1. Uvod - Modeli računanja, strukturirano programiranje

1.1.Von Neumannov model in njegova realizacija CARDIAC

Opis von Neumannovega modela računalnika.

- Pomnilnik
- Vhodna enota
- Izhodna enota
- Ukazna enota
- Logično-aritmetična enota

Pomnilnik je sestavljen iz zaporednih celic. Vsaka celica ima naslov. Celice vsebujejo lahko ukaze in podatke.

Vhodna enota. Omogoča branje podatkov.

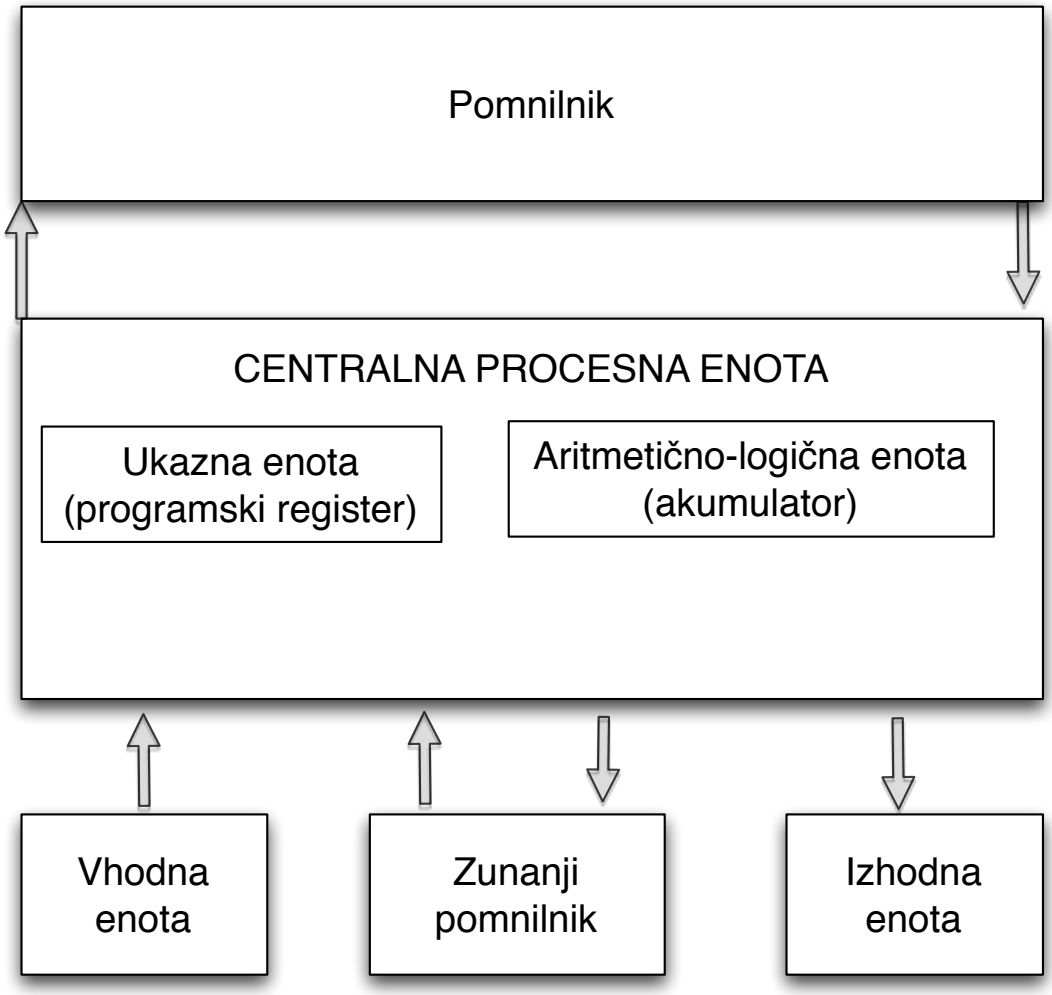
Izhodna enota. Omogoča pisanje podatkov.

Ukazna enota. Naloži tekoči ukaz, ga analizira, izvede in nadaljuje izvajanje pri naslednjem ukazu. Naslednji ukaz je običajno shranjen takoj za trenutnim ukazom. Obstajajo tudi skoki, ki v odvisnosti od vsebine akumulatorja oz. registrov prenese kontrolo drugam.

Logično-aritmetična enota (akumulator, oz. registri). Izvaja logične in aritmetične operacije.

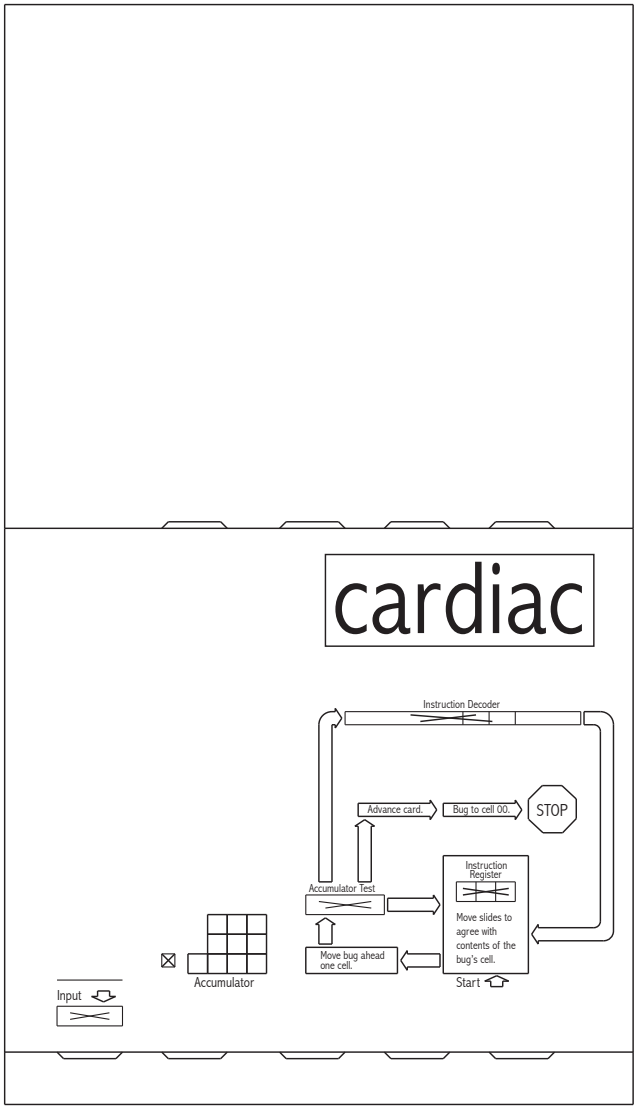
Model ima lahko več vhodnih in izhodnih enot. Vsaj ena od izhodnih enot med njimi pa zapisuje podatke v obliki, ki je berliva za računalnik. Običajno je to kar:

Vhodno-izhodna enota z zamenljivim **zunanjim pomnilnikom**.

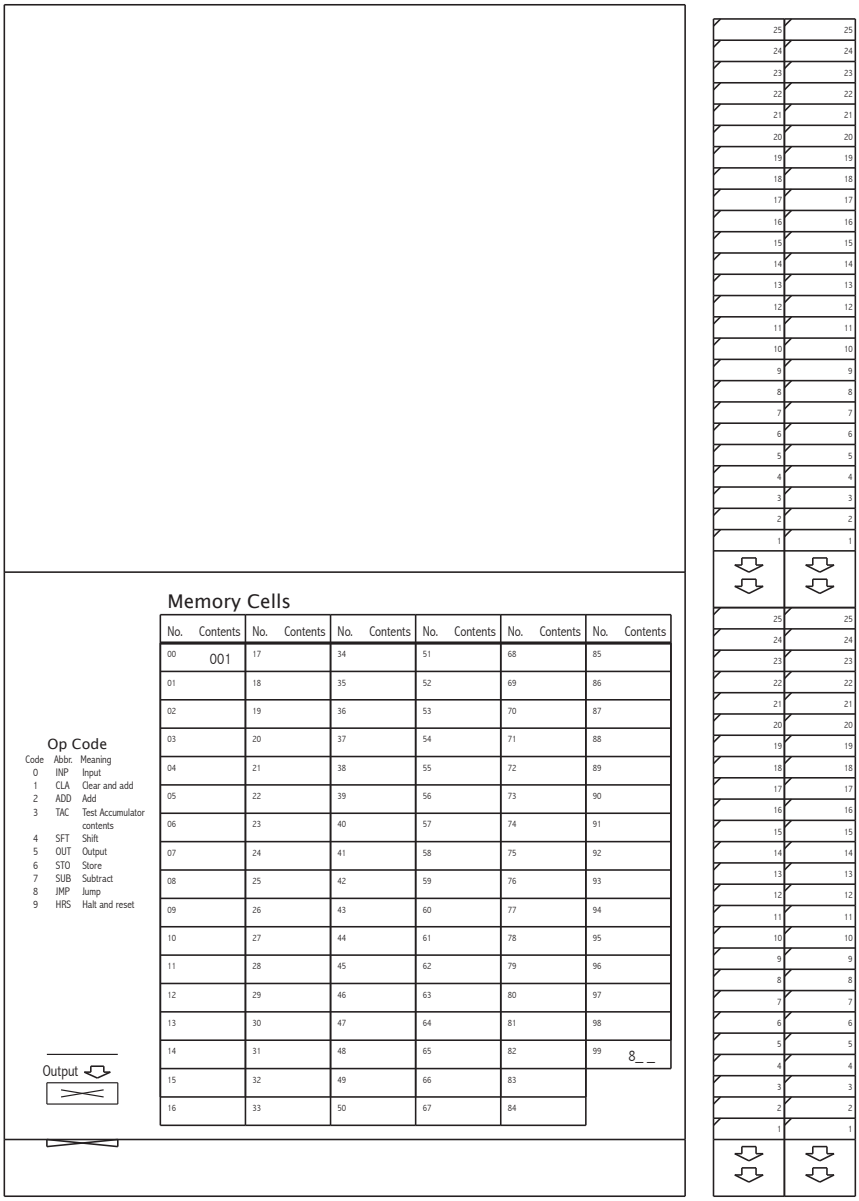


1.2. CARDIAC

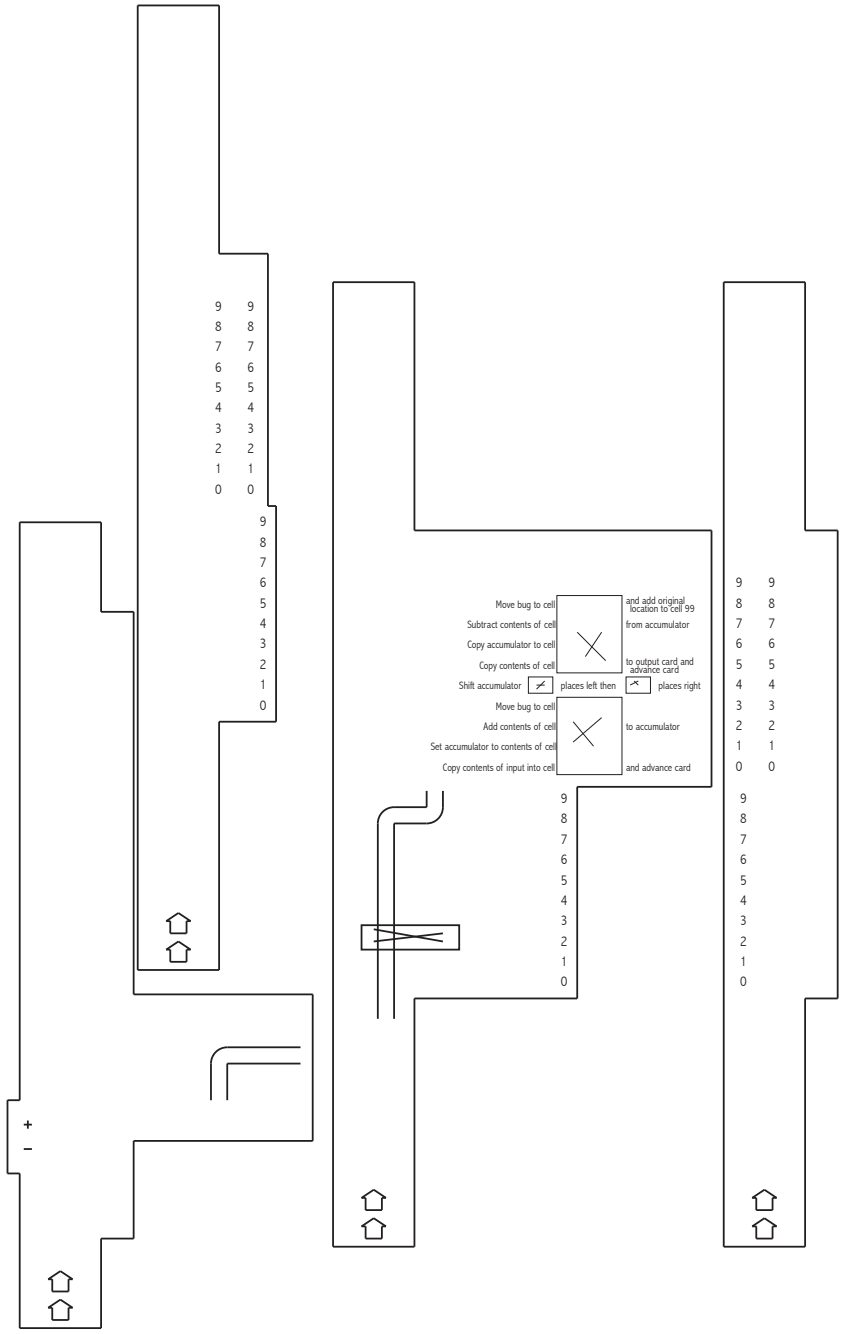
Ogledali si bomo zelo preprosto realizacijo von Neumannoveg modela v obliki računalnika CARDIAC.



CARDIAC in njegova logično-aritmetična enota



CARDIAC in njegov pomnilnik ter vhodno-izhodne enote



CARDIAC in njegova ukazna enota, preden jo vgradimo v sistem,

CARDIACov ukaz je oblike kxy, kjer je k koda operacije, xy pa naslov pomnilmiške celice. Ukazi so:

0 INP – input Preberemo tekočo celico vhodnega traku in jo shranimo v celico z naslovom xy. (Trak premaknem potem za eno mesto naprej)

1 CLA – clear and add V akumulator naložimo vsebino celice z naslovom xy.

2 ADD – add Vsebini akumulatroja prištejemo vsebino celice z naslova xy.

3 TAC – test accumulator Poglej preznak akumulatorja in nadaljuj bodisi z naslednjim ukazom, bodisi z ukazom, ki je shranjen v celici z naslovom xy. (Kakšen je predznak števila 0? V katerem primeru skočimo drugam?)

4 SHF – shift Tega ukaza ne bomo uporabljali

5 OUT – output Zapiši vsebino celice z naslovom xy na izhodni trak. (Trak premaknemo za eno mesto naprej)

6 STO – store Vrednost akumulatorja zapišemo v celico z naslovom xy. (Vprašanje: Ali se prepiše tudi predznak?)

7 SUB –subtract V akumulatorju odštejemo vsebino celice z nalsovom xy. (Kaj se zgodi, če pride do prekoračitve obsega?)

8 JMP – unconditional jump (Program nadaljuje na naslovu xy)

9 HLT – halt and reset (Proram se ustavi, programski števec se postavi na celico z naslovom 00.

Pomnilnik ima 100 celic z naslovi 00-99. Celici 00 in 99 sta malo drugačni od ostalih celic.

Celica 00 ima ves čas vsebino 001. Ukaz STO 00 je ne spremeni(?).

Celica 99 ima vsebino delno določeno 8--. (Kaj to pomeni?)

1.3. Ukazi preprostega računalnika

Računalnik je malo bolj sposoben kot von Neumannov. Ima naslednje ukaze, ki jih pišemo v vrsticah.

1. `x = c, y = f(x) ali z = g(x,y)`. (prireditev)
2. `če p(x) pojdi na vrstico l.`(pogojni skok)
3. `x = preberi()` (branje iz vhodne enote)
4. `natisni(x)` (pisanje na izhodno enoto)
5. `končaj`

Pri tem so c, f,g, in p vgrajene konstante, funkcije, oz. predikati, program pa se izvaja načeloma od prve vrstice navzdol.

Zgledi:
A) Napišite program, ki izpiše prvih deset naravnih števil. Naštejete vgrajene konstante, funkcije in predikate, ki ste jih pri tem uporabili.

```
1. n = 10
2. i = 1
3. natisni(i)
4. i = prištejena(i)
5. če je manjšienak(i,n) pojdi na 3.
6. končaj
```

Uporabljene konstante: 10,1.
Uporabljena funkcija prištejena(i) := i + 1
Uporabljeni predikat: majšienak(i,n) := i <= n.
Uporabljena oznaka vrstice: 3.

B) Napišite program, ki sešteje števili m in n. Pri tem uporabite konstanto 0, predikat jenič(x), ter vgrajeni funkciji prištejena(x) ter odštejena(x).

C) Napišite program, ki prebere število m, osnovo b ter izpiše števke števila m pri osnovi b. Npr. če je m = 2010 in je b = 3, mora program izpisati m v trojiškem sistemu.

D) Napišite program, ki preberi števili m in n ter izpiše njun najmanjši skupni večkratnik. Npr. za m = 10 in n = 12 naj izpiše v = 60. Naštejete uporabljene konstante, funkcije ter predikate.

E) Za vse zglede od A) do D) napišite program za CARDIAC.

1.4. Nekatere razlike med von Neumannovim modelom in preprostim računalnikom:

1. Pomnilnik je bolj sofisticiran. Celice nosijo imena. Program ni shranjen na istem mestu kot podatki.

2. Vhodna enota je zmožna branja različnih tipov podatkov.

3. Izhodna enota lahko izpisuje različne tipe podatkov.

4. Ukazna enota je skrita. Namesto ukaznega registra si moramo zapomniti katera ukazna vrstica se izvaja.

5. Aritmetično logični del je zelo zmogljiv in omogoča računanje funkcije in shranjevanje njenega rezultata, ne da bi morali skrbeti za akumulator (oz. registre).

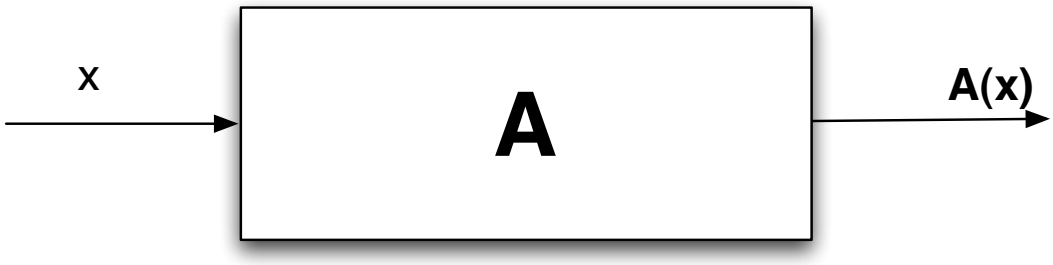
Slabosti obeh modelov: ukaz za skok (jump, goto). S tem lahko grdo šarimo po programu in zelo težko obdržimo kontrolo nad dogajanjem. Temu se lahko z malo discipline izognemo. V šestedsetih letih prešnjega stoletja je Edsgar Dijkstra formuliral paradigmo strukturiranega programiranja.

1.5. Strukturirano programiranje

Metoda reševanja problemov od zgoraj navzdol: paradigma strukturiranega programiranja.

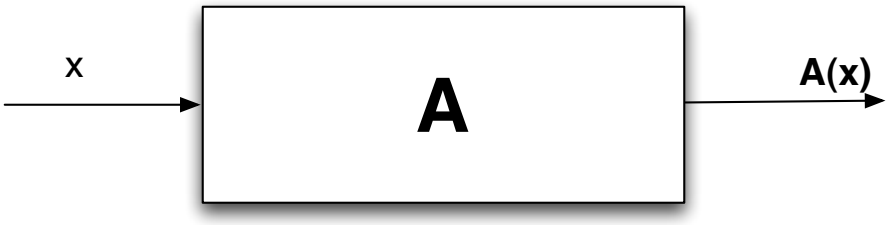
1. Zaporedje,
2. odločitev,
3. zanka,
4. funkcija ter osnovna operacija,
5. stranski učinki in druge izjeme.

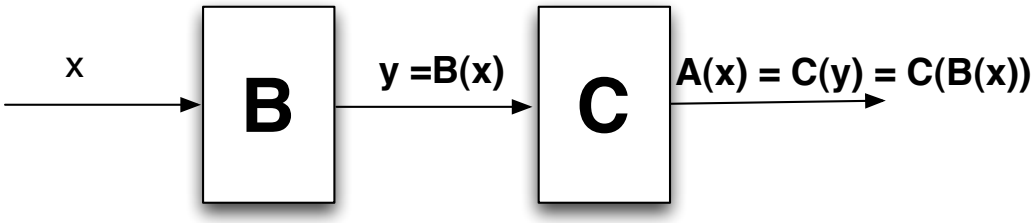
Začnemo s hipotezo, da vsak del programa realizira neko funkcijo y = A(x). To si predstavljamo kot črno škatlo, ki ima en sam vhod in en sam izhod ter nimam stranskih učinkov.



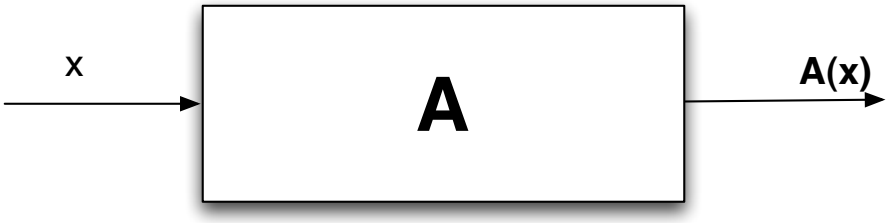
Črna škatla, ki računa y= A(x).

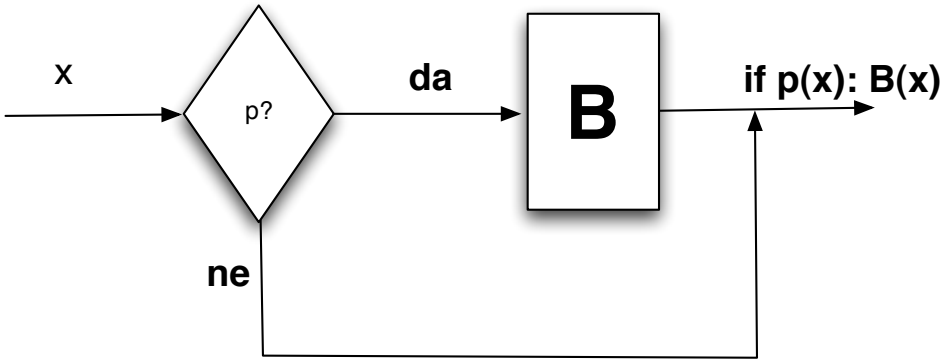
Zdaj lahko tak program (algoritem) razdelimo na bolj preproste po naslednjih načelih.



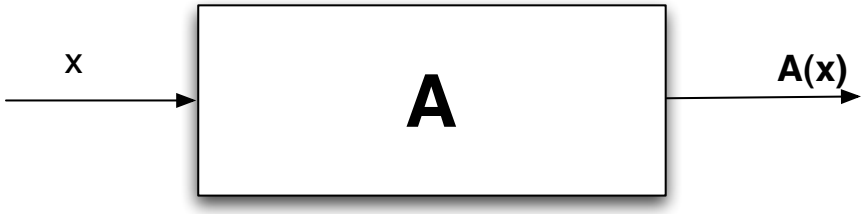


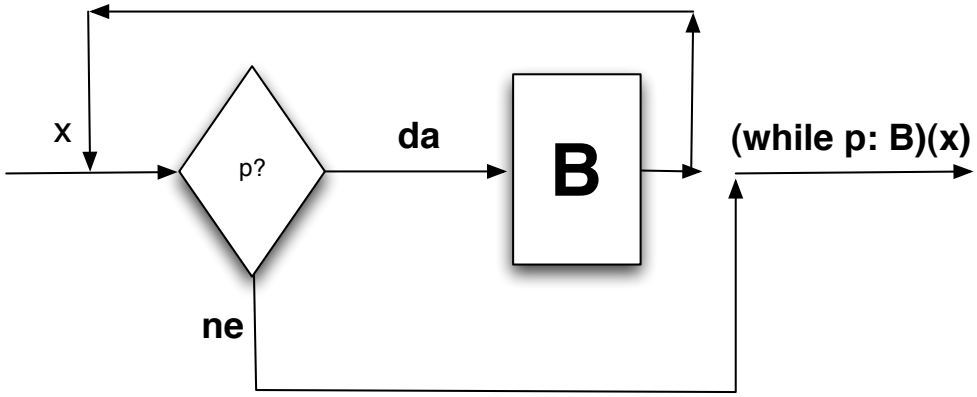
Zaporedje



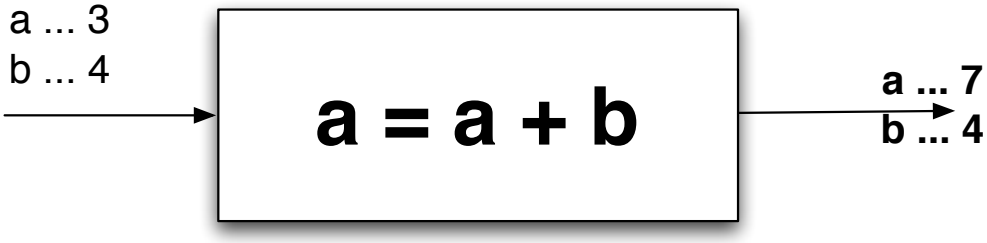


Odločitev izvedemo s stavkom if.

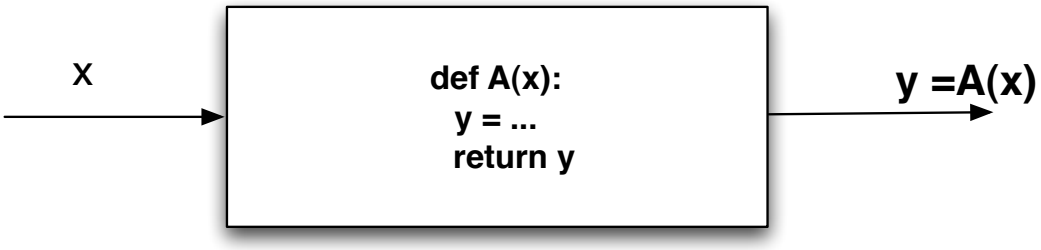




Zanko izvedemo s stavkom while. (Kasneje bomo videli še druge zanke. V jeziku Python je ena med njimi precej močnejša od zank v starejših jezikih.)

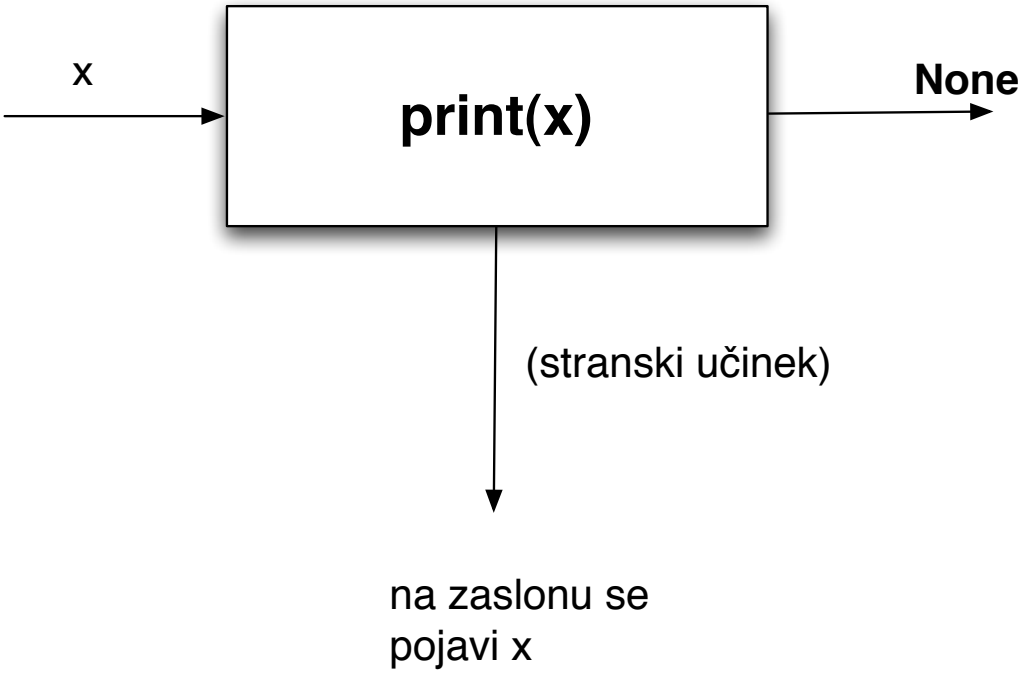


Prireditveni stavek in vgrajeni ukazi. Izraz na desni strani izračunamo in ga priredimo spremenljivki na levi strtani.



Funkcije so idealna podpora za kreativno lenobo. Med programiranjem lahko vedno privzamemo, da je funkcija A že na voljo. Lahko jo uporabimo v prireditvenem stavku y = A(x). Če take funkcije ni na voljo, jo moramo na drugem mestu posebej sprogramirati. V ta namen uporabimo stavka def in return. S stavkom def opozorimo, da gre za začetek funkcije, s stavkom return, ki je običajno na koncu programa, pa povemo, kaj je rezultat funkcije.

V praksi pogosto uporabljamo stavek return na več mestih, to pomeni, da je izhodov lahko več. Tudi vhodov bi lahko bilo več. To vse zlahka zapremo v črno škatlo. Teže pa se izognemo stranskih učinkom.



Funkcija print() v Pythonu ima stranske učinke. Njena vrednost je None, kot stranski učinek pa je izpis na zaslonu.

1.6. Zgled programiranja v pyhtonu v skladu s paradigmo strukturiranega programiranja.

Naloga: Napišite del programa, ki za znani števili a = 45 in b= 32, izračuna njun največji skupni delitelj gcd(a,b). Pri programiranju uporabite naslednje lastnosti funkcije gcd:
(X) gcd(a,b) = gcd(b,a)

(Y) gcd(a,b) = gcd(a,b-a)
(Z) gcd(0,b) = b

V bistvu gre za znani Evklidov algoritem.

```
# 20.2.2013
# Zgled, kako se v Pythonu uporablja konstrukte
strukturiranega
# programiranja pri računanju gcd, če vemo:
# (X) gcd(a,b) = gcd(b,a)
# (Y) gcd(0,b) = b
# (Z) Če a <= b, potem gcd(a,b) = gcd(a,b-a)

#a = int(input("Vtipkaj a: "))
#b = int(input("Vtipkaj b: "))
# print(a,b)
a = 45
b = 32

if a > b:
    t = a
    a = b
    b = t
#    print(a,b)
while a > 0:
#    while b >= a:
#        b = b -a
    b = b%a
    t = a
    a = b
    b = t
#        print(a,b,t)
c = b
#print("Najvecji skupni delitelj je ",c)
```

Še ena okleščena verzija programa:

```
m = int(input(""))
n = int(input(""))
while m > 0:
    while n >= m:
        n = n-m
    t = n
    n = m
    m = t
print(n)
```

Pa še diagram poteka zanjo: