PCP — Lecture 11
Fall 2020 October 20, 2020

1 Increment and Decrement Operators

Increment and decrement operators are used to add or remove one from variables holding numerical values.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Increment</th>
<th>Decrement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postfix (after)</td>
<td>a++</td>
<td>a-</td>
</tr>
<tr>
<td>Infix (before)</td>
<td>++a</td>
<td>-a</td>
</tr>
</tbody>
</table>

Postfix operators are always executed before any other operators (cf. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#operator-precedence-and-associativity), but the way those four operators interact with other assignments is subtle. Generally, postfix operators are executed after the rest of the statement is executed, while prefix operators are executed before the rest of the statement:

```csharp
int a = 0;
Console.WriteLine(a--); // The value of a (0) is displayed, then decremented (-1).
Console.WriteLine(a);  // The value of a is (-1) displayed
Console.WriteLine(--a); // The value of a (-1) is decremented (-2) then displayed.
int b = a++;           // The value of a (-2) is assigned to b then incremented (-1).
```

There is an additional example at the beginning of today’s lab.

2 Definition and First Example of while loops

2.1 Formal Syntax

```csharp
while (<condition>)
{
    <statement block>
}
```

2.2 Example

```csharp
int number = 0;
while (number <= 5)
{
    C.WL("Hi Mom!");
    C.WL(number);
    number++;
}
```
Notes:

- If `<condition>` is false: 0 execution of the body.
- If `<condition>` is always true: program loop for ever!
- The conditions under which `<condition>` changes should be given a chance to change in the body of the loop!

3 Five Ways Things Can Go Wrong

It is easy to write wrong loop statements. Let us review some of the “classic” blunders.

3.1 Failing to update the variable occurring in the condition

```c
int number = 0;
while (number <= 5) {
    C.WL("Hi Mom!");
    C.WL(number);
}
```

Number isn’t changed!

3.2 Updating the “wrong” value

```c
int number1, number = 0;
while (number <= 5) {
    C.WL("Hi Mom!");
    C.WL(number);
    number1++;
}
```

3.3 Having an empty body

```c
int number = 0;
while (number <= 5);  // Note the semi-colon here!
{
    C.WL("Hi Mom!");
    C.WL(number);
    number++;
}
```
3.4 Having an empty body (variation)

```c
int number = 0;
while (number <= 5)
    C.WL("Hi Mom!");
    C.WL(number);
    number++;
```

3.5 Going in the wrong direction

```c
int number = 0;
while (number >= 5)
{
    C.WL("Hi Mom!");
    C.WL(number);
    number++;
}
```

The variable `number` should be decremented, not incremented.

4 User-Input Validation

We can use loops to test what was entered by the user, and ask again if the value does not fit our needs:

```c
Console.WriteLine("Please enter a positive number");
int n = int.Parse(Console.ReadLine());
while (n < 0)
{
    Console.WriteLine("You entered {n}, I asked you for a positive number. Please try again.");
    n = int.Parse(Console.ReadLine());
}
```

5 Vocabulary

Variables and values can have multiple roles, but it is useful to mention three different roles in the context of loops:

- **Counter** Variable that is incremented every time a given event occurs.

```c
int i = 0; // i is a counter
while (i < 10){
    Console.WriteLine("{i}");
    i++;
}
```

- **Sentinel Value** A special value that signals that the loop needs to end.
```csharp
Console.WriteLine("Give me a string.");
string ans = Console.ReadLine();
while (ans != "Quit") // The sentinel value is "Quit".
{
    Console.WriteLine("Hi!");
    Console.WriteLine("Enter "Quit" to quit, or anything else to continue.");
    ans = Console.ReadLine();
}

Accumulator Variable used to keep the total of several values.

int i = 0, total = 0;
while (i < 10){
    total += i; // total is the accumulator.
    i++;
}

Console.WriteLine($"The sum from 0 to {i} is {total}.");

We can have an accumulator and a sentinel value at the same time:

Console.WriteLine("Enter a number to sum, or "Done" to stop and print the total.");
string enter = Console.ReadLine();
int sum = 0;
while (enter != "Done")
{
    sum += int.Parse(enter);
    Console.WriteLine("Enter a number to sum, or "Done" to stop and print the total.");
    enter = Console.ReadLine();
}

Console.WriteLine($"Your total is {sum}.");

You can have counter, accumulator and sentinel values at the same time!

int a = 0;
int sum = 0;
int counter = 0;
Console.WriteLine("Enter an integer, or N to quit.");
string entered = Console.ReadLine();
while (entered != "N") // Sentinel value
{
    a = int.Parse(entered);
    sum += a; // Accumulator
    Console.WriteLine("Enter an integer, or N to quit.");
    entered = Console.ReadLine();
    counter++; // counter
}

Console.WriteLine($"The average is {sum / (double)counter}.");

We can distinguish between three “flavors” of loops (that are not mutually exclusive):

**Sentinel controlled loop** The exit condition test if a variable has (or is different from) a specific value.
User controlled loop  The number of iteration depends on the user.

Count controlled loop  The number of iteration depends on a counter.

Note that a user-controlled loop can be sentinel-controlled (that is the example we just saw), but also count-controlled (“Give me a value, and I will iterate a task that many times”).

6 More Input Validation, Using TryParse

The TryParse method is a complex method that will allow us to parse strings, and to “extract” a number out of them if they contain one, or to be given a way to recover if they don’t.

```csharp
Console.WriteLine("Please, enter an integer.");
string message = Console.ReadLine();
int a;
bool res = int.TryParse(message, out a);
if (res)
{
    Console.WriteLine($"The value entered was an integer: \{a\}.";
}
else
{
    Console.WriteLine("The value entered was not an integer, so 0 is assigned to \a.\";
}
Console.WriteLine(a);
```

As you can see, int.TryParse takes two arguments, a string and a variable name (prefixed by the “magic” novel keyword out) and returns a boolean. You will get a chance to experiment with this code in lab.