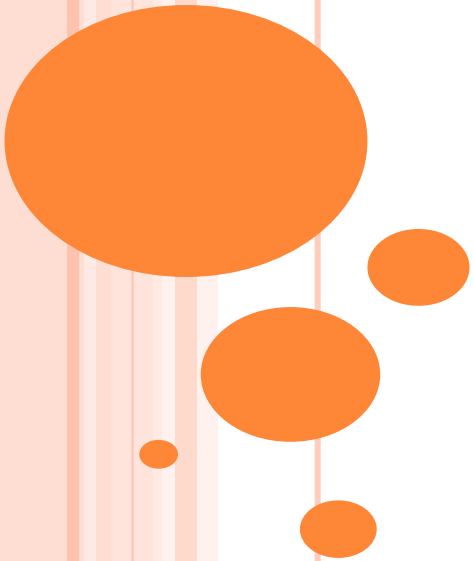
Algoritmi in podatkovne strukture 1



8. laboratorijske vaje

Programerske izpitne naloge (Dodatna naloga)

DODATNA NALOGA

Celi del logaritma nekega števila se računa tako, da se prešteje, kolikokrat je treba število (celoštevilčno) deliti z osnovo logaritma, da število postane manjše od osnove.

a) Sestavi iterativno funkcijo za računanje celega dela logaritma danega števila n pri osnovi b.

b) Sestavi rekurzivno funkcijo za računanje celega dela logaritma danega števila n pri osnovi b.

c) Spremeni rekurzivno verzijo iz naloge b) v iterativno z uporabo sklada.

REŠITEV A)

```
public static int celiDelLogIter(double n, double b) {
    int log = 0;

    while (n > b) {
        log++;
        n = n/b;
    }

    return log;
}
```

Časovna kompleksnost: O(log n).

REŠITEV B)

```
public static int celiDelLogRek(double n, double b) {
    if (n < b)
        return 0;
    else
        return 1 + celiDelLogRek(n/b, b);
}
```

Časovna kompleksnost: O(log n).

REŠITEV C)

```
public static int celiDelLogRek(double n, double b) {
    if (n < b)
        return 0;
    else {
        int tmp = celiDelLogRek(n/b, b);
        return 1 + tmp;
    }
}
```

address0 (začetek funkcije)

address1 (povratek iz rekurzivnega klica)

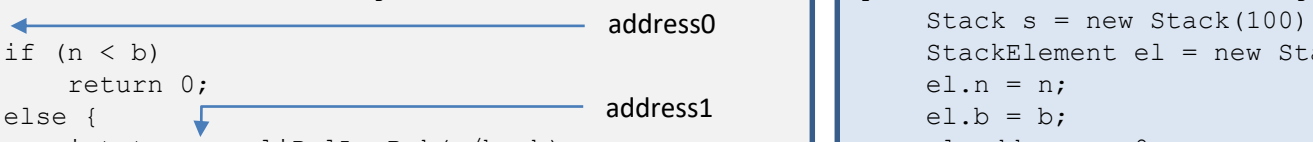
```
class StackElement
{
    double n;      // <------------------------- prvi argument rekurzivne funkcije
    double b;      // <------------------------- drugi argument rekurzivne funkcije
    int tmp;       // <------------------------- lokalna spremenljivka v rek. funkciji
    int address;   // <------------------------- povratni "naslov"

    public StackElement(){}

    public StackElement(StackElement el) {
        n = el.n;
        b = el.b;
        tmp = el.tmp;
        address = el.address;
    }
}
```

REŠITEV C)

```
public static int celiDelLogRek(double n, double b) {
    if (n < b)
        return 0;
    else {
        int tmp = celiDelLogRek(n/b, b);
        return 1 + tmp;
    }
}
```




```
public static int celiDelLogIterStack(double n, double b) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.n = n;
    el.b = b;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        switch(el.address) {
        case 0:
            if (el.n < el.b)
                result = 0;
            else {
                el.address = 1;
                s.push(new StackElement(el));

                el.n = el.n/el.b;
                el.address = 0;
                s.push(new StackElement(el));
            }
            break;

        case 1:
            el.tmp = result;
            result = 1 + el.tmp;
            break;
        }
    } while (!s.empty());

    return result;
}
```