Student: Stănescu Andreea-Francesca

Laborator 14

1.

```
19  function Fib (n: nat) : nat
20  decreases n
21  {
22      if n < 2 then n else Fib(n - 2) + Fib(n - 1)
23  }
24
```

Verificam terminarea functiei prin testarea diferitelor valori ale lui n pentru a vedea pana unde poate verifica Dafny corectitudinea functiei fara erori.

```
26  method TestFib()
27  {
28      assert Fib(0) == 0;
29      assert Fib(1) == 1;
30      assert Fib(2) == 1;
31      assert Fib(3) == 2;
32      assert Fib(4) == 3;
33      assert Fib(5) == 5;
34      assert Fib(6) == 8;
35      assert Fib(7) == 13;
36      assert Fib(8) == 21;
37      assert Fib(9) == 34;
38      assert Fib(10) == 55;
39      assert Fib(20) == 6765;
40      assert Fib(50) == 12586269025;
41      assert Fib(100) == 354224848179261915075;
42  }
```

Observam ca Davny verifica corectitudinea functiei si pentru n = 100, dar in cazul in care incercam sa verificam corectitudinea pentr n = 1000, verificatorul nu mai poate face fata. Asadar verificatorul va functiona pentru valori ale lui n care apartin intervalului [0,1000).

```
26  method TestFib()
27  {
28      assert Fib(0) == 0;
29      assert Fib(1) == 1;
30      assert Fib(2) == 1;
31      assert Fib(3) == 2;
32      assert Fib(4) == 3;
33      assert Fib(5) == 5;
34      assert Fib(6) == 8;
35      assert Fib(7) == 13;
36      assert Fib(8) == 21;
37      assert Fib(9) == 34;
38      assert Fib(10) == 55;
39      assert Fib(20) == 6765;
40      assert Fib(50) == 12586269025;
41      assert Fib(1000) == 43466557686937456435688527675040625802564660517371780402481729089536555417949051890403879840079255169295922593080322634775209689623239873322471161642996440906533187938298969649928516003704476137795166849228875;
42
43  }
```

Student: Stănescu Andreea-Francesca

2.

a)

```
46  function F(x: int ): int
47  {
48      if x < 10 then x else F(x - 1)
49  }
```

```
46  function F(x: int ): int
47  decreases x
48  {
49      if x < 10 then x else F(x - 1)
50  }
```

Clauza decreases x indica faptul ca, la fiecare apel recursiv, x scade strict. Deoarece x este un intreg si scade cu 1 la fiecare apel, in cele din urma va ajunge la o valoare mai mica de 10, asigurand astfel terminarea recursiei.

b)

```
52  function G(x: int): int{
53    if 0 <= x then G(x - 2) else x
54  }
```

```
52  function G(x: int): int decreases x {
53    if 0 <= x then G(x - 2) else x
54  }
```

Deoarece x scade cu 2 la fiecare apel recursiv, recursia va ajunge in cele din urma la cazul de baza (când 0 <= x nu mai este adevărat).

c)

```
57  function H(x: int): int{
58    if x < -60 then x
59    else H(x - 1)
60  }
```

```
57  function H(x: int): int decreases x{
58    if x < -60 then x
59    else if x <= 0 then x
60    else H(x - 1)
61  }
```

Am adaugat o ramura de else in plus, deoarece aceasta modificare asigura ca recursia se va opri atunci cand x devine suficient de mic, eliminand astfel problema legata de expresia de decrescere care nu este marginita inferior de catre 0.

Student: Stănescu Andreea-Francesca

d)

```
64  function I(x: nat, y: nat): int{
65    if x == 0 || y == 0 then
66      12
67    else if x % 2 == y % 2 then
68      I(x - 1, y)
69    else
70      I(x, y - 1)
71  }
```

```
64  function I(x: nat, y: nat): int decreases x + y {
65    if x == 0 || y == 0 then
66      12
67    else if x % 2 == y % 2 then
68      I(x - 1, y)
69    else
70      I(x, y - 1)
71  }
```

În această versiune a functiei I, am adaugat clauza decreases x + y, indicand ca suma celor doua argumente (x si y) serveste ca masura de descrestere.