Увод у рачунарство 1

Александар Картељ

kartelj@matf.bg.ac.rs

Увод у програмирање (Python)

Алгоритам и програм

Шта ради рачунар?

- Најједноставније речено:
  - Извршава израчунавања – милијарде у секунди!
  - Затим памти резултате тих извршавања – стотине гигабајта меморије!
- Који су то типови израчунавања?
  - Израчунавања подржана језиком којим му се обраћамо
  - Рачунар не зна да уради нешто што му нисмо рекли
  - Чак и оно што зовемо вештачка интелигенција је под нашом контролом само мање разумемо след корака који се ту дешава

Пример

- Пронаћи квадратни корен број x, тј. y такав да је y*y = x
- Како може да изгледа „рецепт“ за решавање овог проблема?
- 1) Започнимо са неким **покушајем** g
- 2) Ако је g*g **довољно близу**, прекидамо тражење, g је одговор
- 3) Иначе правимо **нови покушај** тако што нађемо просек између g и x/g
- 4) Помоћу новог покушаја, **понављамо** исти поступак почев од тачке 1)

g	g*g	x/g	(g+x/g)/2
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

Рецепт

- Шта садржи рецепт?

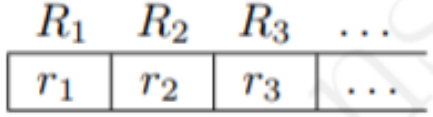
1. Низ једноставних **корака**
2. **Ток контроле** који опсиује када је сваки од корака извршен
3. Механизам којим се одређује **када се рецепт завршава**

То све заједно се зове **алгоритам**!

Запис алгоритма у неком конкретном језику се зове **програм**!

URM – машина са бесконачним регистром

- енг. Unlimited register machine
- Теоријски концепт рачунара који демонстрира:
  - да је за рачунање било које математички израчунљиве функције довољно користити 4 примитивне функције
  - уз довољно велику количину меморије
- Постоје и други еквивалентни системи: Тјурингова машина, Постова машина, итд.

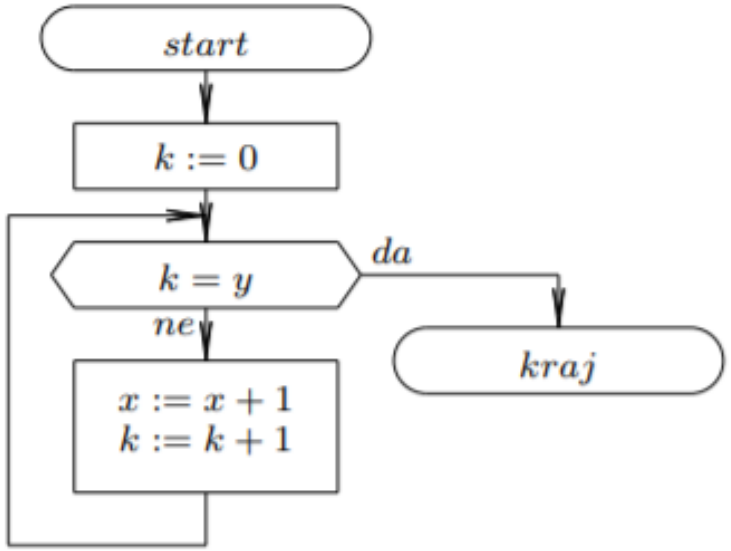


oznaka	naziv	efekat
$Z(m)$	nula–instrukcija	$R_m \leftarrow 0$ (tj. $r_m := 0$)
$S(m)$	instrukcija sledbenik	$R_m \leftarrow r_m + 1$ (tj. $r_m := r_m + 1$)
$T(m,n)$	instrukcija prenosa	$R_n \leftarrow r_m$ (tj. $r_n := r_m$)
$J(m,n,p)$	instrukcija skoka	ako je $r_m = r_n$, idi na p-tu; inače idi na sledeću instrukciju

URM пример – сабирање два броја

- Нека је задатак сабрати два броја *x* и *y* при чему су ти бројеви записани на позицијама регистара R_1 и R_2
  - Коначни резултат је потребно исписати у регистру R_1
- Решење:
  - Приметити да не постоји у датим примитивама функција сабирања два броја
  - Међутим, постоји инструкција следбеника која повећава број за 1
  - Приметимо да је *x+y = x+1+…+1* (*y* пута се сабира број 1)
  - Итерирање *y* пута се може постићи инструкцијом скока

Сабирање два броја - решење



1. Z(3)
2. J(3, 2, 100)
3. S(1)
4. S(3)
5. J(1, 1, 2)

За детаље о URM прочитати поглавље 3.3 књиге
Програмирање 1: Основе програмирања кроз програмски језик C, Филип Марић и Предраг Јаничић, 2018, http://poincare.matf.bg.ac.rs/~janicic//books/p1.pdf

URM задаци

1. Имплементирати URM програм који множи два броја
2. Имплементирати URM програм који враћа 1 ако је улазни број паран, а у супротном враћа вредност 0 у првом регистру

Изражајност програмских језика

- Сваки програмски језик има одређени скуп инструкција
  - Тих инструкција обично има **много више** од 4 као код URM
  - Међутим, сви стандардни програмски језици су **подједнако изражајни** као и URM
  - Додатне инструкције не доприносе већој изражајности већ само томе да програми буду краћи и прегледнији
  - Теоријски гледано су **еквивалентни** – све што може један, може и други –
- Алгоритам представља начелни поступак решавања неког проблема
- Док програм представља реализацију алгоритма на конкретном програмском језику
- Ми ћемо надаље проблеме решавати у програмском језику Python!

Извршни програм (exe)

- Извршни програм је програм који разуме процесор рачунара
- Он се састоји од **низа (машинских) инструкција** које су врло примитивне, слично URM инструкцијама:

1. Аритметичко-логичка (+, -, /..)
2. Једноставно поређење (==, !=, >, …)
3. Инструкција померања из једног дела меморије у други

- Зашто програме ретко пишемо на машинском језику?
  - Јер би били јако нечитљиви и тешки за писање и одржавање
  - Међутим, ако бисмо их добро написали они би били врло ефикасни!

Превођење између програмских језика

- Писање програма подразумева прављење **изворног кода**
- Изворни код програма се не може директно извршити на процесору
  - Процесор не зна то да разуме
- Зато је потребно извршити превођење изворног кода на машински код
- Ово је могуће употребом програмских преводилаца
- Две групе:

1. Интерпретери – преводе изворни код у машински након покретања
   - Python, PHP, JavaScript, …
2. Компајлери – преводе изворни код у машински пре покретања
   - C, C++, C#, …

Увод у програмирање (Python)

Основни елементи језика – промењљиве, вредности, оператори...

Python програми

- **Програм** је низ дефиниција и наредби (инструкција)
  - Дефиниције се евалуирају
  - Наредбе се извршавају од стране Python интерпретера
- Наредбе говоре интерпретеру шта да ради
- Наредбе се могу унети на два начина:
  1. Директно у конзоли
  2. У скрипт датотеку која се накнадно покреће

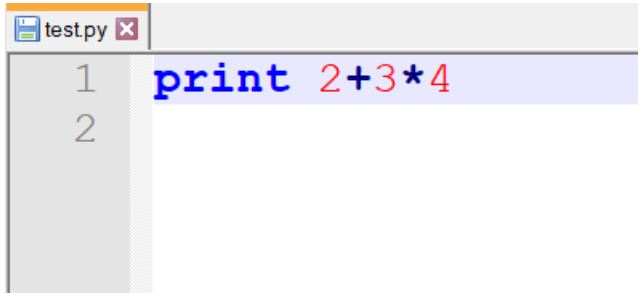
Задавање наредбе путем конзоле

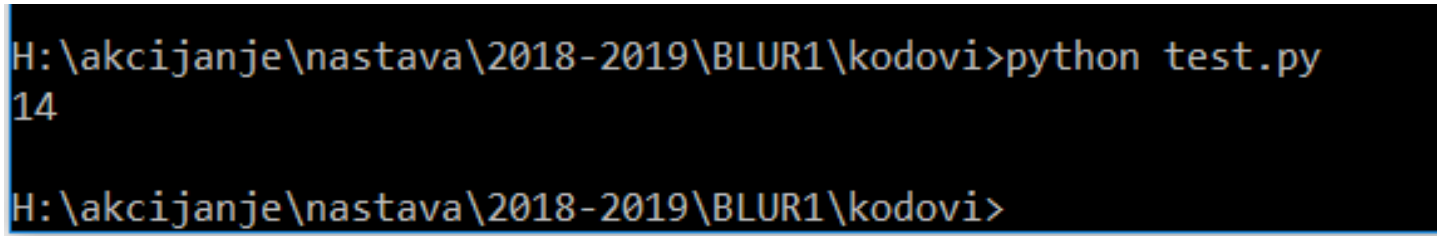
1. Отворити конзолу
2. Укуцати **python**
3. У интерактивном режиму извршавати наредбе, нпр. 2+3

```
H:\akcijanje\nastava\2018-2019\BLUR1\kodovi>python
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 2 * 8 + 4
20
>>>
```

Задавање наредби употребом скрипте

1. Направити датотеку и дати јој екстензију **py**
2. Отворити је употребом неког едитора или развојног окружења и унети наредбе
3. Позиционирати се путем конзоле у директоријум где се налази **py** датотека
4. Покренути је позивом наредбе **python <назив датотеке>**





Пример

- Написати програм који исписује „Здраво свете“
-
```
# Ovo je komentar
# Za ispis teksta koristimo funkciju print
print("Zdravo svete!")
```

Пример (2)

- Написати програм који исписује квадрат унетог целог броја

```
# Unos vrednosti je omogućen funkcijom input
# koja kao argument prihvata tekst ispisan pri unosu
x = int(input("Unesite ceo broj: "))
'''
Ovo je višelinijski komentar,
može biti koristan ako u kodu objašnjavamo nešto složenije
'''
y = x*x
''' Ispisujemo rezultat, samo broj prvo konvertujemo u tekst pošto ne
možemo da sabiramo tekst i broj '''
print("Rezultat je "+str(y))
```

```
Unesite ceo broj: 67
Rezultat je 4489
```

Променљиве

- Променљиве унутар програма могу бити:
  - **Скалари** - немају подподатке
  - **Не-скалари** – имају подподатке односно интерну структуру
- Свакој променљивој је придружен **тип**
  - Типови омогућавају придруживање карактеристика и функција
  - Нпр. Човек може да хода и има руке
  - Птица може да лети и има крила
  - Цео број се може делити са остатком
  - За реалан број такво дељење нема смисла

Скаларни типови

- int – је цео број, нпр. 5
- float – је реалан број, нпр. 3.27
- bool – је логичка вредност, True или False
- NoneType – је специјални тип који има једну вредност, None
- Помоћу наредбе type() може се проверити који је тип придружен податку
- >>> type(5)
- int
- >>> type(3.0)
- float

Додела вредности променљивој

- За доделу вредности се користи **оператор** =
- Нпр.:
- pi = 3.14159
- pi_approx= 22/7
- Вредност се смешта у меморији рачунара

Конверзије типова

- Могуће је променити тип података у неким ситуацијама
- Нпр.:
- float(3) конвертује цео број 3 у реалан број 3.0
- int(3.9) конвертује float 3.9 у цео број 3 и притом губи информације!
- str(643) конвертује цео број 6 у текст “643”

Пример (3)

- Написати програм који рачуна удаљеност између две тачке у равни

```
# Uvoz biblioteke za matematičke funkcije
import math
# Moguće je unositi i više podataka pomoću jednog poziva funkcije input
print("Unesite koordinate prve tacke: ")
• x1 = int(input())
• y1 = int(input())
print("Unesite koordinate druge tacke: ")
• x2 = int(input())
• y2 = int(input())
# Funkcija sqrt (korenovanje) je definisana u modulu math
d = math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))
# Tekst se može pisati i u tzv. formatiranoj varijanti
print('Rastojanje izmedju (%d,%d) i (%d,%d) je %.2f '%(x1,y1,x2,y2,d))
```

```
Unesite koordinate prve tacke:
45
2
Unesite koordinate druge  tacke:
5
6
Rastojanje izmedju (45,2) i (5,6) je
40.20
```

Модули

- Модули или библиотеке представљају
  групе сродних функција и података
- Програму не треба у сваком моменту свака библиотека
  тако да је потребно експлицитно нагласити увоз библиотеке
- Уколико нам је потребна нека функција, нпр. sqrt,
  морамо знати где се она налази и потом је увести наредбом import

Изрази

- Изрази се добијају комбиновањем променљивих и оператора
- Изразу се може одредити вредност која ће имати придружени тип
  - Овај тип се имплицитно одређује израчунавањем вредности
  - Python провера да ли су типови усаглашени да се не би нпр. десило: сабирање броја и логичке вредности или слично...

Оператори над бројевима

- i+j – сума два броја
- i-j – разлика два броја
- i*j - производ
- i/j – количник (резултат је реалан број)
- i%j – остатак при дељењу i са j
- i**j – i подигнут на степен j

Релацијски оператори

- == - једнако
- != - различито
- > - веће
- >= - веће или једнако
- < - мање
- <= - мање или једнако

Логички оператори

- not – логичка негација
- and - логичка конјункција
- or - логичка дисјункција

Приоритет операција

- Заграде служе за експлицитно одређивање редоследа операција
- Ако нису задате заграде, онда се редослед одређује по следећим приоритетима:
  - **
  - *
  - /
  - +, -
- У случају истог приоритета, пре се извршава оператор који је лево

Приоритет операција (2)

- Релацијски оператори уређења имају виши приоритет од оператора поређења на једнакост или неједнакост
- Релацијски оператори имају нижи приоритет од аритметичких
- Логички оператори имају нижи оператор од релационих и аритметичких оператора

Пример (4)

- Шта ће бити резултати извршавања следећег програма:

```
a = 4 + 4*7*2**3
b = 4>4 or 4+6>2
c = True or False and True
d = not 45-40>5 != False
e = not (False != (40>5) != False)
print(str(a))
print(str(b))
print(str(c))
print(str(d))
print(str(e))
```

```
228
True
True
True
False
```

Пример (5)

- Написати програм који проверава да ли је унети број паран

```
x = int(input("Unesite ceo broj: "))
# Operacija % računa ostatak pri deljenju
# dok operacija == proverava jednakost leve i desne strane
if (x % 2 == 0): # % ima viši prioritet od ==
      print("Broj je paran")
else:
      print("Broj nije paran")
```

```
Unesite ceo broj: 7
Broj nije paran
```

Наредба гранања if … elif … else

- Наредба гранања омогућава програму да у зависности од услова извршава један, други или n-ти део кода
- Услов који се проверава мора као резултат давати логичку вредност
- Потпуни облик ове команде омогућава вишеструко испитивање услова

Пример (6)

- Написати програм који провера за унети број да ли је дељив са 3, или је облика 3k+1 или 3k+2
- `x = int(input("Unesite ceo broj: "))`

```
# Poravnavanje mora da bude usaglašeno
# svaki print mora da bude isto udaljen od početka reda
if (x % 3 == 0):
      print("Broj je deljiv sa 3")
elif (x%3 == 1):
      print("Broj je oblika 3k+1")
else:
      print("Broj je oblika 3k+2")
```

Задаци за вежбу

1. Написати програм који за унети полупречник рачуна обим и површину круга (константа **pi** се налази у библиотеци **math**)
2. Написати програм који за унетих 9 бројева израчунава детерминанту

$$\begin{vmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{vmatrix}$$

3. Написати програм који исписује максимум три унета броја

Увод у програмирање (Python)

Петље и листе

Пример

- Написати програм који сабира првих 10 природних бројева

```
zbir = 1+2+3+4+5+6+7+8+9+10
print("Zbir prvih 10 prirodnih brojeva je %d"%(zbir))
```

```
Zbir prvih 10 prirodnih brojeva je 55
```

- Шта ако не знамо унапред границу, већ је уноси корисник?

```
print("Unesite granicu za sumiranje: ")
• n = int(input())
i = 1
suma = 0
while i<=n:
        suma+=i
        i+=1
print("Suma prvih %d prirodnih brojeva je %d"%(n,suma))
```

```
Unesite granicu za sumiranje:
5
Suma prvih 5 prirodnih brojeva je 15
```

```
Unesite granicu za sumiranje:
102
Suma prvih 102 prirodnih brojeva je 5253
```

Петље

- Петље (циклуси) омогућавају понављања
- На претходном слајду смо се упознали са **while** петљом
- Ова петља се извршава док год је испуњен услов задат у загради
  - Овај услов може бити потпуно произвољан
- Постоји и тзв. **for** петља која је специјализована за употребу бројачких променљивих
  - То подразумева да је услов изласка из петље заснован на унапред познатом броју итерација

Пример (2)

- Имплементирати претходни програм употребом **for** петље

```
print("Unesite granicu za sumiranje: ")
• n = int(input())
i = 1
suma = 0
for i in range(n+1): # i ide od 0 do granice - 1
      suma+=i
      i+=1
print("Suma prvih %d prirodnih brojeva je %d"%(n,suma))
```

Пример (3)

- Написати програм који учитава 3 броја и исписује их у обрнутом редоследу

```
print("Unesite 3 broja: ")
• x, y, z = int(input()), int(input()), int(input())
print("Brojevi u obrnutom redosledu su %d %d %d"%(z,y,x))
```

```
Unesite 3 broja:
6
7
3
Brojevi u obrnutom redosledu su 3 7 6
```

Пример (4)

- Шта ако треба окренути 10 уместо 3 броја?

```
# Potrebna nam je lista (ili niz)
brojevi = [3,5,3,1,1,4,6,3,2,3]
i = len(brojevi)-1 # Funkcija len vraća dužinu liste
• while i>=0:
        print(str(brojevi[i])), # Zarez posle naredbe znači da nema novog reda
        i-=1 # Skraćena notacija naredbe i=i-1
```

```
3 2 3 6 4 1 1 3 5 3
```

Пример (5)

- Шта ако не знамо унапред бројеве нити колико их је?

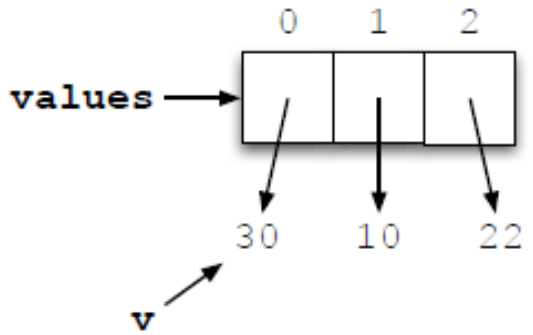
```
print("Unesite broj brojeva: ")
n = int(input())
i = 0
brojevi = [] # pravimo praznu listu brojeva
while i<n:
•       brojevi.append(int(input())) # dodajemo na kraj liste
        i=i+1

i=n-1
while i>=0:
        print(str(brojevi[i])),
        i-=1
```

```
Unesite broj brojeva:
11
2
3
4
6
1
3
5
7
5
3
5
5 3 5 7 5 3 1 6 4 3 2
```

Листе

- Листе представљају линеарне структуре података
- Подаци су у њима представљени уређено и сваком је додељен јединствени индекс у виду целог броја >= 0
- Приступ елементима листе је брз ако се задаје путем индекса
  - Ако се питамо шта је на позицији 10, врло ћемо брзо добити одговор?
- Шта ако постављамо питање да ли се вредност 10 налази у листи?
  - Алгоритам је спор и у најгорем случају морамо проћи кроз целу листу!



Функција **range**

- Видели смо раније функцију **range**
  - Испоставља се да она просто креира листу у позадини
  - Док је наредба **for** у стању да прође кроз ту листу у нотацији

```
4 4 6 7 3
4 4 6 7 3
0 1 2 3 4
```

```
brojevi1 = [4,4,6,7,3]
for b in brojevi1: # Iteriramo kroz listu
        print(str(b)),
print("") # Samo ispisujemo novi red
# Ekvivalentno je sledećem
for i in range(len(brojevi1)):
        print(str(brojevi1[i])),
print("")
# Funkcija range zapravo pravi listu u pozadini
brojevi2 = range(5)
for b in brojevi2:
        print(str(b)),
```

Пример (6)

- Написати програм који из унапред задате листе бројева исписује сваки k-ти елемент (бројање почиње од 0)

```
brojevi = [3,45,5,3,1,5,98,3,11,24,5,88]
k = int(input("Unesite broj k: "))
i = k-1 # Npr. za k=3, krećemo od elementa na poziciji 2
while i<len(brojevi):
      print(str(brojevi[i])),
      i+=k
```

```
Unesite broj k: 3
5 5 11 88
```

Задаци

- Написати програм који проналази највећи број у задатој листи
- Написати програм који рачуна просек елемената листе
- Написати програм који учитава n и рачуна n!
- Корисник уноси позитиван цео број n, а потом и n реалних бројева. Одредити колико пута је приликом уноса дошло до промене знака.

Пример (7)

- Написати програм који исписује таблицу множења за све бројеве од 1 до N. Корисник уноси N.

```
n = int(input())
for i in range(1,n+1):
    j = 1
    while j<=n:
        print("%d x %d = %d"%(i,j,i*j))
        j+=1
```

Угњеждене петље

- Угњеждене петље омогућавају решавање сложенијих проблема
- Угњеждене петље нису исто што и независне петље
- Нпр. ако имамо петљу Б угњеждену у петљи А,
  за сваку итерацију спољне петље А,
  изврши се у потпуности петља Б

Задаци (2)

- Паскалов троугао садржи биномне коефицијенте C(n,k). Написати програм који исписује првих m редова троугла. Нпр. за n=6 се исписује:

```
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
1 5 10 10 5 1
```

- Приметити да важи C(n,k)=C(n-1,k-1)+C(n-1,k)

Паскалов троугао - решење

На сајту је окачено, превелик је код за слајд.

Наредбе break и continue

- Понекад желимо да завршимо петљу пре него што се изврше све итерације
- У тој ситуацији користимо наредбу **break** која моментално завршава актуелну петљу
- У ситуацији када желимо да наставимо са следећом итерацијом петље без њеног прекидања користимо наредбу **continue**

Пример (8)

- Написати програм који учитава бројеве од стране корисника све док се не унесе број 0. Након тога на излазу исписује два узастопна елемента који су дали највећи збир.

```
import math
l = []
while (True):
        x = int(input())
        if (x==0):
                break # aktuelna petlja se zavrsava
        l.append(x)
maks2 = -math.inf
for i in range(1,len(l)):
        if (l[i-1]+l[i]>maks2):
                maks2 = l[i-1]+l[i]
print("Maksimum je %d"%(maks2))
```

Пример (9)

- Написати програм који за све бројеве у интервалу од 1 до N проверава да ли су прости и ако јесу, исписује их. N задаје корисник.

Задаци

- Написати програме који исписују следеће цртеже. Димензију N цртежа задаје корисник. У примерима је N=3.

а)
```
***
**
*
```

ц)
```
  *
 ***
*****
```

б)
```
*
* *
* * *
```

д)
```
  *
 * *
* * *
```

Вишеструке листе

- Листа може унутар себе садржати друге листе
- Један типичан пример вишеструке листе је матрица
- Код ње се додатно подразумева да је свака подлиста исте димензије
- На пример: A = [[1,2,3],[4,5,6]] је матрица димензије 2 x 3
- Дводимензиона листа не мора бити нужно и матрица:
  B = [[1,3],[4,5,6],[1,5,3]]

Пример (11)

- Написати програм који сабира две матрице уколико су одговарајућих димензија, у супротном исписује поруку о грешци

Пример (11)

- Нека је задата листа речи. Пронаћи и исписати све речи које у себи садрже текст који задаје корисник на улазу. На пример, ако је листа [“Mi”, “uzivamo”, “dok”, “programiramo”] и текст „mo“ резултат су речи „uzivamo“ и „programiramo“

Пример (13)

- Написати програм који множи две матрице ако имају одговарајуће димензије, у супротном исписује поруку о грешци

Задаци

- Фибоначијев низ бројева је 1, 1, 2, 3, 5, 8, 13, 21, ... је дефинисан условима $f_0=1$, $f_1=1$, $f_{n+2}=f_{n+1}+f_n$. Написати програм који исписује првих k чланова овог низа.
- Написати програм који исписује све цифре унетог позитивног целог броја почевши од цифре јединица. На пример за број 4542, резултат је 2 4 5 2. Издвајање цифара урадити **аритметички**, а не текстуално!
- Написати програм који **аритметички** надовезује два унета позитивна цела броја. На пример за бројеве 143 и 3431 формира с еброј 1433431.

Задаци (2)

- Још задатак за вежбу из уводних области се могу наћи на адресама:
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/5/g2/zadaci.pdf
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/6/zadaci.pdf
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/7/zadaci.pdf
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/8/zadaci.pdf
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/9/zadaci1.pdf
- http://poincare.matf.bg.ac.rs/~kartelj/nastava/P12014/9/zadaci2.pdf