Image Processing

Lecture 5

Introducing Image Processing

Continue

Image Enhancement in the Spatial Domain

Sharpening Spatial Filters

- **Smoothing Linear Filters :** is used to remove small details in images (noise reduction) or to blur images.

$\frac{1}{9} \times$

1	1	1
1	1	1
1	1	1

$\frac{1}{16} \times$

1	2	1
2	4	2
1	2	1

$$g(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)}$$

$$R = \frac{1}{9} \sum_{i=1}^{9} z_i,$$

Variables Declaration

- int age;
  age = 25 ;
- int age = 25;
- int xSize, ySize;
- int xSize = 4, ySize = 5;
- int xSize, ySize = 5;
  xSize = 4;
- float x=1.234f;
- double pi=22.0/7.0;

Mathematical Operation

- Mathematical Operators :

  Unary (++i) , Binary (x+y), Ternary)

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
+	Binary	`var1 = var2 + var3;`	`var1` is assigned the value that is the sum of `var2` and `var3`.
-	Binary	`var1 = var2 - var3;`	`var1` is assigned the value that is the value of `var3` subtracted from the value of `var2`.
*	Binary	`var1 = var2 * var3;`	`var1` is assigned the value that is the product of `var2` and `var3`.
/	Binary	`var1 = var2 / var3;`	`var1` is assigned the value that is the result of dividing `var2` by `var3`.
%	Binary	`var1 = var2 % var3;`	`var1` is assigned the value that is the remainder when `var2` is divided by `var3`.
+	Unary	`var1 = +var2;`	`var1` is assigned the value of `var2`.
-	Unary	`var1 = -var2;`	`var1` is assigned the value of `var2` multiplied by `-1`.

Mathematical Operation

- Increment/Decrement Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
++	Unary	`var1 = ++var2;`	`var1` is assigned the value of `var2 + 1`. `var2` is incremented by 1.
--	Unary	`var1 = --var2;`	`var1` is assigned the value of `var2 - 1`. `var2` is decremented by 1.
++	Unary	`var1 = var2++;`	`var1` is assigned the value of `var2`. `var2` is incremented by 1.
--	Unary	`var1 = var2--;`	`var1` is assigned the value of `var2`. `var2` is decremented by 1.

Mathematical Operation

- Assignment Mathematical Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
=	Binary	`var1 = var2;`	`var1` is assigned the value of `var2`.
+=	Binary	`var1 += var2;`	`var1` is assigned the value that is the sum of `var1` and `var2`.
-=	Binary	`var1 -= var2;`	`var1` is assigned the value that is the value of `var2` subtracted from the value of `var1`.
*=	Binary	`var1 *= var2;`	`var1` is assigned the value that is the product of `var1` and `var2`.
/=	Binary	`var1 /= var2;`	`var1` is assigned the value that is the result of dividing `var1` by `var2`.
%=	Binary	`var1 %= var2;`	`var1` is assigned the value that is the remainder when `var1` is divided by `var2`.

- Look : x= ++a * b--;

Operator Precedence (Priority)

PRECEDENCE	OPERATORS
Highest	++, -- (used as prefixes); +, - (unary)
	*, /, %
	+, -
	=, *=, /=, %=, +=, -=
Lowest	++, -- (used as suffixes)

- If a=10 , b=20
- x = ++a * b++ → x = 220

Logic Operation (comparison Operators)

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
==	Binary	`var1 = var2 == var3;`	`var1` is assigned the value `true` if `var2` is equal to `var3`, or `false` otherwise.
!=	Binary	`var1 = var2 != var3;`	`var1` is assigned the value `true` if `var2` is not equal to `var3`, or `false` otherwise.
<	Binary	`var1 = var2 < var3;`	`var1` is assigned the value `true` if `var2` is less than `var3`, or `false` otherwise.
>	Binary	`var1 = var2 > var3;`	`var1` is assigned the value `true` if `var2` is greater than `var3`, or `false` otherwise.
<=	Binary	`var1 = var2 <= var3;`	`var1` is assigned the value `true` if `var2` is less than or equal to `var3`, or `false` otherwise.
>=	Binary	`var1 = var2 >= var3;`	`var1` is assigned the value `true` if `var2` is greater than or equal to `var3`, or `false` otherwise.

- Relational Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
&&	Binary	`var1 = var2 && var3;`	`var1` is assigned the value `true` if `var2` and `var3` are both `true`, or `false` otherwise. (Logical `AND`)
||	Binary	`var1 = var2 || var3;`	`var1` is assigned the value `true` if either `var2` or `var3` (or both) is `true`, or `false` otherwise. (Logical `OR`)

Bitwise Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
`!`	Unary	`var1 = !var2;`	`var1` is assigned the value `true` if `var2` is `false`, or `false` if `var2` is `true`. (Logical `NOT`)
`&`	Binary	`var1 = var2 & var3;`	`var1` is assigned the value `true` if `var2` and `var3` are both `true`, or `false` otherwise. (Logical `AND`)
`|`	Binary	`var1 = var2 | var3;`	`var1` is assigned the value `true` if either `var2` or `var3` (or both) is `true`, or `false` otherwise. (Logical `OR`)
`^`	Binary	`var1 = var2 ^ var3;`	`var1` is assigned the value `true` if either `var2` or `var3`, but not both, is `true`, or `false` otherwise. (Logical `XOR` or exclusive `OR`)

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
`&=`	Binary	`var1 &= var2;`	`var1` is assigned the value that is the result of `var1 & var2`.
`|=`	Binary	`var1 |= var2;`	`var1` is assigned the value that is the result of `var1 | var2`.
`^=`	Binary	`var1 ^= var2;`	`var1` is assigned the value that is the result of `var1 ^ var2`.

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
`>>`	Binary	`var1 = var2 >> var3;`	`var1` is assigned the value obtained when the binary content of `var2` is shifted `var3` bits to the right.
`<<`	Binary	`var1 = var2 << var3;`	`var1` is assigned the value obtained when the binary content of `var2` is shifted `var3` bits to the left.

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
`>>=`	Unary	`var1 >>= var2;`	`var1` is assigned the value obtained when the binary content of `var1` is shifted `var2` bits to the right.
`<<=`	Unary	`var1 <<= var2;`	`var1` is assigned the value obtained when the binary content of `var1` is shifted `var2` bits to the left.

Examples

```
   1  0  0        4
&  1  0  1     &  5
-------------  ----
   1  0  0        4
```

The same process occurs if you use the | operator, except that in this case each result bit is 1 if either of the operand bits in the same position is 1, as shown in the following equations:

```
   1  0  0        4
|  1  0  1     |  5
-------------  ----
   1  0  1        5
```

Priority - Again

PRECEDENCE	OPERATORS
Highest	++, -- (used as prefixes); (), +, - (unary), !, ~
	*, /, %
	+, -
	<<, >>
	<, >, <=, >=
	==, !=
	&
	^
	|
	&&
	||
	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=
Lowest	++, -- (used as suffixes)

Notes

Conversion

- Implicitly Conversion : byte x=5; short y; y=x;
- Explicitly Conversion : short b=2345;

  a = (byte) b;

  you can use checked : a=checked((byte) b);
- Convert Using Convert class
- Convert using class : int.Parse() ; int.TryParse()
- Convert Using Encoding & encoder:

  1. *string str="asddff"; byte[ ]b=new byte[str.length];*
     b = Encoding.ASCII.GetBytes();
  2. *str = Encoding.UTF8.GetString();*

Implicit

TYPE	CAN SAFELY BE CONVERTED TO
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

Convert Class

COMMAND	RESULT
`Convert.ToBoolean(val)`	`val` converted to `bool`
`Convert.ToByte(val)`	`val` converted to `byte`
`Convert.ToChar(val)`	`val` converted to `char`
`Convert.ToDecimal(val)`	`val` converted to `decimal`
`Convert.ToDouble(val)`	`val` converted to `double`
`Convert.ToInt16(val)`	`val` converted to `short`
`Convert.ToInt32(val)`	`val` converted to `int`
`Convert.ToInt64(val)`	`val` converted to `long`
`Convert.ToSByte(val)`	`val` converted to `sbyte`
`Convert.ToSingle(val)`	`val` converted to `float`
`Convert.ToString(val)`	`val` converted to `string`
`Convert.ToUInt16(val)`	`val` converted to `ushort`
`Convert.ToUInt32(val)`	`val` converted to `uint`
`Convert.ToUInt64(val)`	`val` converted to `ulong`

Arrays

- int[] myIntArray = { 5, 9, 10, 2, 99 };
- int[] myIntArray = new int[5];
- int[] myIntArray = new int[arraySize];
- int[] myIntArray = new int[5] { 5, 9, 10, 2, 99 };
- int[] myIntArray = new int[10] { 5, 9, 10, 2, 99 };
- const int arraySize = 5;
  int[] myIntArray = new int[arraySize] { 5, 9, 10, 2, 9 };
- int[] myIntArray;
- myIntArray = new int[5];
- string[] friendNames = { "Robert Barwell", "Mike Parry",
  "Jeremy Beacock" };

Examples

- for(int i=0; i <marray.length; ++i)
  Console.Write(marray[i]);

- foreach(int x in marray)
  Console.Write(x);

- foreach(string str in strarray)
  Console.Write(str);

Multidimensional Arrays

- double[,] hillHeight = new double[3,4];
- double[,] hillHeight = { { 1, 2, 3, 4 },
  { 2, 3, 4, 5 },
  { 3, 4, 5, 6 } };
- hillHeight[2,1]
- Example :

```
foreach (double height in hillHeight)
{
   Console.WriteLine("{0}", height);
}
```

Arrays of Arrays

- int[][] jaggedIntArray;
  jaggedIntArray = new int[3][4];
  jaggedIntArray = { { 1, 2, 3 }, { 1 }, { 1, 2 } };
- int[][] jaggedIntArray;
  jaggedIntArray = new int[2][];
  jaggedIntArray[0] = new int[3];
  jaggedIntArray[1] = new int[4];
- int[][] jaggedIntArray;
  jaggedIntArray = new int[3][] { new int[] { 1, 2, 3 },
  new int[] { 1 },
  new int[] { 1, 2 } };
- int[][] jaggedIntArray = { new int[] { 1, 2, 3 },
  new int[] { 1 },
  new int[] { 1, 2 } };

Examples

- ```
  foreach (int divisor in divisors1To10)
  {
     Console.WriteLine(divisor);
  }
  ```

- ```
  foreach (int[] divisorsOfInt in divisors1To10)
  {
     foreach(int divisor in divisorsOfInt)
     {
        Console.WriteLine(divisor);
     }
  }
  ```

```
int[][] divisors1To10 = {
new int[] { 1 },
new int[] { 1, 2 },
new int[] { 1, 3 },
new int[] { 1, 2, 4 },
new int[] { 1, 5 },
new int[] { 1, 2, 3, 6 },
new int[] { 1, 7 },
new int[] { 1, 2, 4, 8 },
new int[] { 1, 3, 9 },
new int[] { 1, 2, 5, 10 } };
```

String

H.W