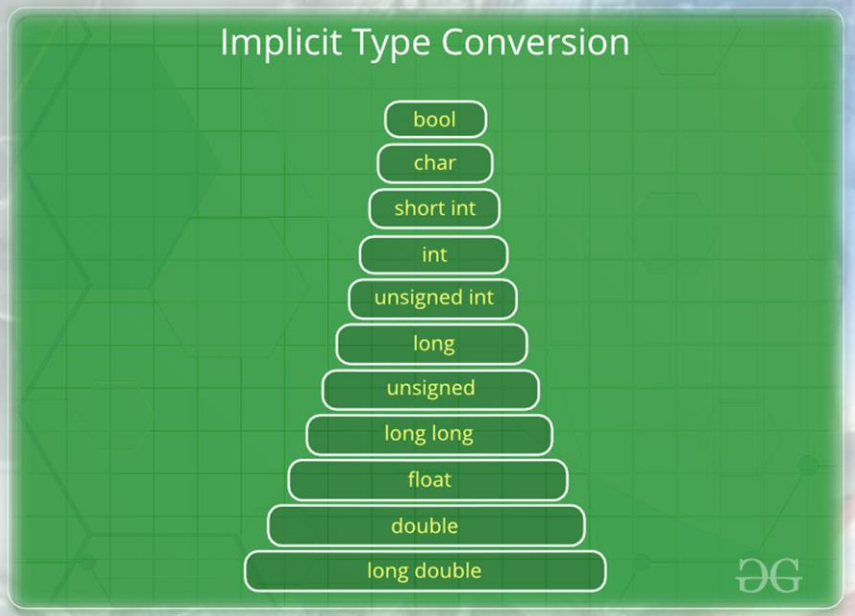
k is an integer variable and a is a real variable.

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

In the first statement, since both 2 and 9 are integers, the result is an integer, i.e. 0. This 0 is then assigned to **k**. In the second statement 9 is promoted to 9.0 and then the division is performed. Division yields 0.222222. However, this cannot be stored in **k**, **k** being an **int**. Hence it gets demoted to 0 and then stored in **k**.

Type Conversion in C

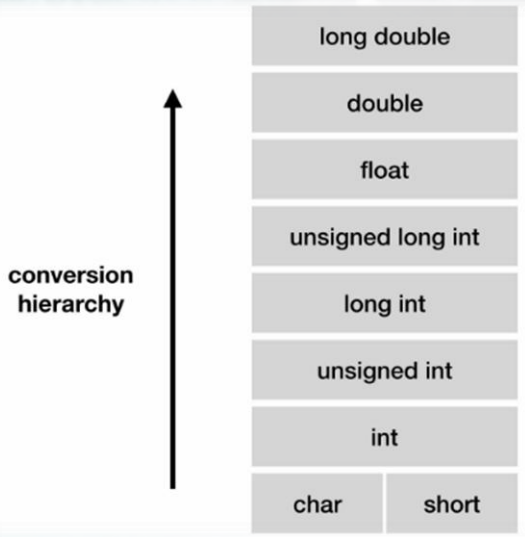




Implicit Type Conversion

Also known as 'automatic type conversion'.

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.
- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).




```c
// An example of implicit conversion
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

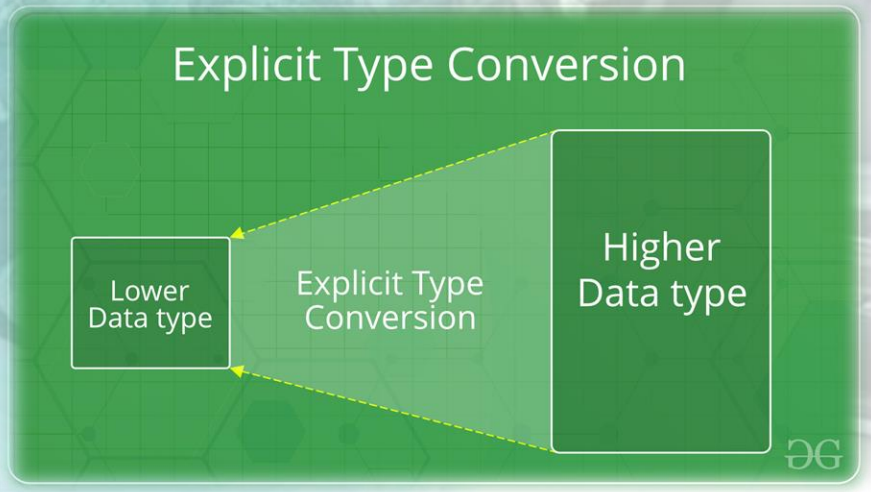
```
x = 107, z = 108.000000
```


Explicit Type Conversion

This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

(type) expression




```c
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

```
sum = 2
```


Hierarchy of Operations

While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed. For example, does the expression 2 * x - 3 * y correspond to (2x)-(3y) or to 2(x-3y)? Similarly, does A / B * C correspond to A / (B * C) or to (A / B) * C? To answer these questions satisfactorily one has to understand the 'hierarchy' of operations.

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Example 1.1: Determine the hierarchy of operations and evaluate the following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8 operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8 operation: /
i = 1 + 1+ 8 - 2 + 5 / 8 operation: /
i = 1 + 1 + 8 - 2 + 0 operation: /
i = 2 + 8 - 2 + 0 operation: +
i = 10 - 2 + 0 operation: +
i = 8 + 0 operation : -
i = 8 operation: +

C Programming Input Output (I/O): printf() and scanf()

- C programming has several in-built library functions to perform input and output tasks.
- Two commonly used functions for I/O (Input/Output) are printf() and scanf().
- The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

Input Output

main.c

```c
//Prepared by keroles
#include <stdio.h>
int   main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d",&testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first_c_code.exe (3/17/

```
12
Enter an integer: Number = 12
```

Eclipse's terminal emulator issue with scanf()

- Eclipse's terminal emulator might be different and do more buffering. Try calling fflush(stdout); between the printout and the call to scanf().

main.c

```
1  //Prepared by keroles
2  #include <stdio.h>
3  int   main()
4  {
5      int testInteger;
6      printf("Enter an integer: ");
7      fflush(stdout);
8      scanf("%d",&testInteger);
9      printf("Number = %d",testInteger);
10     return 0;
11 }
12
```

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs

<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first_c_code.exe (3/17/17, 11:51 AM)

```
Enter an integer: 12
Number = 12
```

C Floats Input/Output

main.c

```c
//Prepared by keroles
#include <stdio.h>
int main()
{
    float f;
    printf("Enter a number: ");
    fflush(stdout);
    // %f format string is used in case of floats
    scanf("%f",&f);
    printf("Value = %f", f);
    return 0;
}

```

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs

<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first_c_code.exe (3/17/17, 11:55 AM)

```
Enter a number: 12.548
Value = 12.548000
```

int printf(const char* format, ...);

'\r\n'	Makes a newline
'\t'	Inserts a tab
'\\'	Prints '\'
'\"'	Prints '"'
'%d'	Prints or scans an integer (int) value
'%x' or '%X'	Prints or scans an integer value in small or capital hexadecimal format
'%f'	Prints or scans a real (float) value
'%lf'	Prints or scans a real (double) value
'%u'	Prints or scans an (unsigned int) value
'%c'	Prints or scans a single character (char)

Quiz

Code	Output	Description
`printf("A\r\nB\r\nC");`	Expect The Output	`"\r\n"` makes a line break where '`\r`' is the carriage return and '`\n`' is the newline command.

`printf("A\tB\tC\r\n");`
`printf("D\tE\tF\r\n");`
`printf("N\tO\tP\r\n");`
		`'\t'` makes a tab separator.

`printf("A\\B\\C ");`		To print the '`\`' letter you must place '`\\`' instead.
`printf("Say \"Hello\"");`		To print the '`"`' letter you must place '`\"`' instead.
`int a = 20*30;` `printf("Area is %d",a);`		The directive `'%d'` is replaced with an integer value (a).
`printf("If the width is %d and the height is %d then the area is %d",20, 30, 20*30);`		There are three `'%d'` directives and three integer values each value is printed instead of one of the `'%d'` directives, Number of `'%d'` directives must equals to the number of numeric values.

Quiz

Code	Output	Description
`printf("A\r\nB\r\nC");`	A B C	`"\r\n"` makes a line break where '`\r`' is the carriage return and '`\n`' is the newline command.

`printf("A\tB\tC\r\n");`
`printf("D\tE\tF\r\n");`
`printf("N\tO\tP\r\n");`
	A B C
	D E F
	N O P
		`'\t'` makes a tab separator.

`printf("A\\B\\C ");`	A\B\C	To print the '`\`' letter you must place '`\\`' instead.
`printf("Say \"Hello\"");`	Say "Hello"	To print the '`"`' letter you must place '`\"`' instead.
`int a = 20*30;` `printf("Area is %d",a);`	Area is 600	The directive `'%d'` is replaced with an integer value (a).
`printf("If the width is %d and the height is %d then the area is %d",20, 30, 20*30);`	If the width is 20 and the height is 30 then the area is 600	There are three `'%d'` directives and three integer values each value is printed instead of one of the `'%d'` directives, Number of `'%d'` directives must equals to the number of numeric values.

Expect The Output

`scanf("%d/%d",&W,&H); printf("\r\nArea is %d ",W*H);`	integer value. The combination `'%d/%d'` is used to scan two integer value separated by `'/'`.
`int x = 172; printf("X equals %x ",x);`	The directive `'%x'` prints the integer value in small hexadecimal format.
`int X = 172; printf("X equals %X ",X);`	The directive `'%x'` prints the integer value in capital hexadecimal format.
`int X; printf("Enter X in hexadecimal format:"); scanf("%X",&X); printf("\r\nX equals %d ",X);`	The directive `'%X'` also used to scan values in hexadecimal format.
`float R = 2.5; printf("R equals %f ",R);`	The directive `'%f'` prints a real (float) value.
`int X = 6235; printf("X equals %10d",X);`	Prints the number in 10 digits including the `'.'` and 2 digits in the fraction part.
`float R = 8372.5675365; printf("R equals %10.2f ",R);`	Prints the number in 10 digits including the `'.'` and 2 digits in the fraction part.
`int X = 15; printf("X equals %05d",X);`	Prints the number in 5 digits and pad it with zeros.

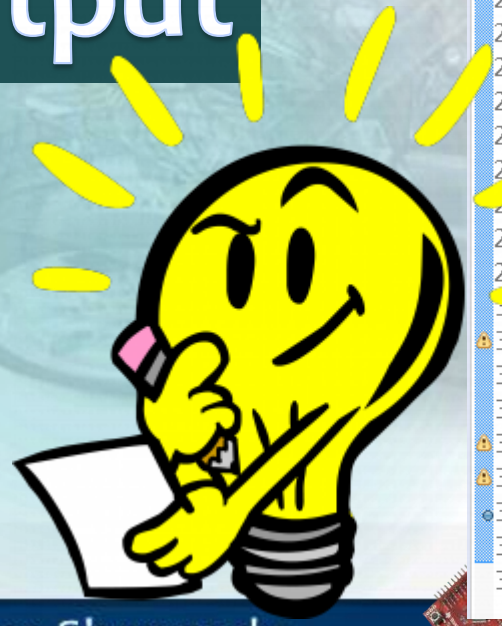
`scanf("%d/%d",&W,&H);` `printf("\r\nArea is %d ",W*H);`		integer value. The combination `'%d/%d'` is used to scan two integer value separated by `'/'`.
`int x = 172;` `printf("X equals %x ",x);`	`X equals ac`	The directive `'%x'` prints the integer value in small hexadecimal format.
`int X = 172;` `printf("X equals %X ",X);`	`X equals AC`	The directive `'%x'` prints the integer value in capital hexadecimal format.

`int X;`
`printf("Enter X in hexadecimal format:");`
`scanf("%X",&X);`
`printf("\r\nX equals %d ",X);`
	`Enter X in hexadecimal format: AC`
	`X equals 172`
		The directive `'%X'` also used to scan values in hexadecimal format.

`float R = 2.5;` `printf("R equals %f ",R);`	`R equals 2.5`	The directive `'%f'` prints a real (float) value.
`int X = 6235;` `printf("X equals %10d",X);`	`X equals 6235` `----------`	Prints the number in 10 digits including the `'.'` and 2 digits in the fraction part.
`float R = 8372.5675365;` `printf("R equals %10.2f ",R);`	`R equals 8372.56` `----------`	Prints the number in 10 digits including the `'.'` and 2 digits in the fraction part.
`int X = 15;` `printf("X equals %05d",X);`	`X equals 00015` `---------`	Prints the number in 5 digits and pad it with zeros.

Printf Tricks

Expect The Output



main.c

```c
//Prepared by keroles
#include <stdio.h>

//Prepared by Keroles
int main()
{
    unsigned char x=0 ;

    printf("Variable width control:\n");
    printf("right-justified variable width: '%*c'\n", 5, 'x');
    printf("left-justified variable width : '%*c'\n", -5, 'x');

    int r = printf("Strings:\n");
    printf("(the last printf printed %d characters)\n", r);

    const char* s = "Hello";
    printf("\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
           s, s, 10, s, 4, s, 10, 4, s);

    printf("Characters:\t%c %%\n", 65);

    printf("Integers\n");
    printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1);
    printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
    printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

    printf("Floating point\n");
    printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);
    printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
    printf("Scientific:\t%E %e\n", 1.5, 1.5);
    printf("Special values:\t 1/0=%g\n", 0.0/0.0, 1.0/0.0);


    printf ("C_trick:\t %d  %d  %d \n",++x,x,x++);
    printf ("C_trick:\t %d  %d  %d \n",x++,++x,x);
    return 0 ;
}
```

FOLLOW US

Press here

70

#LEARN_IN DEPTH

#Be_professional_in embedded_system

oups/embedded.system.KS/

Solution

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCU

<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\fir

```
Variable width control:
right-justified variable width: '    x'
left-justified variable width : 'x    '
Strings:
(the last printf printed 9 characters)
        [     Hello]
        [Hello     ]
        [     Hello]
        [Hell      ]
        [Hell      ]
Characters:     A %
Integers
Decimal:        1 2 000003 0  +4 4294967295
Hexadecimal:    5 a A 0x6
Octal:  12 012 04
Floating point
Rounding:       1.500000 2 1.50000000000000000000000000000000000000
Padding:        01.50 1.50  1.50
Scientific:     1.500000E+000 1.500000e+000
Special values:  1/0=-1.#IND
C_trick:         2  1  0
C_trick:         3  3  2
```

main.c

```c
//Prepared by keroles
#include <stdio.h>

//Prepared by Keroles
int main()
{
    unsigned char x=0 ;

    printf("Variable width control:\n");
    printf("right-justified variable width: '%*c'\n", 5, 'x');
    printf("left-justified variable width : '%*c'\n", -5, 'x');

    int r = printf("Strings:\n");
    printf("(the last printf printed %d characters)\n", r);

    const char* s = "Hello";
    printf("\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
           s, s, 10, s, 4, s, 10, 4, s);

    printf("Characters:\t%c %%\n", 65);

    printf("Integers\n");
    printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1);
    printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
    printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

    printf("Floating point\n");
    printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);
    printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
    printf("Scientific:\t%E %e\n", 1.5, 1.5);
    printf("Special values:\t 1/0=%g\n", 0.0/0.0, 1.0/0.0);


    printf ("C_trick:\t %d  %d  %d \n",++x,x,x++);
    printf ("C_trick:\t %d  %d  %d \n",x++,++x,x);
    return 0 ;
}
```

*main.c

```c
//Prepared by Keroles
int main()
{
    printf("Strings:\n");

    const char* s = "Hello";
    printf("\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
        s, s, 10, s, 4, s, 10, 4, s);

    printf("Characters:\t%c %%\n", 65);

    printf("Integers\n");
    printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1);
    printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
    printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

    printf("Floating point\n");
    printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);
    printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
    printf("Scientific:\t%E %e\n", 1.5, 1.5);
    printf("Hexadecimal:\t%a %A\n", 1.5, 1.5);
    printf("Special values:\t0/0=%g 1/0=%g\n", 0.0/0.0, 1.0/0.0);

    printf("Variable width control:\n");
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first_c_code.exe (3/17/17, 12:08 PM)

```
Strings:
        [     Hello]
        [Hello     ]
        [     Hello]
        [Hell      ]
        [Hell      ]
Characters:     A %
Integers
Decimal:        1 2 000003 0  +4 4294967295
Hexadecimal:    5 a A 0x6
Octal:  12 012 04
Floating point
Rounding:       1.500000 2 1.50000000000000000000000000000000
```

Calculator

PROGRAMMER

4,294,967,295

HEX FFFF FFFF
DEC 4,294,967,295
OCT 37 777 777 777
BIN 1111 1111 1111 1111 1111 1111 1111 1111

QWORD | MS

0000 (60)	0000 (56)	0000 (52)	0000 (48)
0000 (44)	0000 (40)	0000 (36)	0000 (32)
1111 (28)	1111 (24)	1111 (20)	1111 (16)
1111 (12)	1111 (8)	1111 (4)	1111 (0)

Mathematical and Logical Expressions

C language supports following expression operators:

=	Equal operator. X = Y; means copy the value of Y into X
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Division Reminder (Mod) (EX: 15%4 → 3)
( )	Mathematical Parenthesis
++	Increment by one operator
--	Decrement by one operator
|	Logical OR operator
&	Logical AND operator
^	Logical XOR operator
~	Logical NOT operator
>>	Shift Right Operator
<<	Shift Left Operator
+=	Self addition operator. Ex: X+=2 means X = X + 2
Other self operators -=, *=, /=,%=, |=, &=, ^=, >>=, <<=	

Mathematical and Logical Expressions

Following examples provides some specific C expressions:

C Expression	Meaning
`X = X + 9;`	Calculate X+9 then stores the result in X
`X++;`	Add one to X
`X--;`	Subtract one from X

`X = 10;`
`Y = 5;`
`X = X + Y++;`
	Add X+Y → 15
	then increment Y → 6
	Store 15 in X

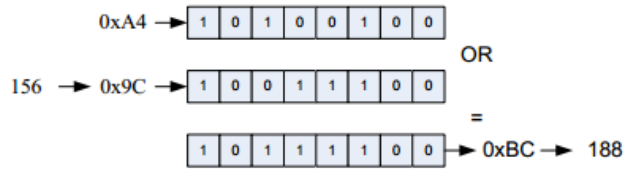
`X = 10;`
`Y = 5;`
`X = X + ++Y;`
	Increment Y → 6
	Add X+Y → 16
	Store 16 in X

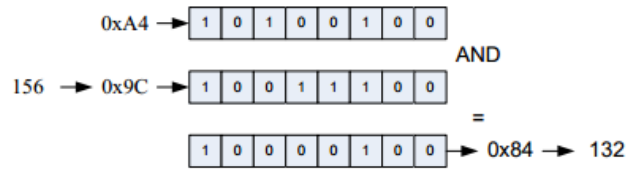
`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X|Y;`
	Calculate the X OR Y → 0xBC (188)

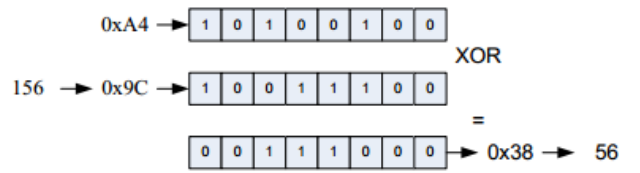
`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X&Y;`
	Calculate the X AND Y → 0x84 (132)

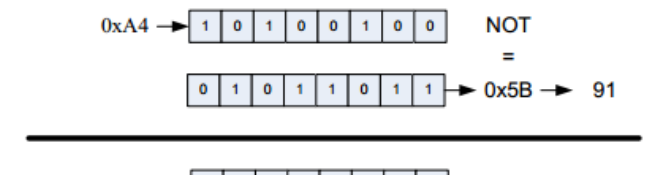
`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X^Y;`
	Calculate the X XOR Y → 0x38 (56)

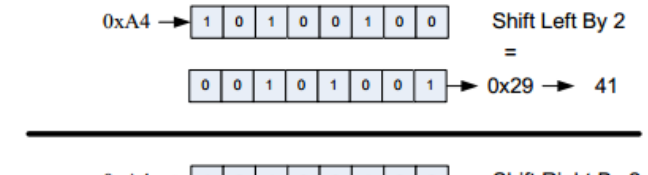
`unsigned char X = 0xA4;` `unsigned char Z = ~X;`	Calculate the (NOT X) → 0x5B (91)
`unsigned char X = 0xA4;` `unsigned char Z = X>>2;`	Calculate the X shifted by two bits to right → 0x29 (41)
`unsigned char X = 0xA4;` `unsigned char Z = X<<2;`	Calculate the X shifted by two bits to right → 0x90 (140)

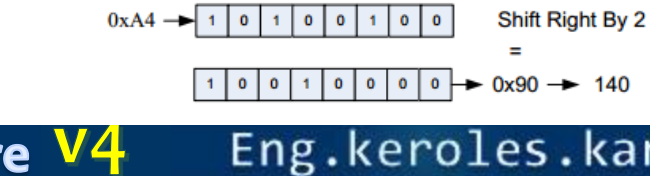












Mathematical and Logical Expressions

C Expression	Meaning
`X = X + Y * Z;`	Multiply Y by Z then Add X
`X = (X + Y) * Z;`	Add X to Y then multiply by Z

Coding Convention

- Coding convention is a set of rules that enhance the readability and the understandability of the code. At the end of each chapter a list of standard and related coding convention is mentioned.

Indentation means arranging the code inside the brackets. Following example shows a non-arranged code.

```
      #include              "stdio.h"

void main()

{
  int x     =6,     y=7;
      int z;
z = x+y                   ;
printf(      "z = %d", z);
                  }
```

Coding Convention

It appears that the above code is unreadable.

First Mistake: New programmers usually use spaces to push some lines forward.

```
  int x     =6,     y=7;
int z;
```

Second Mistake: More than one space is used to separate code sections.

```
x     =6
```

Third Mistake: The code in different lines is not aligned.

Following example shows an arranged code.

```
#include "stdio.h"

void main()
{
      int x = 6, y = 7;
      int z;
      z = x + y;
      printf("z = %d", z);
}
```

Generally following roles must be obeyed to

First: Use One Tab to push the whole code between any two packets

```
{
	int x = 6, y = 7;
	int z;
	z = x + y;
	printf("z = %d", z);
}
```

Second: Use One Space to separate small code segments. For example:

```
int x = 6, y = 7;
```

Third: Use One Line Only to separate code blocks. For example:

```
#include "stdio.h"

void main()
{
	...
}
```

Fourth: Do not leave lines between the head of the block and the body of it. For example, do not leave a line between (`void main()`) and its body:

```
void main()
{
```

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

C Fundamentals

Statements

- Control Statements
  - if
  - switch
  - goto
  - for loop
  - while loop
  - do-while loop
  - break
  - continue
  - Nested Loop
- null statement
- expression statement

Comments

Operators

Relational operators

```
<
>
<=
>=
==
!=
```

Logical operators

Operator
&&
||
!

Assignment operators

```
operator
+=
-=
*=
/=
%=
```

Arithmetic operators

Operator
+
-
*
/
%

Conditional operators

```
min = (x < y) ? x : y;
Identifier = (test expression)? Expression1: Expression2 ;
```

Comma operators

```
int i , j;
i=(j=10,j+20);
```

A set of expression separated by comma is a valid constant in the C language

Bitwise Operators

~	Bitwise Negation one's complement (unary)
&	Bitwise And
|	Bitwise Or
^	Bitwise Exclusive Or
>>	Right Shift by right hand side (RHS) (divide by power of 2)
<<	Left Shift by RHS (multiply by power of 2)

Constants

1. Integer constants
2. Floating-point constants
3. Character constants
4. String constants

Keywords

Identifiers

Data Types

- User Defined
  - enum
  - typedef
- Primitive/Basic Types
  - Real Values
  - Integer Values
    - unsigned
    - signed
- Derived
  - Arrays
  - pointer
  - structure
  - union

https://www.facebook.com/groups/embedded.system.KS/

Identifiers

- Identifiers are the names that are given to various program elements such as **variables, symbolic constants and functions**.
- Identifier can be freely named, the following restrictions.
  - Alphanumeric characters (a ~ z , A～Z , 0～9) and half underscore (_) can only be used.
  - The first character of the first contain letters (a ~ z , A～Z) or half underscore (_) can only be used.

Identifiers

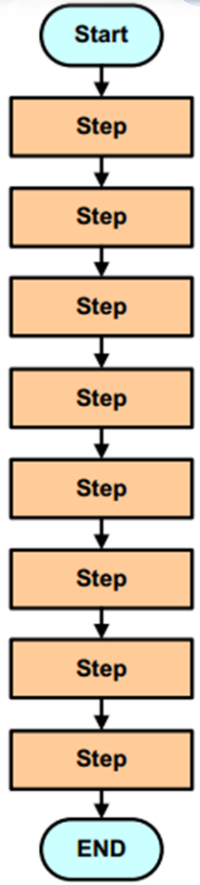
- Here are the rules you need to know:
- 1. Identifier name must be a sequence of letter and digits, and must begin with a letter.
- 2. The underscore character (‘_’) is considered as letter.
- 3. Names shouldn't be a **keyword** (such as int , float, if ,break, for etc)
- 4. Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
- 5. No identifier may be keyword.
- 6. No special characters, such as semicolon,period,blank space, slash or comma are permitted
- Examples of legal and illegal identifiers follow, first some legal identifiers:
  - float _number;
  - float a;
- The following are illegal (it's your job to recognize why):
- float :e; float for; float 9PI; float .3.14; float 7g;

Keywords

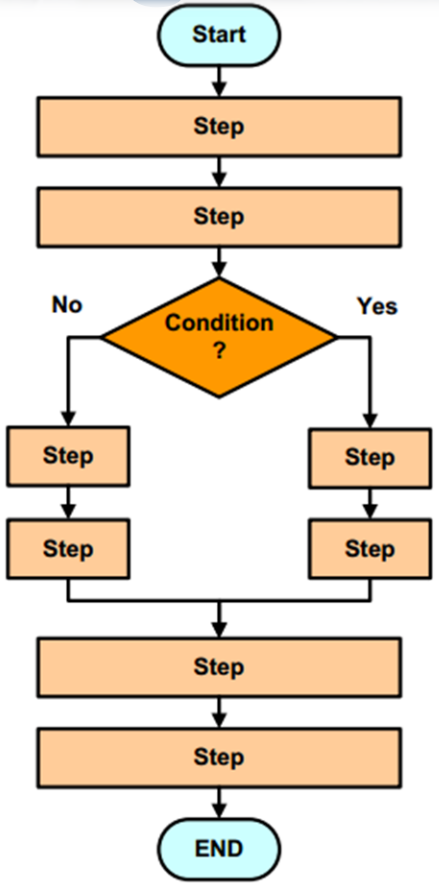
- Keywords are standard identifiers that have standard predefined meaning in C. Keywords are all lowercase, since uppercase and lowercase characters are not equivalent it's possible to utilize an uppercase keyword as an identifier but it's not a good programming practice.
- 1. Keywords can be used only for their intended purpose.
- 2. Keywords can't be used as programmer defined identifier.
- 3. The keywords can't be used as names for variables.
- The standard keywords are given below:

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

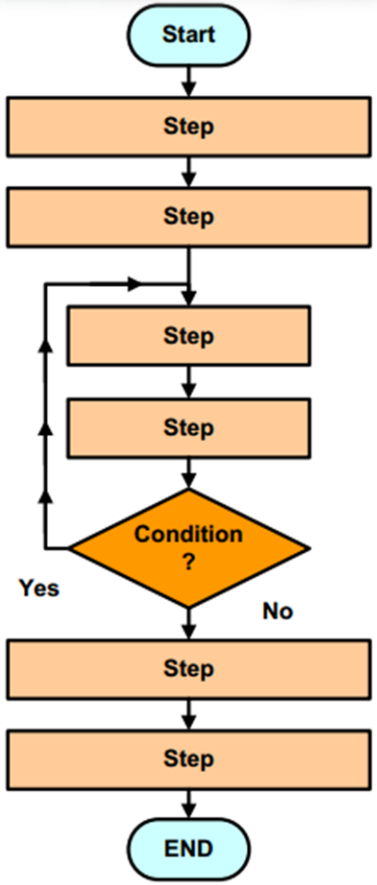
Controlling Program Flow



No Loops
No Decisions



Decision



Loop

Conditions

Operator	Meaning
>	Greater
>=	Greater or equal
<	Less
<=	Less than or equal
==	Equal
!=	Not equal
!	Not If the input is true the output is false if the input is false the output is true
&&	And Example: A>B && C>D If both sides are true the output is true, otherwise it gives false
||	Or Example: A>B || C>D If wither sides is true the output is true, otherwise it gives false

Example : Using Condition

```c
#include "stdio.h"
#include "math.h"
void main()
{
int a = 9;
int b = 8;
int c = 12;
printf("%d\r\n", a>b);
printf("%d\r\n", b>c);
printf("%d\r\n", a<=9);
printf("%d\r\n", a!=9);
printf("%d\r\n", (a-b)>(c-b));
printf("%d\r\n", a>b && c>b);
printf("%d\r\n", a>b && c<b);
printf("%d\r\n", a>b || c<b);
printf("%d\r\n", !(a<b));
printf("%d\r\n", 3 && 0);
printf("%d\r\n", -15 || 0);
printf("%d\r\n", !(-15));
}
```

Expect The Output

Example :Using Conditions

```c
#include "stdio.h"
#include "math.h"
void main()
{
int a = 9;
int b = 8;
int c = 12;
printf("%d\r\n", a>b); //prints 1
printf("%d\r\n", b>c); //prints 0
printf("%d\r\n", a<=9); //prints 1
printf("%d\r\n", a!=9); //prints 0
printf("%d\r\n", (a-b)>(c-b)); //prints 0
printf("%d\r\n", a>b && c>b); //prints 1
printf("%d\r\n", a>b && c<b); //prints 0
printf("%d\r\n", a>b || c<b); //prints 1
printf("%d\r\n", !(a<b)); //prints 1
printf("%d\r\n", 3 && 0); //prints 0
printf("%d\r\n", -15 || 0); //prints 1
printf("%d\r\n", !(-15)); //prints 0
}
```

if Statement

```
if(/*if condition*/)
{
//if body
}
else if(/*else if condition*/)
{
//else if body
}
else if(/*else if condition*/)
{
//else if body
}
else
{
//else body
}
```

Calculate Circle Area or Circumfere

Lab1
Calculate
Circle
Area
or
Circumference

In this program the user has to choose between calculating circle area or circle circumference. The choice comes by taking a character from the keyboard using the (getche) function. If the user presses „a" character it proceeds with area calculation and printing. If the user presses „c" character it proceeds with circumference calculation and printing. If the user presses other letters the program prints an error message.

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17, 3:00 PM)

```
Enter circle radius : 3
Enter your choice (a to print the area,c to print the circumference) : a

area is 28.274309
```

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17, 3:00 PM)

```
Enter circle radius : 3
Enter your choice (a to print the area,c to print the circumference) : c

circumference is 18.849541
```

*main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */

#include <stdio.h>

int main(int argc, char **argv) {

    char choice;
    float radius;
    float area, circumference;
    printf("Enter circle radius : ");
    fflush(stdin); fflush(stdout);
    scanf("%f", &radius);
    printf("Enter your choice (a to print the area,c to print the circumference) : ");
    fflush(stdin); fflush(stdout);
    scanf("%c", &choice);
    if(choice=='a')
    {
    area = 3.14159 * radius * radius;
    printf("\r\narea is %f\r\n", area);
    }
    else if(choice=='c')
    {
    circumference = 2 * 3.14159 * radius;
    printf("\r\ncircumference is %f\r\n",
    circumference);
    }
    else
    printf("\r\nwrong choice\r\n");

}
```

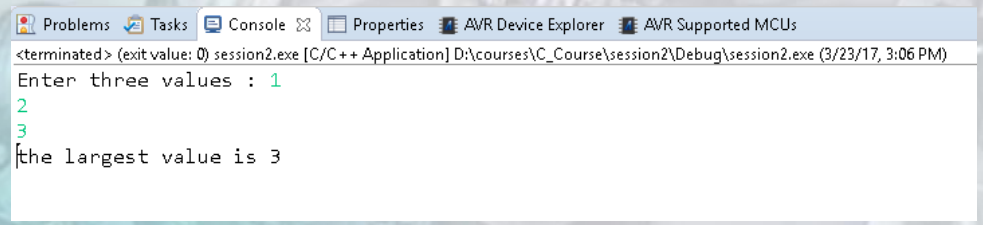
Lab1 Calculate Circle Area or Circumference Solution

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Comparing Three Numbers

This program finds the largest value of the three given values.



Lab2
Comparing
Three
Numbers

#LEARN_IN DEPTH

#Be_professional_in embedded_system

main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */

#include <stdio.h>

int main(int argc, char **argv) {

    int a, b, c;
    printf("Enter three values : ");
     fflush(stdin); fflush(stdout);
    scanf("%d %d %d", &a, &b, &c);
    if(a>b)
    {
        if(a>c)
            printf("the largest value is %d\r\n", a);
        else
            printf("the largest value is %d\r\n", c);
    }
    else
    {
        if(b>c)
            printf("the largest value is %d\r\n", b);
        else
            printf("the largest value is %d\r\n", c);
    }
}
```

Lab2 Comparing Three Numbers Solution

Inline condition / Conditional operators

```
min = (x < y) ? x : y;
Identifier = (test expression)? Expression1: Expression2 ;
```

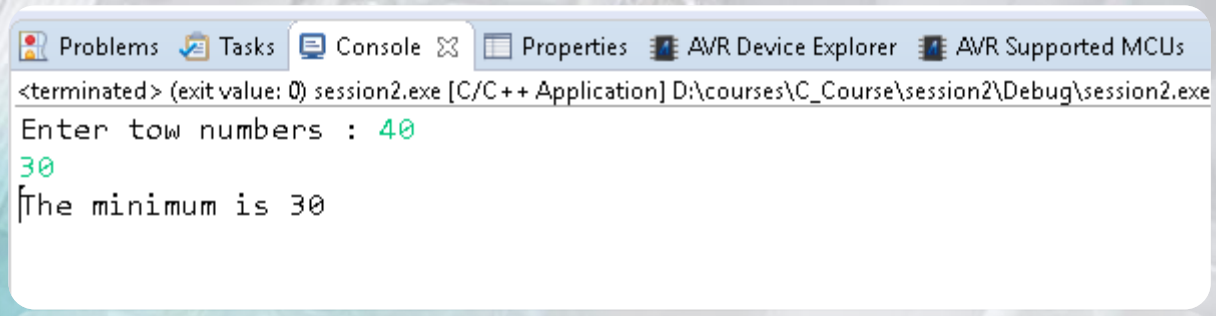
- Sometimes it is required to take a fast decision inside your statements; this is called the inline condition. Following examples illustrate the idea.

```
#include "stdio.h"

void main()
{
	int a, b, minimum;
	printf("Enter tow numbers : ");
	scanf("%d %d", &a, &b);
	minimum = (a<b)?a:b;
	printf("The minimum is %d\r\n", minimum);
}
```

```
minimum = ( a<b ) ? a : b ;
```

- Condition
- Assigned value in case of **true**
- Assigned value in case of **false**

Calculate the Minimum of Two Numbers



Lab3
Calculate
the
Minimum
of Two Numbers

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

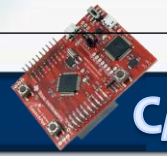
main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */

#include <stdio.h>

int main(int argc, char **argv) {

    int a, b;
    printf("Enter tow numbers : ");
    fflush(stdin); fflush(stdout);
    scanf("%d %d", &a, &b);
    printf("The minimum is %d\r\n", (a<b)?a:b);

}
```

Lab3 Calculate the Minimum of Two Numbers Solution

switch Statement

```
switch(/*switch expression*/)
{
case /*case value*/:
{
//case body
}
break;
...
...
...
case /* case value*/:
{
//case body
}
break;
default:
{
}
break;
}
```



Calculate Circle Area or Circumference

main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    char choice;
    float radius;
    float area, circumference;
    printf("Enter circle radius : ");
    fflush(stdin); fflush(stdout);
    scanf("%f", &radius);
    printf("Enter your choice (a to print the area,c to print the circumference) : ");
    fflush(stdin); fflush(stdout);
    scanf("%c", &choice);
    switch (choice)
    {
    case 'a':
    case 'A':
    {
        area = 3.14159 * radius * radius;
        printf("\r\narea is %f\r\n", area);
    }
    break;
    case 'c':
    case 'C':
    {
        circumference = 2 * 3.14159 * radius;
        printf("\r\ncircumference is %f\r\n",
                circumference);
    }
    break ;
    default:
        printf("\r\nwrong choice\r\n");
        break;
    }

}
```

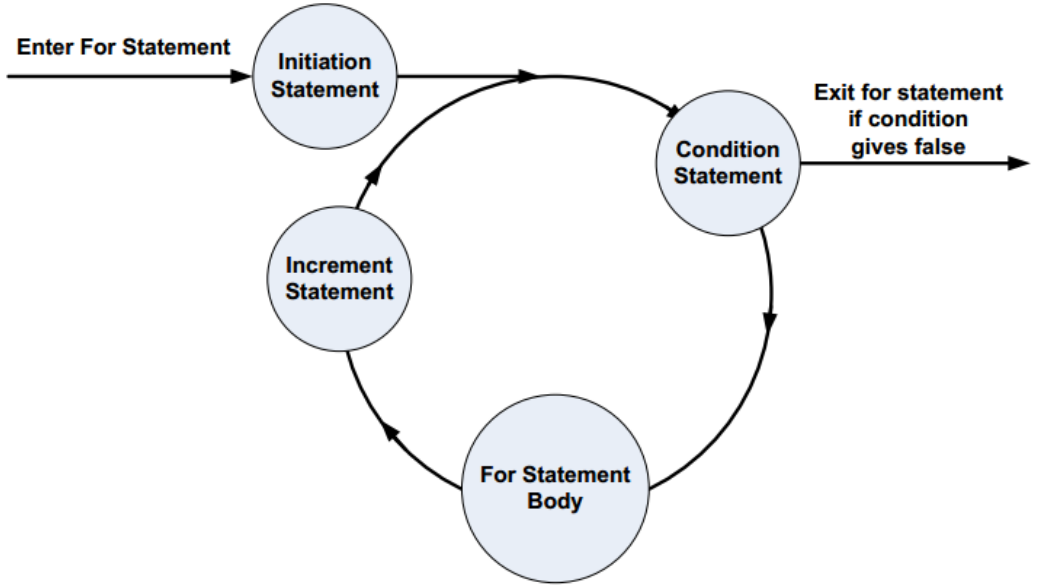

for Statement

Syntax:

```
for(/*intiation*/;/*condition*/;/*increment*/)
{
        //for body
}
```

for statement repeats the execution of the (**for body**) until the (**condition**) statement is not succeeded any more. Computer processes **for** statement as follows:

1. Execute the (**initiation**) statement to assign an initiate value to some variable if it is required.
2. Execute the (**condition**) statement, if false, go out of the for statement, otherwise, proceed to the next step
3. Execute the (**for body**)
4. Execute the (**increment**) statement to update some variables
5. Go back to (Step 2)



Printing Hello World in a Loop

main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i;
    for(i=0;i<10;i++)
    {
    printf("%d : Hello World\r\n", i);
    }

}
```

Problems Tasks Console

<terminated> (exit value: 0) session2.exe [C/C++

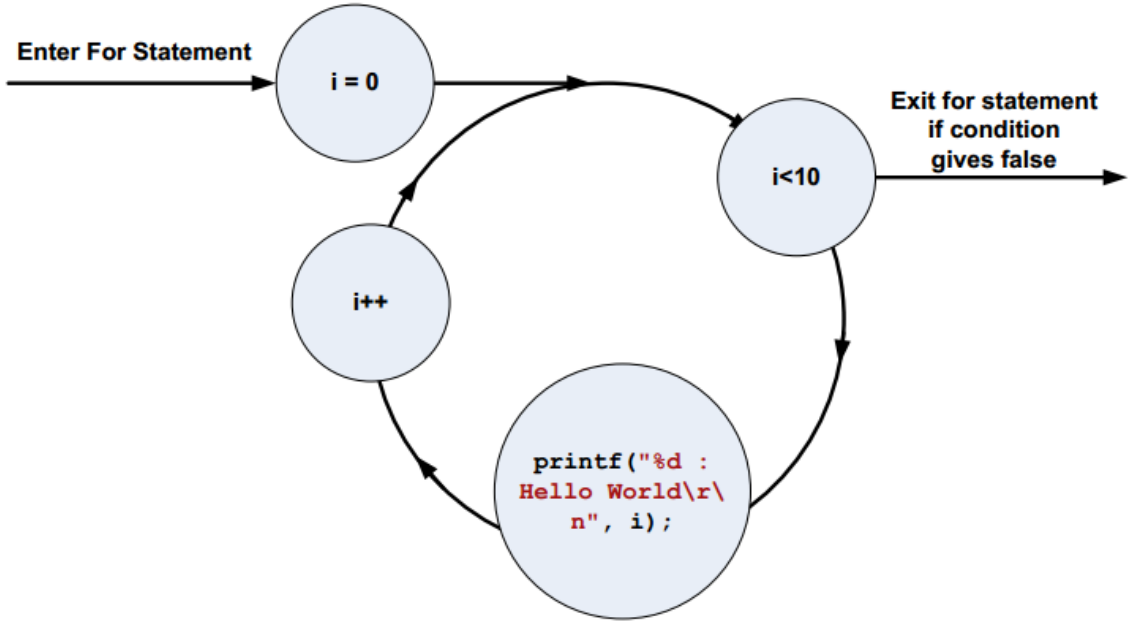
```
0 : Hello World

1 : Hello World

2 : Hello World

3 : Hello World

4 : Hello World

5 : Hello World

6 : Hello World

7 : Hello World

8 : Hello World

9 : Hello World
```

Printing Hello World in a Loop

Above example prints "Hello World" 10 times. The program works as following:

1. Execute the initiation statement (i=0)
2. Execute the condition statement (i<10), if false, exit from the for statement
3. Execute the body of the for statement
4. Execute the increment statement (i++)
5. Go back to (Step 2)

Initially (i) is loaded with (0), after each loop it is incremented by (1). If (i) value reaches (10) the condition statement fails and the computer exits from the loop. It is clear that (i) variable takes the values {0,1,2,3,4,5,6,7,8,9} during the execution.



Calculate the Summation of values between 1 and 99

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17, 3:47 PM)

```
Summation of values between (1 and 99) is : 4950
```

Lab4
Calculate
the
Minimum
of Two Numbers

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Calculate the Summation of values between 1 and 99

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i, sum = 0;
    for(i=1;i<=99;i++)
    {
    sum += i;
    }
    printf("Summation of values between (1 and 99) is : %d",sum);

}
```

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17, 3:47 PM)
Summation of values between (1 and 99) is : 4950
```

Lab4 Calculate the Minimum of Two Numbers Solution

Calculate the Average Students Degrees

calculates the average students degree for any given students number.

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17,

```
Enter the number of the students:7
Enter student (1) degree:12
Enter student (2) degree:13
Enter student (3) degree:14
Enter student (4) degree:15
Enter student (5) degree:16
Enter student (6) degree:17
Enter student (7) degree:18
Average students degree is : 15.000000
```

Lab5

main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i, nStudents;
    float degree, sum;
    printf("Enter the number of the students:");
    fflush(stdin); fflush(stdout);
    scanf("%d", &nStudents);
    for(i=1, sum=0; i<=nStudents; i++)
    {
        printf("Enter student (%d) degree:", i);
        fflush(stdin); fflush(stdout);
        scanf("%f", &degree);
        sum += degree;
    }
    printf("Average students degree is : %f\r\n",
            sum/nStudents);

}
```

Lab5 Solution

while Statement

Syntax:

```
while(/*condition*/)
{
        //while body
}
```

while statement is similar to the **for** statement, however it is more simple, there is no initiation or increment statements, you have to choose where to initiate and where to increment your variables if you need this. The computer executes the while statement as follows:

1. Execute the (**condition**) statement, if false, go out of the **while** statement, otherwise, proceed to the next step
2. Execute the (**while body**)
3. Repeat (Step 1)

Calculate the Summation of odd values between 1 and 99

```c
#include "stdio.h"

void main()
{
	int i = 1, sum = 0; // Initiation is placed here

	while(i<=99)
	{
		sum += i;
		i+=2; // Increment is placed here
	}
	printf("Summation of odd values between
				(1 and 99) is : %d", sum);
}
```

Calculate the Average Students Degrees


```c
#include "stdio.h"

void main()
{
	int nStudents = 0;
	float degree, sum = 0;

	printf("Enter negative value to exit:\r\n");
	while(1)
	{
		printf("Enter student (%d) degree:",
							nStudents + 1);
		scanf("%f", &degree);

		if(degree<0)break; //force exit from while loop

		sum += degree;

		nStudents++;
	}

	printf("Average students degree is : %f\r\n",
							sum/nStudents);
}
```

Important:
break statement is used to exit from any loop type.

do...while Statement

Syntax:

```
do
{
        //do...while body
}
while(/*condition*/);
```

do ... while statement is similar to while statement, except that the condition is checked after executing each loop, which means that, the first loop is performed without a check. The computer executes the while statement as follows:

1. Execute the (**do...while body**)
2. Execute the (**condition**) statement, if false, go out of the **do...while** statement, otherwise go to (Step 1)

Calculate Polynomial Value

```c
#include "stdio.h"
#include "conio.h"

void main()
{
	float x, y;

	do
	{
		printf("\r\nEnter x value:");
		scanf("%f", &x);
		y = 5*x*x + 3*x + 2;
		printf("\r\ny(%f) = %f", x, y);

		printf("\r\ndo you want to evaluate
							again (y/n):");
	}
	while(getche()=='y');
}
```

goto Statement

Syntax:

```
        // C Statment
labelname:
        // C Statment
        // C Statment
        goto labelname;
        // C Statement
```

```
        // C Statment
        goto labelname;
        // C Statment
        // C Statment
labelname:
        // C Statment
```

Simply **goto** statement tells the program where to jumps, it can jump forward or backward. Following example illustrates the idea.

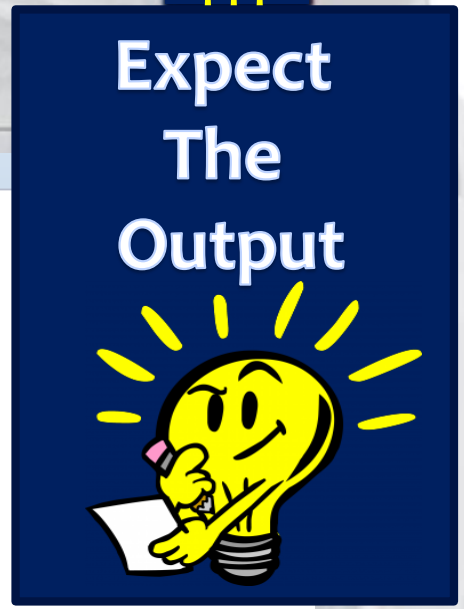
goto Statement

```
#include "stdio.h"
#include "conio.h"

void main()
{
	float x, y;

evaluate_again:

	printf("\r\nEnter x value:");
	scanf("%f", &x);
	y = 5*x*x + 3*x + 2;
	printf("\r\ny(%f) = %f", x, y);

	printf("\r\ndo you want to evaluate again (y/n):");

	if(getche()=='y')
		goto evaluate_again;
}
```

Important: It is **not recommended** to use **goto** statement extensively, because it allows programmers to jump anywhere in their program and this lead to unorganized and unreadable codes.

break statement

- The **break** statement is a jump instruction and can be used inside a **switch** construct, for **loop**, **while** loop and **do-while** loop.
- The execution of break statement causes immediate exit from the concern construct and the control is transferred to the statement following the loop.



main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i;

       for(i=0;i<10;i++)
       {
         if(i==5)
         {
            printf("\nComing out of for loop when i =");
            break;
         }
         printf("%d ",i);
       }

}
```

break statement

- The **break** statement is a jump instruction and can be used inside a **switch** construct, for **loop**, **while** loop and **do-while** loop.
- The execution of break statement **causes immediate exit** from the concern construct and the control is transferred to the statement following the loop.

main.c

```
1  /*
2   * main.c
3   *
4   *  Created on: Mar 23, 2017
5   *      Author: Keroles
6   */
7  #include <stdio.h>
8
9  int main(int argc, char **argv)
10
11     int i;
12
13        for(i=0;i<10;i++)
14        {
15          if(i==5)
16          {
17              printf("\nComing out of for loop when i =");
18              break;
19          }
20          printf("%d ",i);
21        }
22
23 }
24
```

Problems | Tasks | Console | Properties

<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\

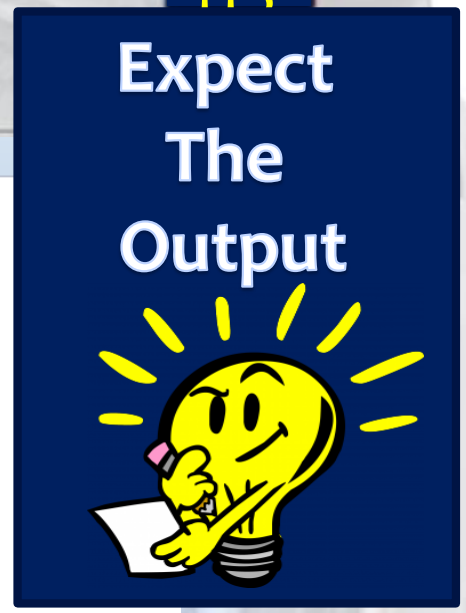
```
0 1 2 3 4
Coming out of for loop when i =
```

continue statement

- Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- Syntax : continue;

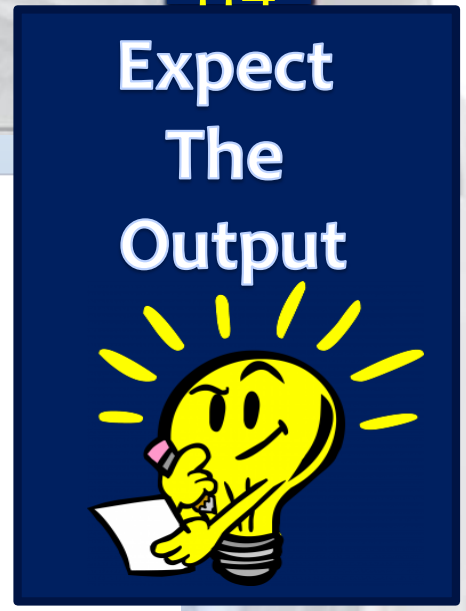
main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i;
     for(i=0;i<10;i++)
     {
       if(i==5 || i==6)
       {
         printf("\nSkipping %d from display using " \
         "continue statement \n",i);
         continue;
       }
       printf("%d ",i);
     }

}
```



continue statement

- Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- Syntax : continue;



main.c

```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **argv) {

    int i;
     for(i=0;i<10;i++)
     {
       if(i==5 || i==6)
       {
         printf("\nSkipping %d from display using " \
         "continue statement \n",i);
         continue;
       }
       printf("%d ",i);
     }

}
```

Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Sup

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Deb
0 1 2 3 4
Skipping 5 from display using continue statement

Skipping 6 from display using continue statement
7 8 9
```

Nested loop

- In many cases we may use loop statement inside another looping statement. This type of looping is called nested loop

The syntax for a **nested while loop** statement in C programming language is as follows –

```
while(condition) {

   while(condition) {
      statement(s);
   }

   statement(s);
}
```

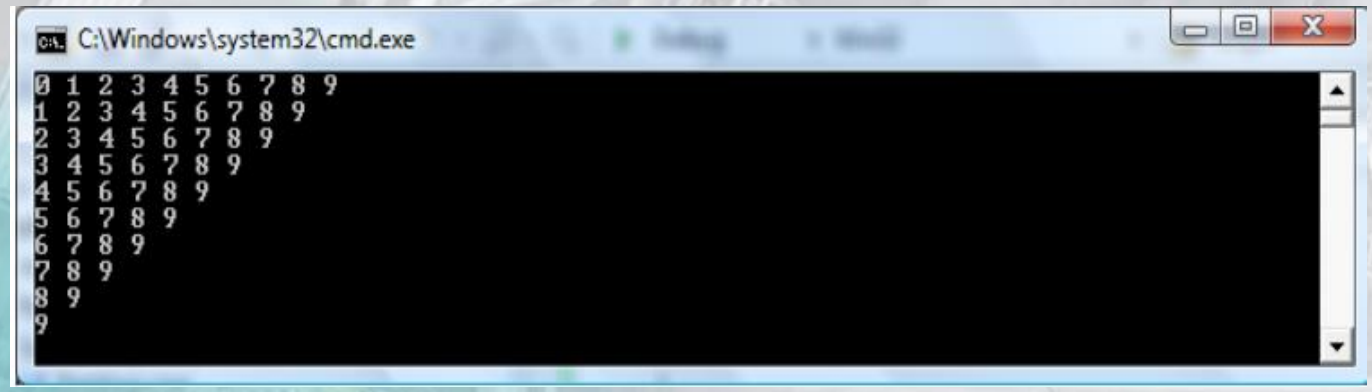
The syntax for a **nested for loop** statement in C is as follows –

```
for ( init; condition; increment ) {

   for ( init; condition; increment ) {
      statement(s);
   }

   statement(s);
}
```

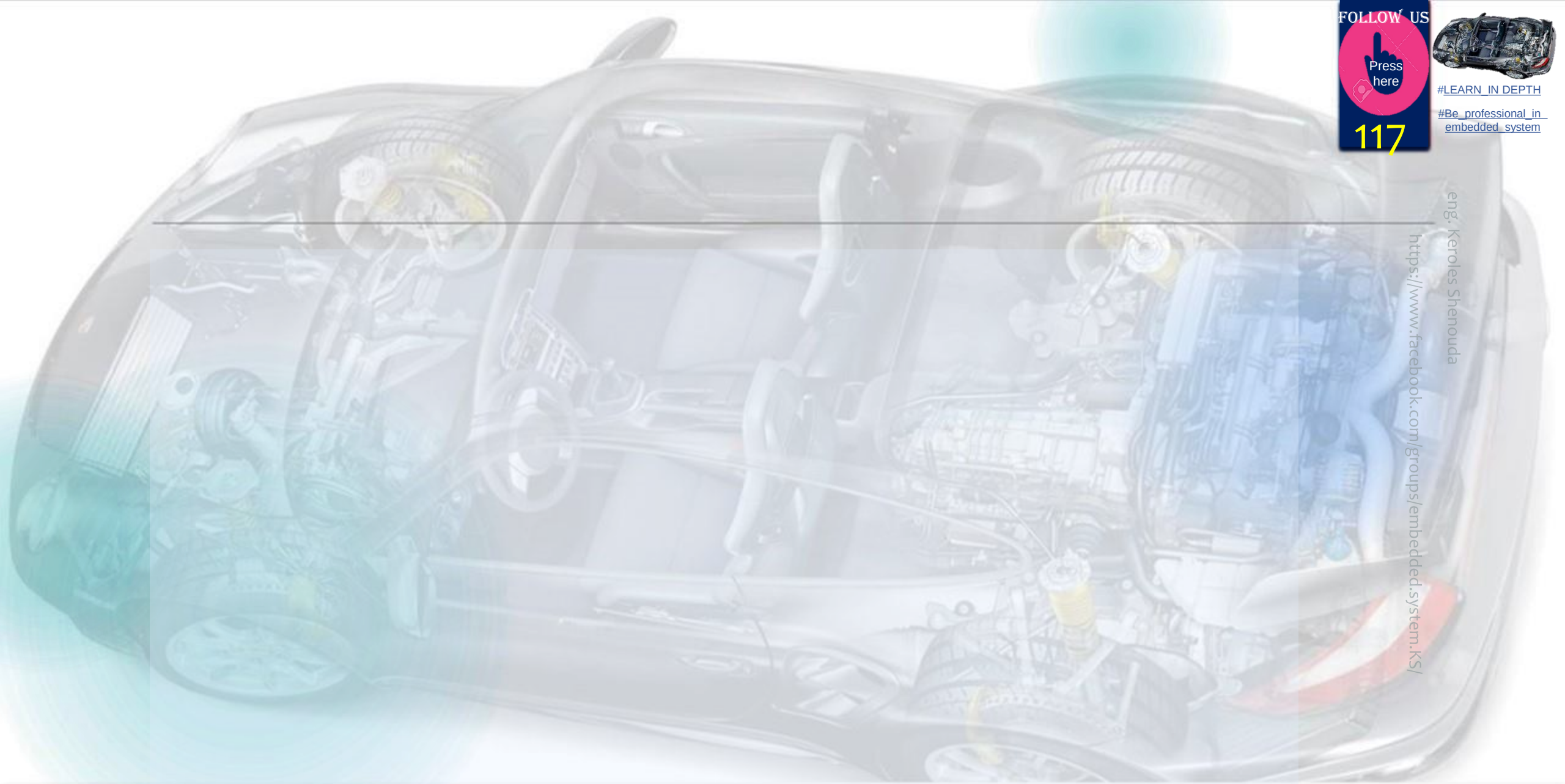
The syntax for a **nested do...while loop** statement in C programming language is as follows –

```
do {

   statement(s);

   do {
      statement(s);
   }while( condition );

}while( condition );
```

Write a program that produces the following output:



Lab



#LEARN_IN DEPTH

#Be_professional_in_embedded_system