

PCP — Lecture 09

Fall 2020 October 3, 2020

Last Time - Wrapping up classes

Please, refer to the [second milestone](#) for a brief recap of what we've done recently.

1 Decision (or Control) Structures

- Until now, we only wrote programs with a sequential flow of execution (line n was always executed after line $n-1$ and before line $n+1$).
- But we can control the flow using conditions, to perform selection (“execute this part of the code if this, that part if that”) and iteration (“repeat this part of the code until something happens”).
- The novel keyword we will introduce correspond to decision structures, and are `if`, `else`, `switch`, `while`, `do...while` and `for`.
- Decision structures always have to know if something is the case, a notion expressed in programs thanks to the notion of conditions, and captured by the boolean datatype.

2 Boolean and Conditions

A condition is either true or false. We can store if something is true or false in a (boolean) *flag*, which is simply a variable of type boolean.

We can declare, assign, initialize and display it as any other variable:

```
1 bool flag = true;
2 Console.WriteLine(true);
```

But the only two possible values are `true` and `false`, and we will study three operations on them: “and” (`&&`, the conjunction), “or” (`||`, the disjunction) and “not” (`!`, the negation). They have the expected meaning that the condition “A and B” is true if and only if A is true, and B is true. Similarly, “A or B” is false if and only if A is false, and B is false (that is, it takes only one to make their disjunction true).

We present this behavior with *truth tables*, as follows:

| | | |
|--------------------|-------------------------|--------------------|
| <hr/> | | |
| <code>true</code> | <code>&&</code> | <code>true</code> |
| <code>true</code> | <code>&&</code> | <code>false</code> |
| <code>false</code> | <code>&&</code> | <code>true</code> |
| <code>false</code> | <code>&&</code> | <code>false</code> |
| <hr/> | | |

| | | |
|--------------------|-----------------|--------------------|
| <hr/> | | |
| <code>true</code> | <code> </code> | <code>true</code> |
| <code>true</code> | <code> </code> | <code>false</code> |
| <code>false</code> | <code> </code> | <code>true</code> |
| <code>false</code> | <code> </code> | <code>false</code> |
| <hr/> | | |

| | |
|--------|-------|
| !true | false |
| !false | true |

We could also have represented those tables in 2-dimensions, and will do so in lab.

3 Equality and Relational Operators

| Equality Operators | | |
|-----------------------|-------------|----------------|
| Mathematical Notation | C# Notation | Example |
| = | == | 3 == 4 → false |
| ≠ | != | 3 != 4 → true |

We test numerical value for equality, as well as `string`, `char` and `bool`!

```
1 Console.WriteLine(3 == 4);
2 Console.WriteLine(myStringVar == "Train");
3 Console.WriteLine(myCharVar == 'b');
```

We can also test if a value is greater than another, using the following *relational* operators.

| Relational Operators | | |
|-----------------------|-------------|----------------|
| Mathematical Notation | C# Notation | Example |
| > | > | 3 > 4 → false |
| < | < | 3 < 4 → true |
| ≥ | >= | 3 >= 4 → false |
| ≤ | <= | 3 <= 4 → true |

We can also compare `char`, but the order is a bit complex (you can find it, for instance, at <https://stackoverflow.com/a/14967721/>).

The precedence, that we will study in lab, is as follows:

! (* / %) (+ -) (< > <= >=) (== !=) && ||

4 if Statement

4.1 First Example

```
1 Console.WriteLine("Enter your age");
2 int age = int.Parse(Console.ReadLine());
3 if (age >= 18)
4 {
```

```
5     Console.WriteLine("You can vote!");  
6 }
```

The idea is that the statement `Console.WriteLine("You can vote!");` is executed only if the condition `(age >= 18)` evaluates to `true`. Otherwise, that statement is simply “skipped”.

4.2 Syntax

```
1  if (<condition>)  
2  {  
3      <statement block>  
4  }
```

Please observe the following.

- `<Condition>` is something that evaluates to a `bool`. For instance, having a number like in `if(3)` would not compile.
- Note the absence of semicolon after `if (<condition>)`.
- The curly braces can be removed if the statement block is just one statement.
- The following statements (that is, after the `}` that terminates the body of the `if` statement) are executed in any case.

5 if-else Statements

5.1 Syntax

```
1  if (<condition>)  
2  {  
3      <statement block 1>  
4  }  
5  else  
6  {  
7      <statement block 2>  
8  }
```

With `if-else` statements, the idea is that the statement block 1 is executed only if the condition evaluates to `true`, and that the statement block 2 is executed only if the condition evaluates to `false`. Note that since a condition is always either true or false, we know that at least one of the block will be executed, and since a condition cannot be true and false at the same time, at most one block will be executed: hence, exactly one block will be executed.

6 Nested if-else Statements

`<statement block>` can actually be an `if-else` statement itself!

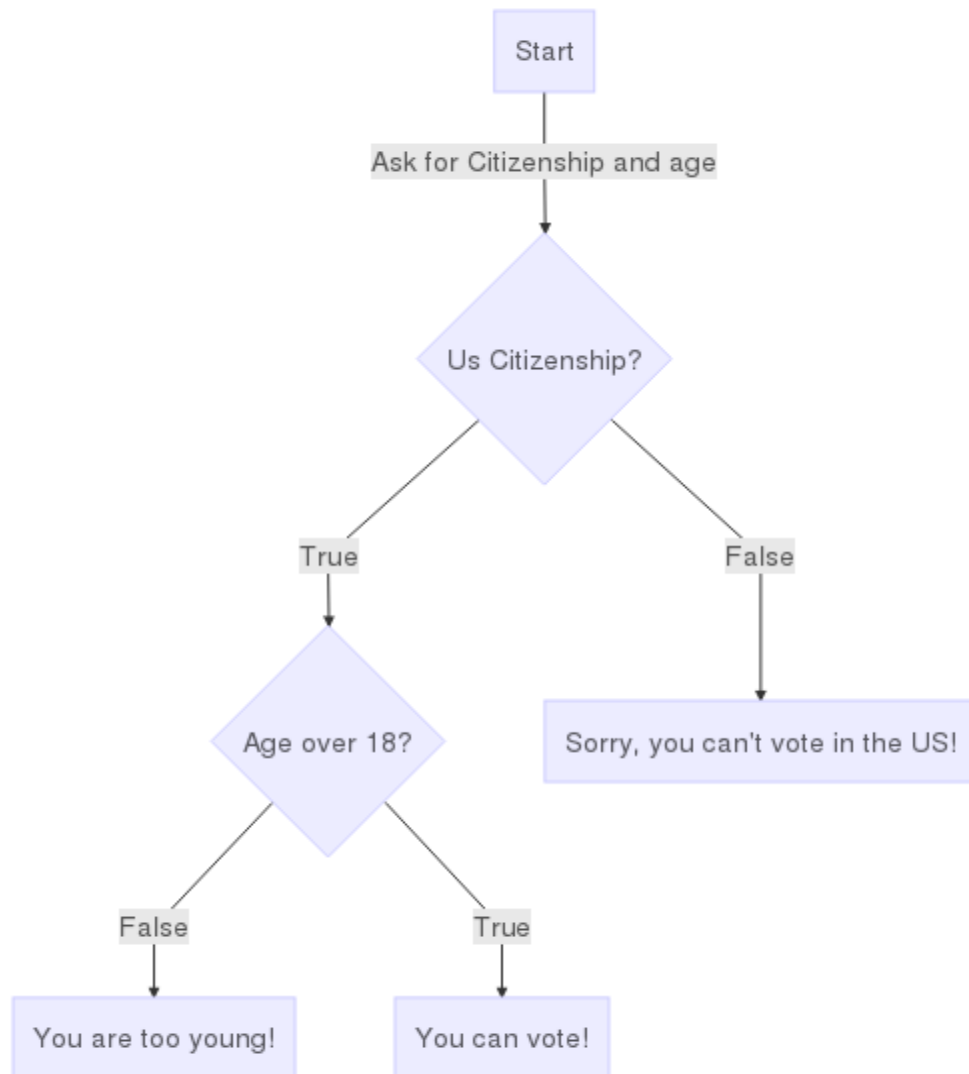


Figure 1: A flowchart representation of the nested if-else statement

```

1  bool usCitizen = true;
2  int age = 19;
3
4  if (usCitizen == true)
5  {
6      if (age > 18)
7      {
8          Console.WriteLine("You can vote!");
9      }
10     else
11     {
12         Console.WriteLine("You are too young!");
13     }
14 }
15 else
16 {
17     Console.WriteLine("Sorry, only citizens can vote");
18 }

```

Note that

- There is a simpler way to write `usCitizen == true`: simply write `usCitizen`!
- We could remove the braces
- We could have a similar flavor with only `if`: `if(age > 18 && usCitizen) ... else ...`, but the messages would be less accurate.

7 if-else-if Statements

```

1  if (<condition 1>)
2  {
3      <statement block> // Executed if condition 1 is true
4  }
5  else if (<condition 2>)
6  {
7      <statement block> // Executed if condition 1 is false and condition 2 is true
8  }
9  ...
10 else if (<condition N>)
11 {
12     <statement block> // Executed if all the previous conditions are false and condition
    ↪ N is true
13 }
14 else
15 {
16     <statement block> // Executed if all the conditions are false
17 }

```

Note that the conditions could be really different, not even testing the same thing!

7.1 Example

We can make an example with really different conditions, not overlapping:

```
1  if (age > 12)
2      x = 0;
3  else if (charVar == 'c')
4      x = 1;
5  else if (boolFlag)
6      x = 2;
7  else
8      x = 3;
```

Giving various values to age, charVar and boolFlag, we will see which value would x get in each case.

8 ?: Operator

There is an operator for if else statements for particular cases (assignment, call, increment, decrement, and new object expressions):

```
condition ? first_expression : second_expression;
```

```
1  int price = adult ? 5 : 3;
```

We will have a brief look at it if time allows, otherwise you can read about it at <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator>.