Char and Int Conversion, Ordering of Characters

https://csci-1301.github.io/about#authors

June 30, 2021 (05:30:51 PM)

Contents

0.1 Reading and Understanding

Characters are represented by integers: cf. https://en.wikipedia.org/wiki/ASCII#Printable_characters for a mapping between the glyphs (i.e., space, `A`, `!`, etc.) and **dec**imal values, to be read as "integer code", i.e., 32, 33, 34, etc.

In the referenced table, each character's integer code is given for different numeral systems[1]:

- Binary: base 2
- Oct: octal, base 8
- Dec: decimal, base 10
- Hex: hexadecimal, base 16

Decimal system is what we use everyday, but computer programs occasionally use other numerical systems.

Note that the characters are divided in groups, and that there are 95 printable characters.

0.2 Converting

Copy the following snippet of code in a `Main` method:

```
int intVar = (int)'C';
char charVar = (char)84;
Console.WriteLine($"'C' is represented as {intVar}");
Console.WriteLine($"{charVar} corresponds to the value 84");
```

And note that we can explicitly convert `int` into `char`, and `char` into `int`.

Actually, the conversion from `char` to `int` could be done implicitly by C#: replace the previous first line with

[1] https://en.wikipedia.org/wiki/Radix#In_numeral_systems

```
int intVar = 'C';
```

And note that your program still compiles.

Can you also convert implicitly `int` into `char`?

Next write code to determine the `int` values for the following characters:

`char` value	`int` value
`w`	119
`A`	
`5`	
`#`	

Also determine what characters the following integers (in decimal system) represent:

`int` value	`char` value
49	
104	
89	

0.3 Comparing

Exactly as 65 is less than 97, the character associated with 65, `A`, is less than the character associated with 97, `a`.

You can convince yourself by executing the following code:

```
if ('A' > 'a')
    Console.Write("A is greater than a");
else
    Console.Write("A is less than a");
```

Implement the following short program to practice this concept:

1. Ask user to enter a lowercase character.
2. Check that the character is within a - z range, to check that it *is* a lowercase character
3. when it is not, display "not a lowercase character"
4. otherwise perform the following steps:
    - if user enters character `'n'`, display "You entered n"
    - if the character occurs before `'n'`, display "Before n"
    - if the character occurs after `'n'`, display "After n"

- To read *a single character* (instead of a whole string), use ReadKey() method: `Console.ReadKey().KeyChar`

0.4 Testing for Equality

You can also test if a character is equal to an other by using ==, as for integer values. This is particularly useful when we want to ask the user for a "yes" / "no" decision.

Write a program that

- Asks the user for a character,

- Displays on the screen “The user said yes” if the user entered `'Y'` or `'y'`,
- Displays on the screen “The user said no” if the user entered `'N'` or `'n'`,
- Displays on the screen “The user entered an incorrect value” if the user entered any other character.

1 Pushing Further (Optional)

1.1 String Comparison

Comparing strings cannot be done with > and < operators. To compare them, we have to use the CompareOrdinal[2] method of the String[3] class.

It works as follow:

```
if (String.CompareOrdinal("A", "a") > 0)
{
    Console.Write("A is greater than a");
}
else
{
    Console.Write("A is less than a");
}
```

Note that `CompareOrdinal` returns an integer, that we then compare with 0.

- If the value returned is 0, then the strings are the same,
- If the value returned is less than 0, then the first string is less than the second one,
- If the value returned is greater than 0, then the first string is greater than the second one.

In the previous example, we tested string made of only one character, but we can compare arbitrarily complex strings:

```
if (String.CompareOrdinal("Augusta", "August") > 0)
{
    Console.Write("Augusta is greater than August");
}
else
{
    Console.Write("Augusta is less than August");
}
```

To conclude with this topic, note that the integer returned actually has a precise value.

Examine the following code to understand it.

```
if (String.CompareOrdinal("A", "a") == ((int)'A' - (int)'a'))
    Console.WriteLine("Ok, I get it now");

if (String.CompareOrdinal("Ab", "az") == (((int)'A' + (int)'b') - ((int)'a' + (int)'z')))
    Console.WriteLine("Yes, I really do.");

else if (String.CompareOrdinal("Ab", "az") == ((int)'A' - (int)'a'))
    Console.WriteLine("Or do I?");
```

[2] https://docs.microsoft.com/en-us/dotnet/api/system.string.compareordinal
[3] https://docs.microsoft.com/en-us/dotnet/api/system.string

```
if (String.CompareOrdinal("ABCDEf", "ABCDEF") == (int)'f' - (int)'F')
    Console.WriteLine("Ok, now I'm good.");
```

Do you understand how the returning value is computed for these strings?