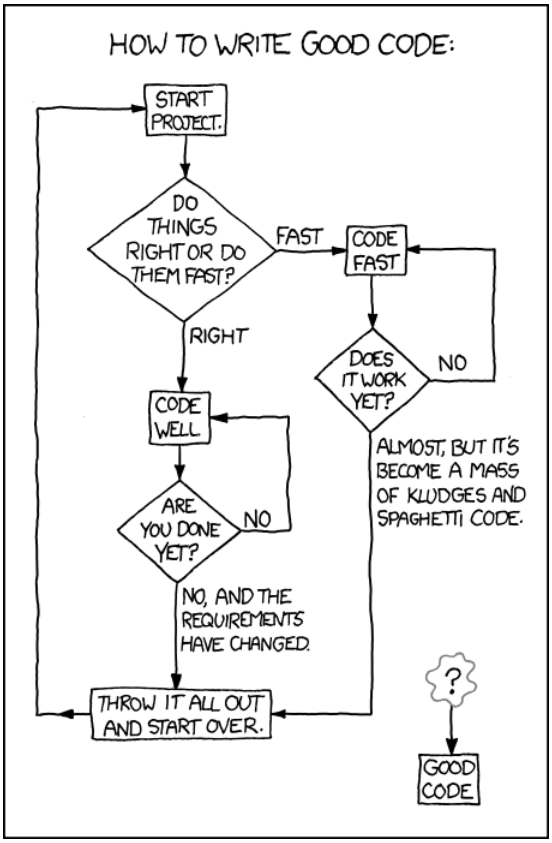
Assignment 6: Loopy games

ETH Zurich

Hand-out: Friday, 21 October 2011
Due: Tuesday, 1 November 2011



Good code © Randall Munroe (http://xkcd.com/844/)

Goals

- Practice loops and conditionals.
- Implement a system consisting of multiple classes.

1 Loop painting

To do

Write a program that does the following:

1. Asks the user to input a positive integer.
2. Displays, using asterisks, a checkered right-angled triangle having as hypotenuse a number of asterisks equal to the user input (see Figure 1.). Asterisks and white spaces should be alternating, both horizontally and vertically.
3. Displays, using asterisks, a diamond having as side the same number of asterisks as the user input. Here as well, asterisks and white spaces should be alternating.
4. Take into consideration that the user might not always input values you expect. Make sure your program does not crash, no matter what the user inputs.

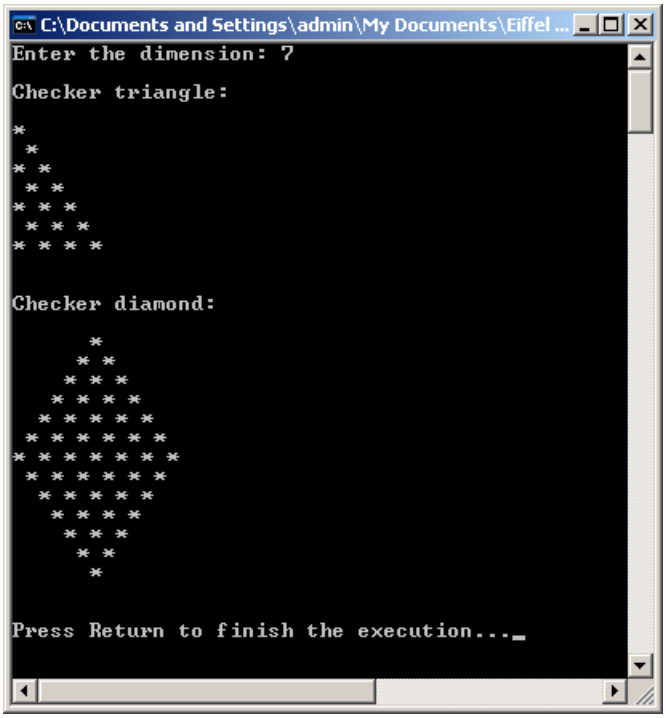


Figure 1: Example with value 7

To hand in

Hand in your class text.

2 Bagels

You are to write a program that plays a game called Bagels. It is a variation of Mastermind that uses the digits 1 to 9 (no zeros). Here is the specification:

- The player specifies a positive number n. The program generates an n-digit number that the player will try to guess.
- The program asks the player to provide a guess. Only n-digit numbers with no zero digits are accepted.
- After each guess, the program gives clues as to how close the player is to getting the right answer.
  - For each right digit in the right position, it will say `Fermi`.
  - For each right digit in the wrong position, it will say `Pico`.
  - If there are no right digits, it will say `Bagels`.
- The program should list all of the `Fermi`s first and then all of the `Pico`s. Because of this, the player can't assume anything from the order of the clues.

To do

1. In this and the following tasks you might want to use some classes from the EiffelBase2 library[1] (for example arrays or the random number generator). Download the library from `http://se.inf.ethz.ch/courses/2011b_fall/eprog/assignments/06/base2.zip` and extract it to `$ISE_EIFFEL/library`, where `$ISE_EIFFEL` is the installation directory of EiffelStudio (for example `C:\ProgramFiles(x86)\EiffelStudio\`). Now you should have the `base2` directory inside the `library` directory along with other libraries, such as `base`, `gobo` and `vision2`. If you cannot move `base2` to the specified location (for example, because of the User access control on Windows), use any other directory.

   Class names in EiffelBase2 start with the prefix *V_*,[2] e.g. *V_ARRAY*, *V_RANDOM*.

2. Download the program skeleton from `http://se.inf.ethz.ch/courses/2011b_fall/eprog/assignments/06/bagels.zip` and open `bagels.ecf`. Add the EiffelBase2 library to your project by clicking on "Add library" at the top of the "Groups" tool. If you placed the library in the default location, it will appear in the list of available libraries; otherwise click "Browse" and point to the location you chose. You have to compile the system before you can access classes from the new library.

3. Implement the Bagels game inside the class *BAGELS*.

   Hint. To generate a sequence of random numbers, you can use the class *V_RANDOM*. Here is an example of how it works:

```
-- Print random numbers in range [1..100] until we hit 13:
local
    random: V_RANDOM
do
    from
        create random -- Create a random sequence
    until
        random.bounded_item (1, 100) = 13
    loop
        print (random.bounded_item (1, 100)) -- Print current sequence element
```

[1] EiffelBase2 is a successor of the standard EiffelBase library; most of the time, EiffelBase2 classes are smaller and easier to use. You can, of course, still use EiffelBase if you prefer, however in this case you might have to modify the code templates we provide.

[2] It stands for "verified".

```
            print ("%N")
            random.forth -- Go to the next element
        end
    end
```

4. We prepared several tests that will help you determine if your Bagels implementation is correct. When you are finished with the *BAGELS* class, go to `Project -> Project Settings`, click on "Target: bagels" in the tree on the left, and then change the "Root" entry on the right side from *BAGELS.execute* to *TESTER.test*. Compile and run your program again.

   Now the system has a different entry point: it starts executing from another routine. Instead of playing Bagels with the user, it will exercise the function *clue* you wrote on a predefined set of inputs and compare the function's return values with the known correct answers. Whenever the values are not the same, it will report a mismatch.

   If any tests fail for your program, try changing it until all 10 tests pass. You are not supposed to change the class *TESTER*.

To hand in

Hand in the code of class *BAGELS* and the result of running the tests (screenshot or text).

3 Board game: Part 2

In this task you will implement a given set of classes. They may not coincide with the ones you picked last week, but it is easier to go on altogether this way.

As a reminder, you will find below the description of the problem. It has been slightly modified because it mentions six-sided dice. While this is a little detail, it gives you an idea of the fact that across different iterations of the design and development process the specifications can actually change.

The board game comes with a *board*, divided into 40 *squares*, a pair of six-sided *dice*, and can accommodate 2 to 6 *players*. It works as follows:

- All players start from the first square.
- One at the time, players take a *turn*: roll the dice and advance their respective *tokens* on the board.
- A *round* consists of all players taking their turns once.
- The winner will be the player that first advances beyond the 40th square.

To do

Implement the prototype of the board game using the following classes:

- *GAME*: encapsulates the logic of the game (start state, the structure of a round, ending conditions).
- *DIE*: provides random numbers in the required range.
- *PLAYER*: stores the state of each player in the game and performs a turn.

You can also use class *APPLICATION* as root class of your system, which is responsible for interaction with the user. Don't forget to add the EiffelBase2 library to your system (see task 2 for instructions).

To hand in

Submit the code of classes *GAME*, *DIE*, *PLAYER*, *APPLICATION*.

If you feel lost...

... you can download the class skeletons from `http://se.inf.ethz.ch/courses/2011b_fall/eprog/assignments/06/board_game.zip`.