**NOTE:**

**1.All class lectures are included in the exam.(quiz)**

**2.All email attachments are also included.**

**3.class, modules, behavior,constructors, overloaded constructors, real world examples regarding oops discussed in class lectures.**

**4.arrays in c# (reading assignment) important .**

**CONSTANTS:**

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.

The constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and no prefix id required for decimal.

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

Here are some examples of integer literals:

212 /\* Legal \*/

215u /\* Legal \*/

0xFeeL /\* Legal \*/

078 /\* Illegal: 8 is not an octal digit \*/

032UU /\* Illegal: cannot repeat a suffix \*/

Following are other examples of various types of Integer literals:

85 /\* decimal \*/

0213 /\* octal \*/

0x4b /\* hexadecimal \*/

30 /\* int \*/

30u /\* unsigned int \*/

30l /\* long \*/

30ul /\* unsigned long \*/

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

Here are some examples of floating-point literals:

3.14159 /\* Legal \*/

314159E-5L /\* Legal \*/

510E /\* Illegal: incomplete exponent \*/

210f /\* Illegal: no decimal or exponent \*/

.e55 /\* Illegal: missing integer or fraction \*/

While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Character Constants

Character literals are enclosed in single quotes, e.g., 'x' and can be stored in a simple variable of char type. A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C# when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Here, you have a list of some of such escape sequence codes:

|  |  |
| --- | --- |
| **Escape sequence** | **Meaning** |
| \\ | \ character |
| \' | ' character |
| \" | " character |
| \? | ? character |
| \a | Alert or bell |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \ooo | Octal number of one to three digits |
| \xhh . . . | Hexadecimal number of one or more digits |

Following is the example to show few escape sequence characters:

namespace EscapeChar

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello\tWorld\n\n");

Console.ReadLine();

}

}

}

When the above code is compiled and executed, it produces the following result:

Hello World

String Literals

String literals or constants are enclosed in double quotes "" or with @"". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating the parts using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

@"hello dear"

Defining Constants

Constants are defined using the **const** keyword. Syntax for defining a constant is:

const <data\_type> <constant\_name> = value;

using System;

namespace DeclaringConstants

{

class Program

{

static void Main(string[] args)

{

const double pi = 3.14159; // constant declaration

double r;

Console.WriteLine("Enter Radius: ");

r = Convert.ToDouble(Console.ReadLine());

double areaCircle = pi \* r \* r;

Console.WriteLine("Radius: {0}, Area: {1}", r, areaCircle);

Console.ReadLine();

}

}

}

Types of loops in c#:

WHILE LOOP:

A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.

Syntax:

The syntax of a **while** loop in C# is:

while(condition)

{

statement(s);

}

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram:



Here, key point of the *while* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

/\* local variable definition \*/

int a = 10;

/\* while loop execution \*/

while (a < 20)

{

Console.WriteLine("value of a: {0}", a);

a++;

}

Console.ReadLine();

}

}

}

FOR LOOP:

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

## Syntax:

The syntax of a **for** loop in C# is:

for ( init; condition; increment )

{

statement(s);

}

Flow Diagram:



Example:

using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

/\* for loop execution \*/

for (int a = 10; a < 20; a = a + 1)

{

Console.WriteLine("value of a: {0}", a);

}

Console.ReadLine();

}

}

}

DO-WHILE LOOP:

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

## Syntax:

The syntax of a **do...while** loop in C# is:

do

{

statement(s);

}while( condition );

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

## Flow Diagram:

## Example:

using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

/\* local variable definition \*/

int a = 10;

/\* do loop execution \*/

do

{

Console.WriteLine("value of a: {0}", a);

a = a + 1;

} while (a < 20);

Console.ReadLine();

}

}

}

NESTED LOOP:

C# allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

## Syntax:

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 while loop** statement in C# is as follows:

while(condition)

{

while(condition)

{

statement(s);

}

statement(s);

}

The syntax for a **nested do...while loop** statement in C# is as follows:

do

{

statement(s);

do

{

statement(s);

}while( condition );

}while( condition );

FOR EACH LOOP:

**Foreach** is a loop construct. It does not use an integer index. Instead, it is used on a collection and returns each element in order. This is called enumeration. It eliminates errors caused by incorrect index handling.

**Example**

STRING ARRAY:

We first use the keyword foreach on a string array to loop through the elements in the array. In the foreach-statement, you do not need to specify the loop bounds minimum or maximum. You do not need an "i" variable as in for-loops.

 the foreach-statement contains the reserved "foreach" keyword. The iteration variable "string value" can be a different type such as "int number" if you are looping over that type.

using System;

namespace FOREACH\_LOOP

{

class Program

{

static void Main(string[] args)

{

string[] ferns =

{

"saad",

"salman",

"sarah",

"seema"

};

// Loop with the foreach keyword.

foreach (string value in ferns)

{

Console.WriteLine(value);

}

Console.ReadKey();

}

}

}