

PCP — Lecture 10

Fall 2020 October 8, 2020

Last Time - Introducing Decision Structures

- We saw that iteration and selection had in common to “break” the sequential flow of execution, and needed decisions to work.
- Decisions are implemented using boolean variables, and operators such as `&&`, `||` and `!`.
- Expression, that evaluates to boolean, can test for equality using `==`, inequality using `!=`, and for order using `>`, `<`, `>=` and `<=`.
- `if` statement allows to implement the selection procedure, so that we can “skip” over parts of code.
- `if-else` statements use that boolean are always either true or false to execute one among two blocks of code.
- `if` and `if-else` statements can be nested.

1 if-else-if Statements

```

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

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

1.1 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;

```

Try to give various values to `age`, `charVar` and `boolFlag`, and see which value would `x` get in each case.

2 Boolean Flags

Remember that a boolean *flag* is a boolean variable? We can use it to “store” the result of an interaction with a user.

Assume we want to know if the user work full time at some place, we could get started with:

```

1  Console.WriteLine("Do you work full-time here?");
2  char ch = Console.ReadKey().KeyChar; // Note that here, passing by, we are using a new
    ↪ method, to read characters.
3
4  if (ch == 'y' || ch == 'Y')
5      Console.WriteLine("Answered Yes");
6  else if (ch == 'n' || ch == 'N')
7      Console.WriteLine("Answered No");
8  else
9      Console.WriteLine("Said what?");

```

But we can’t accomodate this 3-party situation (you either work here full-time, or you don’t), so we can change the behaviour to

```

1  if (ch == 'y' || ch == 'Y')
2      Console.WriteLine("Answered Yes");
3  else
4      Console.WriteLine("Answered No");

```

We’ll study *user input validation*, that allows to get better answers from the users, later on.

But imagine we are at the beginnig of a long form, and we will need to re-use that information multiple times. With this previous command, we would need to duplicate all our code in two places. Instead, we could “save” the result of our test in a boolean variable, like so:

```

1  bool fullTime;
2  if (ch == 'y' || ch == 'Y')
3      fullTime = true;
4  else
5      fullTime = false;

```

If you looked at the `?` operator in lab, you can even shorten that statement to:

```

1  fullTime = (ch == 'y' || ch == 'Y') ? true : false;

```

Why stop here? We could even do

```
1 fullTime = (ch == 'y' || ch == 'Y');
```

Tada! We went from a long, convoluted code, to a very simple line! We already did this trick last time, but I thought that seeing it again would help.

3 Constructing a Value Progressively

In lab, last time, you were asked the following:

Ask the user for an integer, and display on the screen “positive and odd” if the number is positive and odd, “positive and even” if the number is positive and even, “negative and odd” if the number is negative and odd, “negative and even” if the number is negative and even, and “You picked 0” if the number is 0.

A possible answer is:

```
1 int a;
2 Console.WriteLine("Enter an integer");
3 a = int.Parse(Console.ReadLine());
4 if (a >= 0)
5 {
6     if (a % 2 == 0)
7         Console.WriteLine("Positive and even");
8     else // if (a % 2 != 0)
9         Console.WriteLine("Positive and odd");
10 }
11 else
12 {
13     if (a % 2 == 0)
14         Console.WriteLine("Negative and even");
15     else
16         Console.WriteLine("Negative and odd");
17 }
```

That is a lot of repetition! We could actually construct “progressively” the message we will be displaying:

```
1 string msg;
2 if (a >= 0)
3 {
4     msg = "Positive";
5 }
6 else
7 {
8     msg = "Negative";
9 }
10 if (a % 2 == 0)
11     msg += " and even";
12 else // if (a % 2 != 0)
13     msg += " and odd";
```

Much better! Since the two conditions are actually independant, we can test them in two different if statements!

4 Switch Statements

switch statements allow to simplify the “matching” of a value against a pre-determined set of values. Its formal syntax is as follows:

```

1  switch (<variable name>)
2  {
3      case (<literal 1>):
4          <statement block 1>
5          break;
6      case (<literal 2>):
7          <statement block 2>
8          break;
9      ...
10     default:
11         <statement block n>
12         break;
13 }
```

The (...) are mandatory, the {...} are optional.

- All the literals need to be different.
- The literal and the variable have to be of the same type.
- You can't have case(<variable name>)

For instance, imagine we want to go from a month's number to its name. We could do that with an if...else if ...:

```

1  int month = 11;
2  string monthname;
3  if (month == 1) monthname = "January";
4  else if (month == 2) monthname = "February";
5  // ...
6  else if (month == 12) monthname = "December";
7  else monthname = "Error!";
```

But since we know that “month” will be a value between 1 and 12, or else we have an error, we could also have:

```

1  switch (month)
2  {
3      case (1):
4          monthname = "January";
5          break;
6      case (2):
7          monthname = "February";
8          break;
```

```

9      // ..
10     case (12):
11         monthname = "December";
12         break;
13     default:
14         monthname = "Error!";
15         break;
16 }

```

Another example, to match a section letter against 4 possibilities, where two actually result in the same behaviour:

```

1  char section = 'c';
2  string meet;
3  switch (section)
4  {
5      case ('a'):
6          meet = "MW 1-2PM";
7          break;
8      case ('b'):
9          meet = "TT 1-2PM";
10         break;
11     case ('c'):
12     case ('d'):
13         // case ('a'): Would not compile!
14         meet = "F 2-4PM";
15         break;
16     default:
17         meet = "Invalid code";
18         break;
19 }

```

5 Combining Methods and Decision Structures

Note that we can have a decision structure inside a method! If we were to re-visit the Rectangle class, we could have a constructor of the following type:

```

1  public Rectangle(int wP, int lP)
2  {
3      if (wP <= 0 || lP <= 0)
4      {
5          Console.WriteLine("Invalid Data, setting everything to 0");
6          width = 0;
7          length = 0;
8      }
9      else
10     {
11         width = wP;
12         length = lP;

```

13 }
14 }