BME3321:Introduction to Microcontroller Programming

Topic 3: C programming language basics, Data types, Variables, Arrays, Loops, Conditionals, Functions, Pointers, Structures...

Asst. Prof. Dr. İsmail Cantürk

C How To Program (ch2,3,4,5,6,7,8,10)

Welcome to C programming – A simple C program

```
#include <stdio.h>
/* function main begins program execution */
int main( void )
{
printf( "Welcome to C!\n" );
return 0; /* indicate that program ended successfully */
} /* end function main */
```

This line tells the preprocessor to include the contents of the standard input/output header file(<stdio.h>) in the program

This line is a part of every C program. Main is a program building block called a function. Left and right braces {} are limits of main.

A function defined in the stdio.h to display the quoted message on the screen.

Used to exit a function. The value 0 indicates that the program has terminated successfully.

- /* comment lines*/ or // comment line

- \n Newline. Position the cursor at the beginning of the next line.
- \t Horizontal tab. Move the cursor to the next tab stop.

- Do not forget semicolons (;) for statements.

Development enviroments for C

- Google ‘online C compiler’
- Try this one
  https://www.programiz.com/c-programming/online-compiler/

- Or, download Dev-C++ to your computer and install it.
- https://sourceforge.net/projects/orwelldevcpp/
- File->New->Source File

- Or, you can use Microsoft visual studio

Arithmetic and decision making

- C programs perform arithmetic calculations.

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Arithmetic and decision making

- Decision making with operators.

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

- Do not confuse equal (==) with assigning (=) operator.
- x=y →Value of y is copied to x.

```
if (x==y)
{
//Do this
}
```

```
if (x=y)
{
//Do this
}
```

if, else Control structures

```
if (condition)
{
//Do this
}
```

Syntax for if

```
if ( grade >= 40 ) {
printf( "Passed\n" );
}
else {
printf( "Failed\n" );
}
```

Conditional operator (?:) → similar to if else

```
grade >= 40 ? printf( "Passed\n" ) : printf( "Failed\n" );
```

```
if ( grade >= 90 )
printf( "A\n" );
else if ( grade >= 80 )
printf( "B\n" );
else if ( grade >= 70 )
printf( "C\n" );
else if ( grade >= 60 )
printf( "D\n" );
else
printf( "F\n" );
```

Nested if else statements

switch Multiple-Selection Statement

Syntax for *Switch Case* Statement:

```
switch (variable or an integer expression)
{
    case constant:
    //C Statements
    ;
    case constant:
    //C Statements
    ;
    default:
    //C Statements
    ;
}
```

```
#include <stdio.h>
int main()
{
    int i;
    printf("Enter i value as 1,2,3, or 4\n");
    scanf("%d",&i);
    printf("Entered value %d\n",i);

    switch (i)
    {
        case 1:
          printf("Case1 ");
          break;
        case 2:
          printf("Case2 ");
          break;
        case 3:
          printf("Case3 ");
          break;
        case 4:
          printf("Case4 ");
          break;
        default:
          printf("Default ");
    }
    return 0;
}
```

Assignment, increment and decrement operators

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Operator	Sample expression	Explanation
`++`	`++a`	Increment a by 1, then use the new value of a in the expression in which a resides.
`++`	`a++`	Use the current value of a in the expression in which a resides, then increment a by 1.
`--`	`--b`	Decrement b by 1, then use the new value of b in the expression in which b resides.
`--`	`b--`	Use the current value of b in the expression in which b resides, then decrement b by 1.

Try out this code

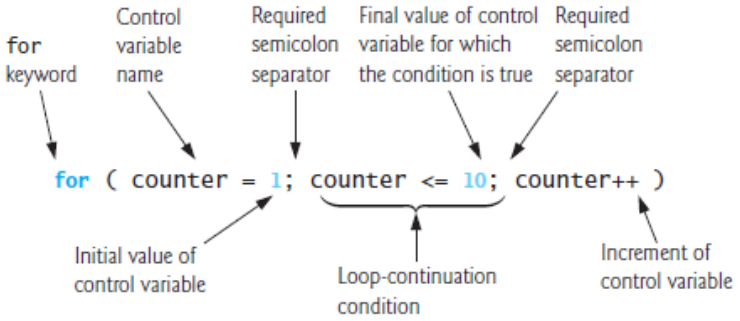
```
#include <stdio.h>
 int main( void )
 {
int c;
c = 5;
printf( "%d\n", c );
printf( "%d\n", c++ );
printf( "%d\n", c );
}
```

for Loops or repetition statements

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
```

Syntax of for loop

```
#include <stdio.h>
int main( void )
 {
 int counter;

 for ( counter = 1; counter <= 10; counter++ )
 {
 printf( "%d\n", counter );
 }
 return 0;
}
```



while Loops

```
while (condition)
{
   // do this
}
```

Syntax for while

```
product= 3;
while ( product <= 100 )
{
product = 3 * product;
}
```

Run this code:

```
#include <stdio.h>
int main()
{
        int product=3;
        while (product<=100)
        {
                product=3*product;
                printf("%d\n",product);
        }
return 0;
}
```

do…while Loops

- The *do…while* repetition statement is similar to the *while* statement.

```
do{
   // statements
}
while (condition) ;
```

Syntax for do…while

Run this code:

```
#include <stdio.h>
int main()
{
            int product=3;
            do{
            product=3*product;
            printf("%d\n",product);
                }
            while (product<=100);
  return 0;
}
```

break and *continue* statements

- *break* statement is used to terminate a loop or to skip the remainder of a switch statement.

```
#include <stdio.h>
int main()
{
    int num =0;
    while(num<=100)
    {
      printf("value of variable num is: %d\n", num);
      if (num==2)
      {
         break;
      }
      num++;
    }
    printf("Out of while-loop");
    return 0;
}
```

break and *continue* statements

- *continue* is used to skip the remaining statements and performs the next iteration of the loop.

```
#include <stdio.h>
int main()
{
    int num =0;
    while(num<=10)
    {
      num++;
      if (num==5)
      {
         continue;
      }
      printf("value of variable num is: %d\n", num);
    }
    printf("Out of while-loop");
    return 0;
}
```

Logical operators

&& (logical AND)

```
if ( a > 20 && b >20 ){
a= a+1;
}
```

|| (logical OR)

```
if (a > 20 || b >20 ){
b= b+1;
}
```

! (Logical NOT)

```
if ( !( a == b ) )
{
// do this
}
```

Do NOT confuse *equality (==)* and *assignment (=)* operators

```
if ( x==y ){
// do this
}
```

```
if (x=y){
//do this
}
```

Logic error. Any nonzero value will be interpreted as “true”

C functions

- Functions allow you to modularize a program.

```
#include <stdio.h>

int square( int y );

int main( void )
{
int x;

for ( x = 1; x <= 10; x++ )
{
printf( "%d\n ", square (x));
}

return 0;
}

int square( int y )
{
return y * y;
}
```

function prototype. The int in parentheses informs the compiler that square expects to receive an integer value from the caller.

Call of the function

Function definition

- The format of a function definition is:

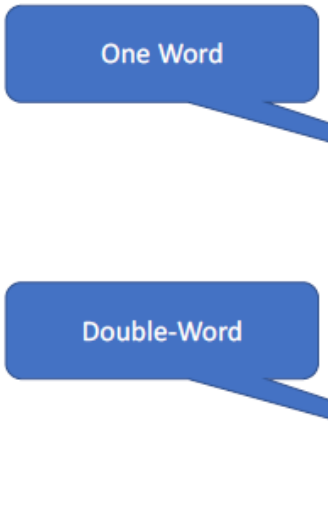
return-value-type function-name(*parameter-list*)
{
definitions
statements
}

C functions

```
#include <stdio.h>

int maximum( int x, int y, int z ); // function prototype

int main( void )
{
int number1; int number2; int number3;
printf( "Enter three integers: " );
scanf( "%d%d%d", &number1, &number2, &number3 );
printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
return 0;
}

int maximum( int x, int y, int z ) // function definition
{
int max = x;
if ( y > max ) {
max = y;
}
if ( z > max ) {
max = z;
}
return max; /* max is largest value */
}
```

Data types

- char, short, integer...
- signed, unsigned



TYPE	BITS	MINIMUM	MAXIMUM	DECIMAL FORMAT
Unsigned char	8	0	255	Integer
Signed char	8	-128	127	Integer
Unsigned short	16	0	65535	Integer
Signed short	16	-32768	32767	Integer
Unsigned int	32	0	4294967295	Integer
Signed int	32	-2147483648	2147483647	Integer
Float (IEEE754)	32	-3.4028E+38	3.4028E+38	Real number
Double (IEEE754)	64	-1.7977E+308	1.7977E+308	Real number

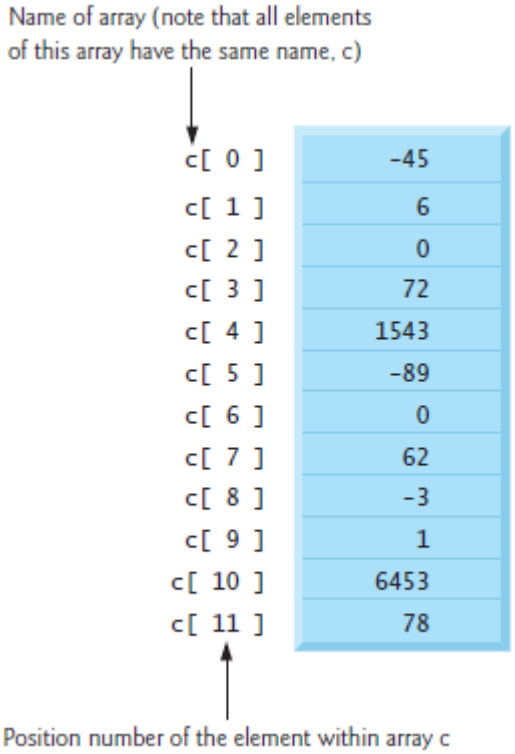
Data type specifications

Data type	printf conversion specification	scanf conversion specification
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

sizeof() operator

```
#include <stdio.h>
int main()
{
		int a = 1;
		char b = 'G';
		double c = 3.14;

		printf("Hello! I am an integer. My value is %d and my size is %d bytes.\n", a, sizeof(int));
		// can use sizeof(a) above as well

		printf("Hello! I am a character. My value is %c and my size is %d byte.\n",b, sizeof(char));
		// can use sizeof(b) above as well

		printf("Hello! I am a double floating point variable and my value is %lf and my size is %d bytes.\n",
				c, sizeof(double));
		// can use sizeof(c) above as well

		return 0;
}
```

C arrays



- An array is a group of memory locations .
- They all have the same name and the same type.

x = c[6] / **2**;

y = c[7] % c[1]

Array definitions

```
int c[ 12 ];
```

→ Computer reserves appropriate amount of memory for 12 integer elements for array c.

```
int b[ 100 ], x[ 27 ];
```

→ Computer reserves 100 elements for integer array b and 27 elements for integer array x.

```
int n[ 10 ] = { 0 };
```

→ If there are fewer initializers than elements in the array, the remaining elements are initialized to zero. For example, the elements of this array were initialized to zero.

```
int n[ 5 ] = { 32, 27, 64, 18, 95};
```

→ All elements were defined. More element definitions will cause syntax error.

```
int n[] = { 1, 2, 3, 4, 5 };
```

→ would create a five-element array.

Array example

Try out this code

```
#include <stdio.h>
#define SIZE 10
int main( void )
{
        int j;
        int s[ SIZE ];
        for ( j = 0; j < SIZE; j++ )
        {
                s[ j ] = 2 + 2 * j;
        }

        for ( j = 0; j < SIZE; j++ )
        {
                printf( "%d    %d\n", j, s[ j ] );
        }
         return 0;

}
```

defines a symbolic constant SIZE whose value is 10

Character arrays or strings

```
char color[] = "blue";
```

This definition creates a 5-element array color containing the characters 'b', 'l', 'u', 'e' and '\0'

Try out this codes

```
#include <stdio.h>
int main( void )
{
	char color[]="blue";
	printf("%s\n",color);
	printf("%d",sizeof(color));
	return 0;
}
```

```
#include <stdio.h>
int main( void )
{
	char color[20];
	printf("Enter the color\n");
	scanf("%s",color);
	printf("The car is %s",color);
	return 0;
}
```

Pointers

- Pointers store memory addresses.
- Normally, a variable directly contains a specific value. E.g., int x=5;
- A pointer, contains an address of a variable that contains a specific value. E.g., address of int x;

Pointer declarations:

```
int x;
int *xPtr;
```

*xPtr is a pointer to an integer x. Therefore it must be type integer

When (*) is used in this manner in a definition, it indicates that the variable being defined is a pointer.

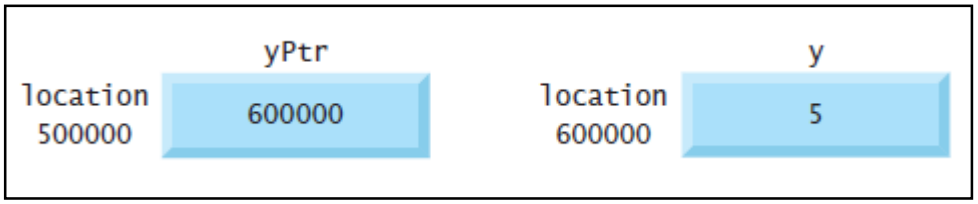
Pointers

```
int y = 5;
int *yPtr;
yPtr = &y;
```

Variable and pointer declarations

Assigning the address of the y to pointer yPtr.

& :address operator



It shows the representation of the pointer in memory, assuming that integer variable y is stored at location 600000, and pointer variable yPtr is stored at location 500000.

yPtr stores the location (address) of y.

Pointer example

Try out this codes

```
#include <stdio.h>
int main( void )
{
        int a;
        int *aPtr; /* aPtr is a pointer to an integer */

        a = 7;
        aPtr = &a;

        printf( "The address of a is %p \nThe value of aPtr is %p", &a, aPtr);

        printf( "The value of a is %d \nThe value of *aPtr is %d", a, *aPtr );

        return 0;
}
```

```
#include <stdio.h>
int main( void )
{
int a;
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a;

printf( "The value of a+a is %d\n", a+a);
printf( "The value of *aPtr+*aPtr is %d\n", *aPtr+*aPtr);

return 0;
}
```

%p → Displays in hex format

Relationship between Pointers and Arrays

```
#include <stdio.h>


int main( void )
{
        int b[]= {1,2,3,4,5};
        int *bPtr=b;

        printf( " %p\n", &b );
        printf( " %p\n\n", bPtr );

        printf( " %d\n", b[3] );
        printf( " %d\n", *(bPtr+3) );


        return 0;
}
```

bPtr equals to the address of the first element in array b. This statement is equal to bPtr = &b[0];

The 3 in the expression is the offset to the pointer

Output:

```
0x7ffc9101f610
0x7ffc9101f610

4
4
```

Functions with Pointer variables - Passing Arguments to Functions by Reference

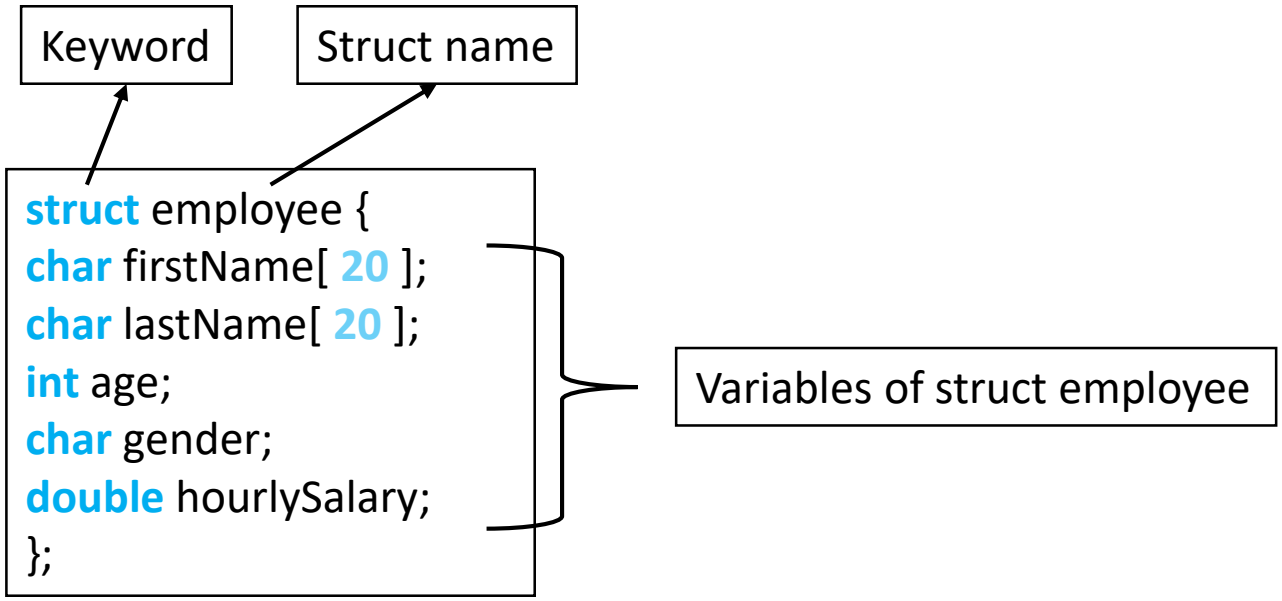
```
#include <stdio.h>

void cubeByReference( int *nPtr ); /* prototype */

int main( void )

{
        int number = 5;
        printf( "The original value of number is %d", number );
        cubeByReference( &number );
        printf( "\nThe new value of number is %d\n", number );
        return 0;
}

void cubeByReference( int *nPtr )
{
        *nPtr = *nPtr * *nPtr * *nPtr;
}
```

A function with pointer argument:
It must receive an address as an argument

Address of the variable is passed to function

Structures

- Structures collections of related variables under one name
- Structures may contain variables of many different data types



Defining Variables of Structure Types

```
int x;
int y;
```

These declerations generate integer variables x and y

- Similarly, we can generate new variables of structure types

```
struct employee employee1;
struct employee employee2;
```

We have two structure variables: employee1 and employee2.

```
struct employee1 {
char firstName[ 20 ];
char lastName[ 20 ];
int age;
char gender;
double hourlySalary;
};
```

```
struct employee2 {
char firstName[ 20 ];
char lastName[ 20 ];
int age;
char gender;
double hourlySalary;
};
```

Struct variables can also be defined in this way:

```
struct example {
char c;
int i;
} sample1, sample2;
```

How to access structure members

- Two operators are used to access members of structures:
- (.) the dot operator,
- (->) the arrow operator.

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

struct employee {
char name[ 20 ];
int age;
}employee1, employee2;

int main( void ){

strcpy(employee1.name,"xxx yyy");
strcpy(employee2.name,"zzz www");

employee1.age=33;
employee2.age=28;

printf("First employee is %s and age %d\n",employee1.name,employee1.age);

printf("First employee is %s and age %d\n",employee2.name,employee2.age);

return 0;
}
```

```
#include <stdio.h>

struct employee {
char *name;
int age;
}employee1, employee2;

int main( void ){

char temp1[]="xxx yyy"; employee1.name=temp1;

char temp2[]="zzz www"; employee2.name=temp2;

employee1.age=33;
employee2.age=28;

printf("First employee is %s and age
%d\n",employee1.name,employee1.age);

printf("First employee is %s and age
%d\n",employee2.name,employee2.age);

return 0;
}
```

Outputs:

```
First employee is xxx yyy and age 33
First employee is zzz www and age 28
```

Structure inside another structure - Nested structures

```
struct B
{
   int number;
};
```

```
struct A
{
   int data;
   struct B myB;
};
```

Struct B is nested in struct A

```
#include <stdio.h>
struct B
{
   int number;
};
struct A
{
   int data;
   struct B myB;
};
int main ()
{
struct A myA;
myA.myB.number = 42;
printf(" %d\n",myA.myB.number);
return 0;
}
```

A type struct variable decleration

Accessing the inner structure

typedef keyword

- The keyword typedef provides a mechanism for creating synonyms for previously defined data types.
- Names for structure types are often defined with typedef to create shorter type names.

```
typedef struct {
int x;
int y;
} student;
```

(student) is the new variable name of this struct variable.

```
student student1;
```

(student1) variable is in student datatype.

```
struct employee employee1;
struct employee employee2;
```

This is how we generate struct variables previously

typedef keyword

- Typedef is commonly used struct variables

```
#include <stdio.h>

typedef struct  {
char *name;
int age;
}employee;

employee employee1, employee2;
int main( void ){

char temp1[]="xxx yyy";
employee1.name=temp1;

char temp2[]="zzz www";
employee2.name=temp2;

employee1.age=33;
employee2.age=28;

printf("First employee is %s and age
%d\n",employee1.name,employee1.age);

printf("First employee is %s and age
%d\n",employee2.name,employee2.age);

return 0;
}
```

```
#include <stdio.h>
typedef struct
{
   int number;
}B;
typedef struct
{
   int data;
   B myB;
}A;
int main ()
{
A myA;
myA.myB.number = 42;
printf(" %d\n",myA.myB.number);
return 0;
}
```

Previous examples with typedef

Bitwise Operations

- Computers represent all data internally as sequences of bits.
- Each bit can assume the value 0 or the value 1.
- The bitwise operators are used to manipulate the bits.

The bitwise operators are:

- bitwise AND (&)
- bitwise inclusive OR (|)
- bitwise exclusive OR (^)
- left shift (<<)
- right shift (>>)
- complement (~)

Bitwise Operations

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	one's complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise operations

Example #1: Bitwise AND

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

Example #2: Bitwise OR

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Bitwise operations

Example #3: Bitwise XOR

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

Example #5: Shift Operators

```c
#include <stdio.h>
int main()
{
    int num=212, i;
    for (i=0; i<=2; ++i)
        printf("Right shift by %d: %d\n", i, num>>i);

    printf("\n");

    for (i=0; i<=2; ++i)
        printf("Left shift by %d: %d\n", i, num<<i);

    return 0;
}
```

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary)
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary)
212<<0 = 11010100 (Shift by 0)
212<<4 = 110101000000 (In binar
```

Bitwise assignment operators

Bitwise assignment operators	
&=	Bitwise AND assignment operator.
|=	Bitwise inclusive OR assignment operator.
^=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Right-shift assignment operator.

a&=b	Bitwise AND a with b, assign result to a.
a|=b	Bitwise OR a with b, assign result to a.
a^=b	Bitwise XOR a with b, assign result to a.
a<<=b	Left shift a b bits, assign result to a.
a>>=b	Right shift a b bits, assign result to a.

Enumeration Constants

- An enumeration, introduced by the keyword enum, is a set of integer enumeration constants represented by identifiers.

- Values in an enum start with 0, and are incremented by 1.

```
#include <stdio.h>

enum months {
JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,
OCT, NOV, DEC
};

int main ()
{
        enum months month;
        month = JUL;
        printf("%dth month",month);

        return 0;
}
```

creates a new type, enum months, in which the identifiers are set to the integers 0 to 11,respectively.

Month variable decleration in the type of months. Month variable can contain any of the months.

JUL is taken as integer 6.

Outputs:

```
6th month
```