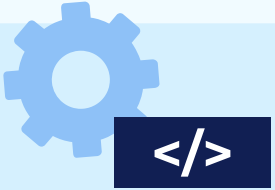
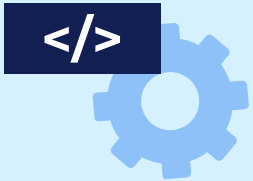
UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET



TESTIRANJE BAZIRANO NA PROGRAMSKIM PUTANJAMA

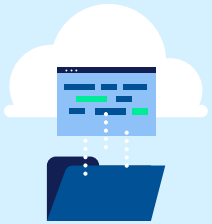


TESTIRANJE SOFTVERA



01

PROGRAMSKE PUTANJE

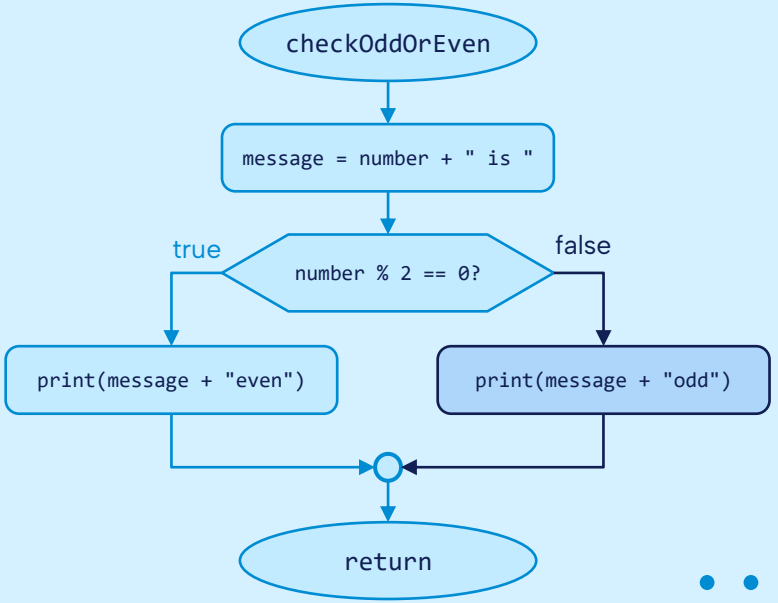


PROGRAMSKE PUTANJE

- Programska putanja predstavlja putanju u grafu toka kontrole od ulaznog čvora do nekog izlaznog čvora.
- Jedna putanja je definisana skupom čvorova i grana u grafu toka kontrole.

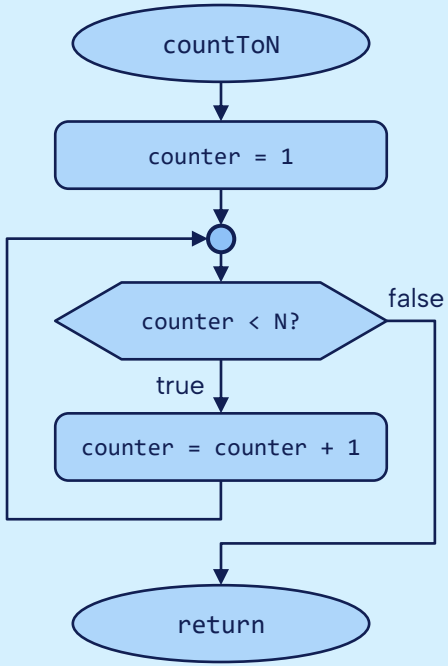
```
public static void checkOddOrEven(int number) {
    String message = number + " is ";

    if (number % 2 == 0) {
        System.out.println(message + "even");
    } else {
        System.out.println(message + "odd");
    }
    return;
}
```

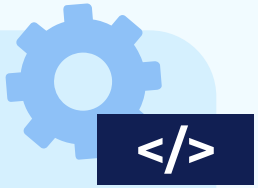


BAZIČNE PROGRAMSKE PUTANJE

- Kriterijum pokrivenosti **svih putanja** u programu nije praktičan, pogotovo ukoliko program sadrži petlje sa promenljivim brojem iteracija.
- Kriterijum pokrivenosti **bazičnih putanja** uzima reprezentativni podskup putanja tako da svaka odabrana putanja predstavlja novu dimenziju kontrole toka.
- Bazične putanje su linearno nezavisne, odnosno ne mogu se dobiti linearnom kombinacijom drugih bazičnih putanja.
- Svaka druga putanja u programu se može dobiti kao linearna kombinacija bazičnih putanja.



CIKLOMATSKA KOMPLEKSNOST

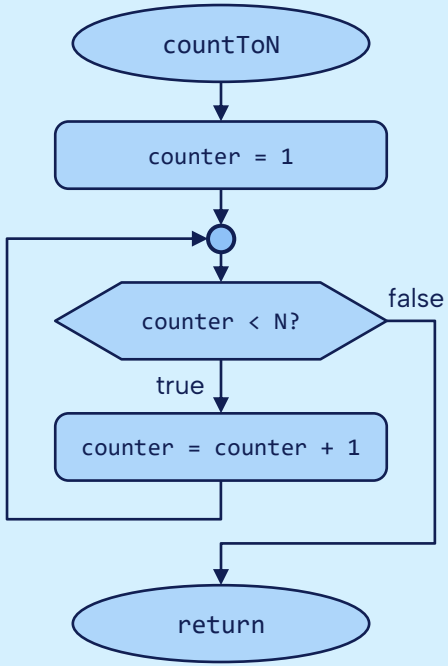


- Ciklomatska kompleksnost, u oznaci **V(G)**, definiše broj bazičnih putanja u grafu toka kontrole.
- Za grafove sa jednom ulaznom i jednom izlaznom tačkom:
  - **V(G) = e – n + 2**, gde je **e** broj grana, a **n** broj čvorova u grafu.
  - **V(G) = b + 1**, gde je **b** broj binarnih odluka u grafu.
- Za grafove sa više od jedne ulazne ili više od jedne izlazne tačke:
  - Prvo se graf modifikuje tako da se svaki izlazni čvor poveže sa svakim ulaznim čvorom.
  - **V(G) = e’ – n + 1**, gde je **e’** broj grana u modifikovanom grafu, a **n** broj čvorova u grafu.



BAZIČNI SKUP PUTANJA

- Bazični skup putanja je skup linearno nezavisnih putanja u programu.
- *McCabe*-ov metod za određivanje bazičnog skupa putanja:
    1. Bira se osnovna putanja na proizvoljan način, ali tako da ona pokriva što više čvorova.
    2. Prati se osnovna putanja i za svaku nepokrivenu izlaznu granu bilo koje odluke u putanji se dodaje jedna nova putanja sa tom izlaznom granom u skup.
    3. Postupak se ponavlja za sve dodate putanje, dok se ne dobije ukupno V(G) bazičnih putanja.



ZADATAK 1 – OCENE NA ISPITU

POSTAVKA (1)

- Data je implementacija funkcije **repeat_exam** u programskom jeziku Java koja na osnovu ocena studenata na ispitu određuje da li je ispit neophodno ponoviti ili ne.
- Funkcija sa standardnog ulaza učitava tačno 10 celih brojeva koji predstavljaju ocene studenata na ispitu. Ukoliko je barem pola studenata položilo ispit, ne treba ga ponavljati. Funkcija vraća logičku vrednost koja određuje da li je ispit neophodno ponoviti.

```
  public static boolean repeat_exam() {
1.    Scanner sc = new Scanner(System.in);
2.    int passed=0, failed=0, cnt=1, grade;
3.    boolean repeat = false;
4.    while (cnt <= 10) {
5.      grade = sc.nextInt();
6.      if (grade > 5) passed++;
7.      else failed++;
8.      cnt++;
9.    }
10.   sc.close();
11.   System.out.println("Pass: " + passed);
12.   System.out.println("Fail: " + failed);
13.   if (passed < 5) repeat = true;
14.   return repeat;
15. }
```

ZADATAK 1 – OCENE NA ISPITU

POSTAVKA (2)

1. Nacrtati graf toka kontrole za datu funkciju.
2. Odrediti ciklomatsku kompleksnost funkcije.
3. Odrediti bazični skup putanja u funkciji.
4. Napisati minimalan skup test primera kojim će biti pokrivene sve bazične putanje iz prethodne tačke. Za svaki test primer navesti ulazne podatke, očekivani izlaz i putanje koje su pokrivene test primerom.

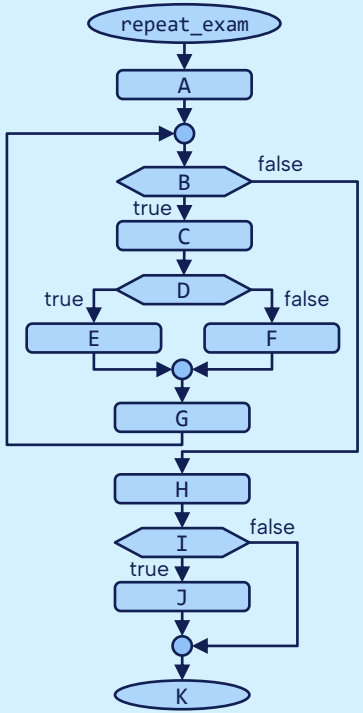


ZADATAK 1 – OCENE NA ISPITU

REŠENJE – GRAF TOKA KONTROLE

```
public static boolean repeat_exam() {
1.    Scanner sc = new Scanner(System.in);
2.    int passed=0, failed=0, cnt=1, grade;
3.    boolean repeat = false;
4.    while (cnt <= 10) {
5.      grade = sc.nextInt();
6.      if (grade > 5) passed++;
7.      else failed++;
8.      cnt++;
9.    }
10.   sc.close();
11.   System.out.println("Pass: " + passed);
12.   System.out.println("Fail: " + failed);
13.   if (passed < 5) repeat = true;
14.   return repeat;
15. }
```

A
B
C
D, E
F
G
H
I, J
K



ZADATAK 1 – OCENE NA ISPITU

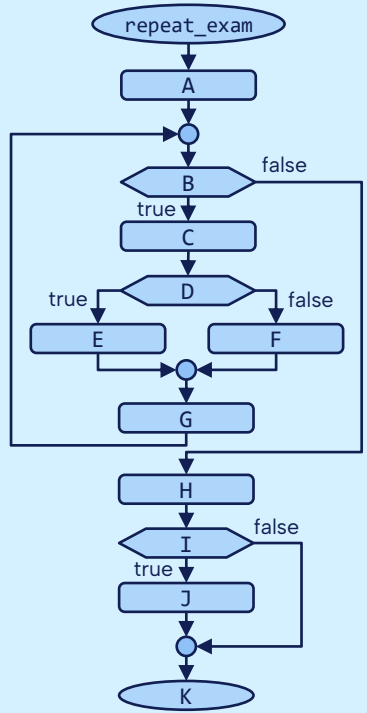
REŠENJE – CIKLOMATSKA KOMPLEKSNOST

- Graf toka kontrole ima tačno jednu ulaznu i jednu izlaznu tačku.
- Ovde se razmatra varijanta grafa sa čvorovima spajanja kao odvojenim čvorovima.
  - Broj čvorova u grafu (n) je 15.
  - Broj grana u grafu (e) je 17.
  - Broj binarnih odluka u grafu (b) je 3.
- Ciklomatska kompleksnost funkcije je:

$$V(G) = e - n + 2 = 4$$

ili

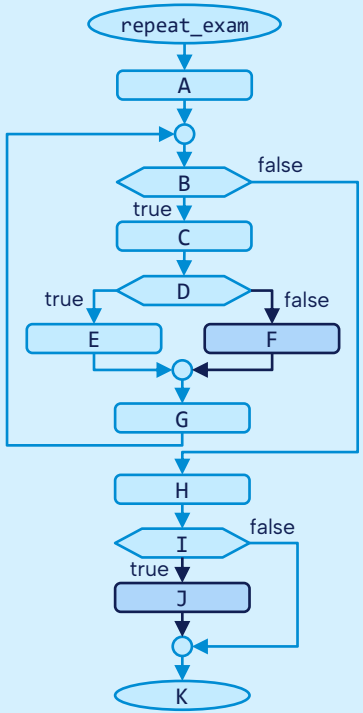
$$V(G) = b + 1 = 4$$



ZADATAK 1 – OCENE NA ISPITU

REŠENJE – BAZIČNE PUTANJE (1)

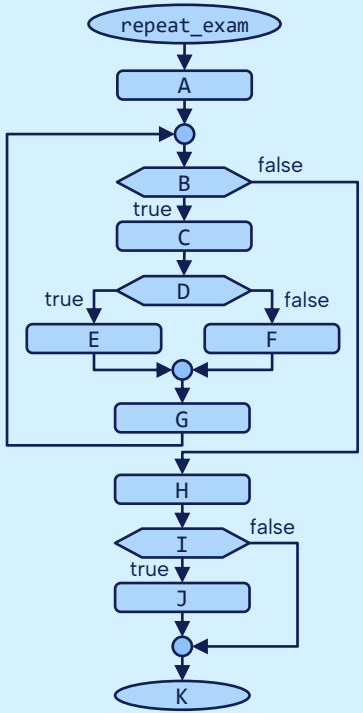
- Bira se osnovna putanja u grafu:
  - **P1:** repeat_exam–A–B–C–D–E–G–B–H–I–K
- Dalje, na osnovnoj putanji se traži prva nepokrivena izlazna grana neke odluke.
  - To je *false* grana odluke B.
  - Dakle, dodaje se nova putanja modifikacijom osnovne putanje, tako da uključi ovu izlaznu granu.
  - **P2:** repeat_exam–A–B–H–I–K



ZADATAK 1 – OCENE NA ISPITU

REŠENJE – BAZIČNE PUTANJE (2)

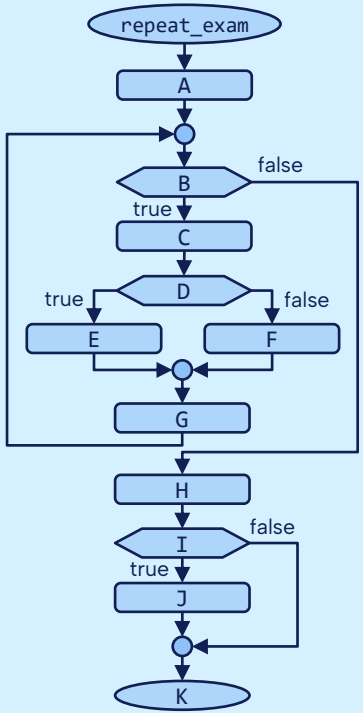
- Trenutni skup putanja čine:
  - **P1:** repeat_exam–A–B–C–D–E–G–B–H–I-K
  - **P2:** repeat_exam–A–B–H–I-K
- Sledeća nepokrivena grana je *false* grana odluke D, pa se modifikuje prva putanja da se dobije nova:
  - **P3:** repeat_exam–A–B–C–D–F–G–B–H–I-K
- Poslednja nepokrivena grana je *true* grana odluke I, pa se opet modifikuje prva putanja da se dobije nova:
  - **P4:** repeat_exam–A–B–C–D–E–G–B–H–I–J-K



ZADATAK 1 – OCENE NA ISPITU

REŠENJE – BAZIČNE PUTANJE (3)

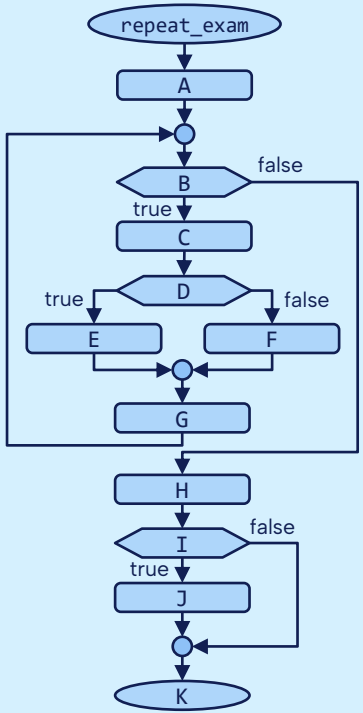
- Konačan bazični skup putanja čine:
  - **P1:** repeat_exam–A–B–C–D–E–G–B–H–I-K
  - **P2:** repeat_exam–A–B–H–I-K
  - **P3:** repeat_exam–A–B–C–D–F–G–B–H–I-K
  - **P4:** repeat_exam–A–B–C–D–E–G–B–H–I–J-K
- Putanje pokazuju samo jednu iteraciju petlje, ali se telo petlje uvek ponavlja 10 puta.
- Koji uslovi moraju biti zadovoljeni da bi se svaka putanja izvršila?
- Da li postoji neka putanja koja nije izvršiva?



ZADATAK 1 – OCENE NA ISPITU

REŠENJE – TEST PRIMERI (1)

- Putanja **P1** se izvršava ukoliko postoji barem jedan student koji je položio ispit i ukoliko ispit ne treba da se ponavlja.
- Putanja **P2** nije izvršiva jer podrazumeva da se u telo petlje ne uđe nijednom, ali to nije moguće jer je broj iteracija fiksiran na 10.
- Putanja **P3** se izvršava ukoliko postoji barem jedan student koji nije položio ispit i ukoliko ispit ne treba da se ponavlja.
- Putanja **P4** se izvršava ukoliko postoji bar jedan student koji je položio ispit i ukoliko ispit treba da se ponavlja.

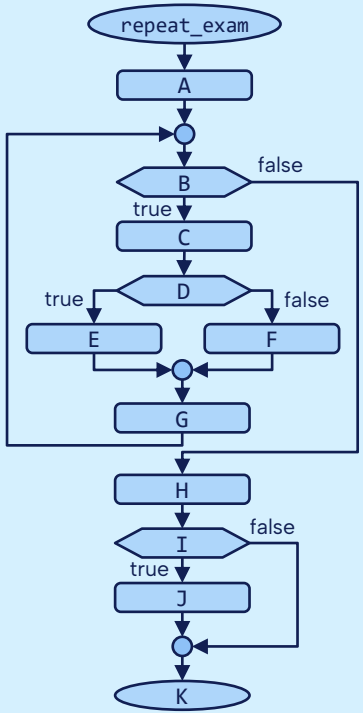


ZADATAK 1 – OCENE NA ISPITU

REŠENJE – TEST PRIMERI (2)

- Test primeri koji pokrivaju izvršive bazične putanje:

TP	Ulazni podaci	Očekivani izlaz	Povratna vrednost	Pokrivene putanje
1	8, 10, 7, 10, 9, 10, 10, 9, 6, 10	Pass: 10 Fail: 0	False	P1
2	5, 5, 5, 10, 8, 9, 10, 7, 8, 6	Pass: 7 Fail: 3	False	P3
3	6, 6, 5, 5, 5, 5, 5, 5, 5, 5	Pass: 2 Fail: 8	True	P4



ZADATAK 2 – KLASIFIKACIJA BROJA

POSTAVKA (1)

- Data je implementacija funkcije **classify** u programskom jeziku Java koja klasifikuje prosleđeni ceo broj **x** u neku od sledećih klasa:
  - „negativan“, ako je $x < 0$
  - „nula“, ako je $x = 0$
  - „pozitivan manji od 5“, ako je $0 < x < 5$
  - „pozitivan manji od 10“, ako je $5 \leq x < 10$
  - „jednak 10“, ako je $x = 10$
  - „pozitivan veci od 10“, ako je $x > 10$
- Povratna vrednost funkcije je string koji označava klasu kojoj broj pripada.

```
  public static String classify(int x) {
1.    if (x < 0) return "negativan";
2.    if (x == 0) return "nula";
3.    else if (x < 10)
4.      return "pozitivan manji od 10";
5.    else {
6.      if (x == 10) return "jednak 10";
7.      if (x < 5)
8.        return "pozitivan manji od 5";
9.     }
10.    return "pozitivan veci od 10";
11.}
```

ZADATAK 2 – KLASIFIKACIJA BROJA

POSTAVKA (2)

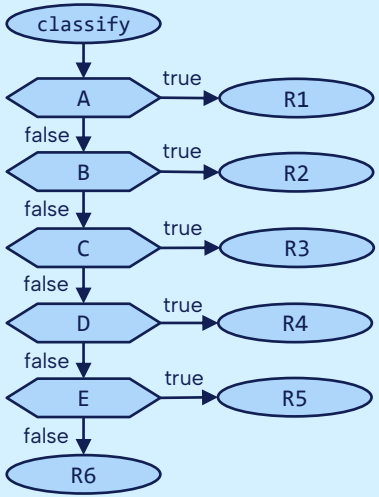
1. Nacrtati graf toka kontrole za datu funkciju.
2. Odrediti ciklomatsku kompleksnost funkcije.
3. Odrediti bazični skup putanja u funkciji.
4. Napisati minimalan skup test primera kojim će biti pokrivene sve izvršive bazične putanje. Za svaki test primer navesti ulazne podatke, očekivani izlaz i putanje koje su pokrivene test primerom.
5. Ukoliko postoje putanje koje nije moguće izvršiti, obrazložiti zašto ih nije moguće izvršiti i odrediti defektni segment koda u funkciji koji to uzrokuje.



ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – GRAF TOKA KONTROLE

```
  public static String classify(int x) {
1.    if (x < 0) return "negativan";
2.    if (x == 0) return "nula";
3.    else if (x < 10)
4.      return "pozitivan manji od 10";
5.    else {
6.      if (x == 10) return "jednak 10";
7.      if (x < 5)
8.        return "pozitivan manji od 5";
9.     }
10.    return "pozitivan veci od 10";
11. }
```

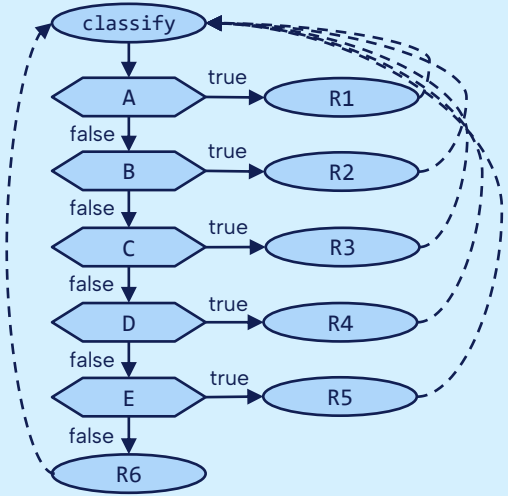


ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – CIKLOMATSKA KOMPLEKSNOST

- Graf toka kontrole ima tačno jednu ulaznu, ali više mogućih izlaznih tačaka.
  - Svaki izlazni čvor mora da se spoji sa ulaznim čvorom.
- Za modifikovani graf toka kontrole važi:
  - Broj čvorova u grafu (n) je 12.
  - Broj grana u grafu (e') je 17.
- Ciklomatska kompleksnost funkcije je:

$$V(G) = e' - n + 1 = 6$$

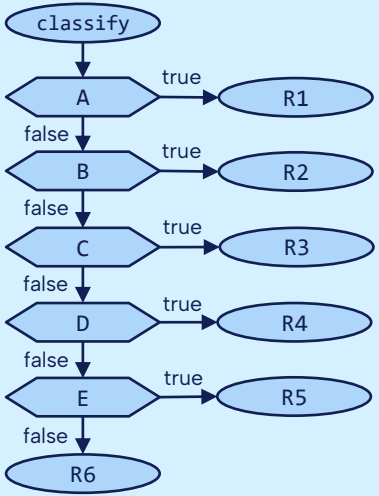


ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – BAZIČNE PUTANJE (1)

- Bazični skup putanja je:
  - **P1:** classify – A – B – C – D – E – R6
  - **P2:** classify – A – R1
  - **P3:** classify – A – B – R2
  - **P4:** classify – A – B – C – R3
  - **P5:** classify – A – B – C – D – R4
  - **P6:** classify – A – B – C – D – E – R5

- Da li postoji putanja koju nije moguće izvršiti?

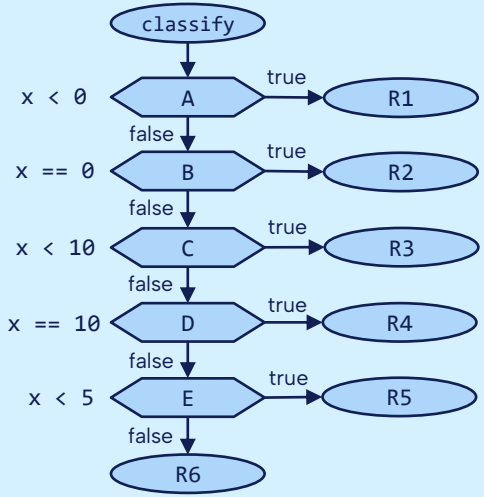


ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – BAZIČNE PUTANJE (2)

- Uslovi za izvršavanje putanja:

P	Putanja	Pojedinačni uslovi
1	classify – A – B – C – D – E – R6	x >= 0, x != 0, x >= 10, x != 10, x >= 5
2	classify – A – R1	x < 0
3	classify – A – B – R2	x >= 0, x == 0
4	classify – A – B – C – R3	x >= 0, x != 0, x < 10
5	classify – A – B – C – D – R4	x >= 0, x != 0, x >= 10, x == 10
6	A – B – C – D – E – R5	x >= 0, x != 0, x >= 10, x != 10, x < 5



ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – BAZIČNE PUTANJE (3)

- Uslovi za izvršavanje putanja:

P	Putanja	Pojedinačni uslovi	Sveobuhvatan uslov
1	classify – A – B – C – D – E – R6	x >= 0, x != 0, x >= 10, x != 10, x >= 5	x > 10
2	classify – A – R1	x < 0	x < 0
3	classify – A – B – R2	x >= 0, x == 0	x == 0
4	classify – A – B – C – R3	x >= 0, x != 0, x < 10	0 < x < 10
5	classify – A – B – C – D – R4	x >= 0, x != 0, x >= 10, x == 10	x == 10
6	classify – A – B – C – D – E – R5	x >= 0, x != 0, x >= 10, x != 10, x < 5	**x > 10, x < 5**

ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – TEST PRIMERI

- Test primeri koji pokrivaju izvršive bazične putanje:

TP	x	Očekivani izlaz	Pokrivene putanje
1	17	pozitivan veci od 10	P1
2	–3	negativan	P2
3	0	nula	P3
4	7	pozitivan manji od 10	P4
5	10	jednak 10	P5

```
  public static String classify(int x) {
1.    if (x < 0) return "negativan";
2.    if (x == 0) return "nula";
3.    else if (x < 10)
4.      return "pozitivan manji od 10";
5.    else {
6.      if (x == 10) return "jednak 10";
7.      if (x < 5)
8.        return "pozitivan manji od 5";
9.     }
10.    return "pozitivan veci od 10";
11. }
```



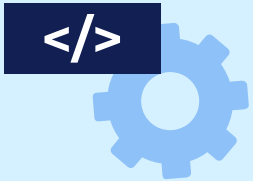
ZADATAK 2 – KLASIFIKACIJA BROJA

REŠENJE – DEFEKTNI KOD

- Putanja P6 nije izvršiva zbog uslova koji su kontradiktorni, a koji su neophodni za njeno izvršavanje.
- Da bi putanja bila izvršiva, mora se eliminisati uslov **x > 10**, odnosno proveru uslova **x < 5** (čvor E) treba premestiti iznad provere uslova **x < 10** (čvora C).

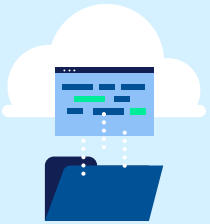
P	Putanja	Uslov
1	classify – A – B – C – D – E – R6	x > 10
2	classify – A – R1	x < 0
3	classify – A – B – R2	x == 0
4	classify – A – B – C – R3	0 < x < 10
5	classify – A – B – C – D – R4	x == 10
6	classify – A – B – C – D – E – R5	**x > 10, x < 5**

```
  public static String classify(int x) {
1.    if (x < 0) return "negativan";
2.    if (x == 0) return "nula";
3.    else if (x < 10)
4.      return "pozitivan manji od 10";
5.    else {
6.      if (x == 10) return "jednak 10";
7.      if (x < 5)
8.        return "pozitivan manji od 5";
9.     }
10.    return "pozitivan veci od 10";
11. }
```

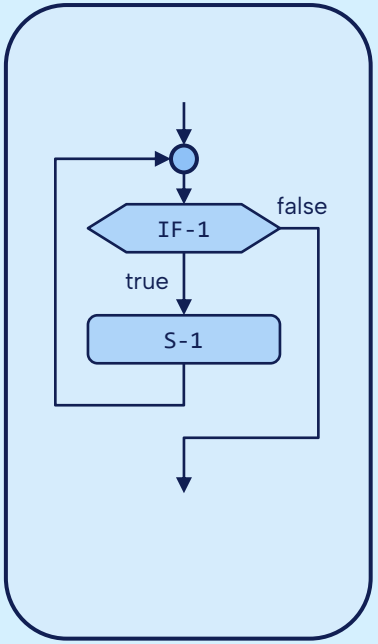


02

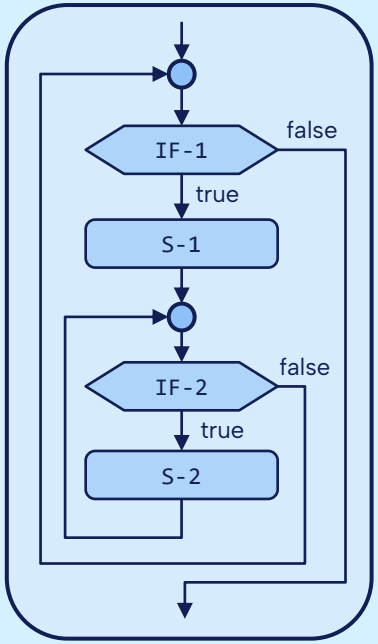
TESTIRANJE PETLJI



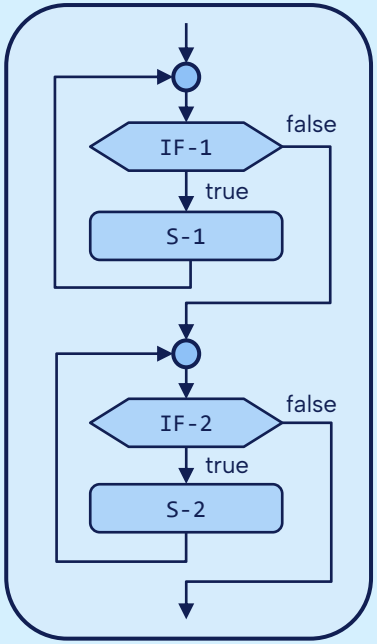
TIPOVI PETLJI



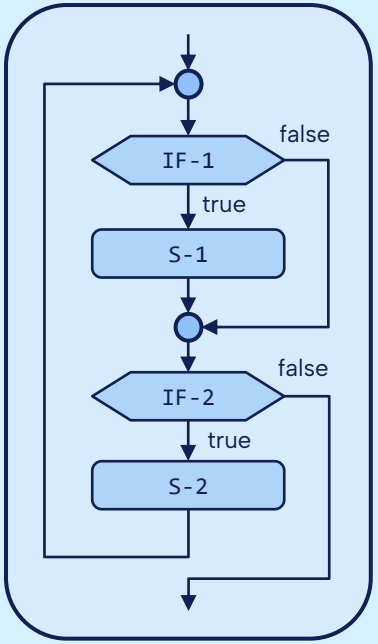
Jednostruka petlja



Ugnežđena petlja



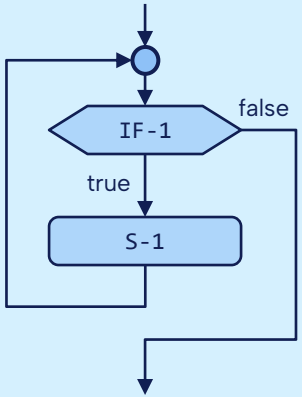
Nadovezana petlja



Nestrukturirana petlja

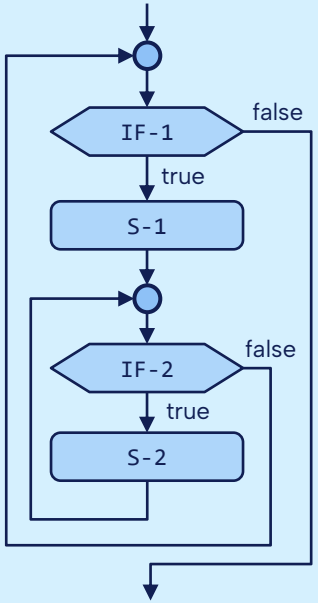
TESTIRANJE JEDNOSTRUKIH PETLJI

- Bajzerova metoda podrazumeva da se za jednostruku petlju koja je dizajnirana da se izvršava maksimalno **n** puta, testiraju sledeći slučajevi:
  - Petlja se ne izvršava **(0 iteracija)**
  - Petlja se izvršava tačno jednom **(1 iteracija)**
  - Petlja se izvršava prosečan broj puta **(m iteracija)**
  - Petlja se izvršava za jednu iteraciju manje od maksimalnog predviđenog broja **(n – 1 iteracija)**
  - Petlja se izvršava tačno onoliko puta koliko je maksimalno predviđeno **(n iteracija)**
  - Petlja se izvršava za jednu iteraciju više od maksimalnog predviđenog broja **(n + 1 iteracija)**



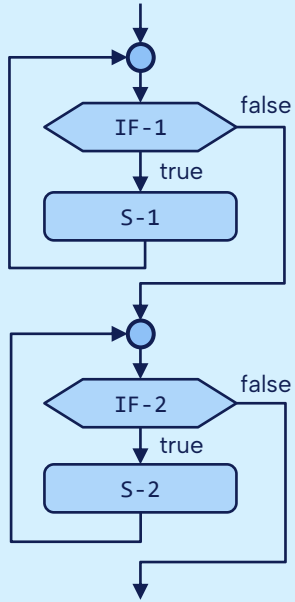
TESTIRANJE UGNEŽDJENIH PETLJI

- Ugnežđena petlja se testira na sledeći način:
  - Prvo se testira najugnežđenija petlja pravilima za testiranje jednostruke petlje, tako da se sve okružujuće petlje izvršavaju minimalan broj puta.
  - Onda se testira njena prva okružujuća petlja, ponovo pravilima za testiranje jednostruke petlje, tako da se sve njene okružujuće petlje izvršavaju minimalan broj puta, a njena ugneždjena petlja prosečan broj puta.
  - Postupak se nastavlja kretanjem kroz okružujuće petlje dok se ne dođe do spoljašnje petlje.



TESTIRANJE NADOVEZANIH PETLJI

- Nadovezane petlje mogu da budu zavisne ili nezavisne.
- **Nezavisne nadovezane petlje** se testiraju individualno, pravilima za testiranje jednostrukih petlji.
- **Zavisne nadovezane petlje** se testiraju na sličan način kao ugnežđene, kako bi se ispitala njihova međuzavisnost.



ZADATAK 3 – BUBBLE SORT

POSTAVKA

- Data je funkcija u pseudokodu koja sortira niz celih brojeva korišćenjem *Bubble Sort* algoritma. Funkcija ima dva parametra, a to su niz celih brojeva **A** i dužina niza **n**.

1. Nacrtati graf toka kontrole za datu funkciju.
2. Odrediti skup test primera kojim će se testirati petlje u funkciji korišćenjem Bajzerove metode. Za svaki test primer navesti ulazne podatke i očekivani izlaz. Obrazložiti izbor.

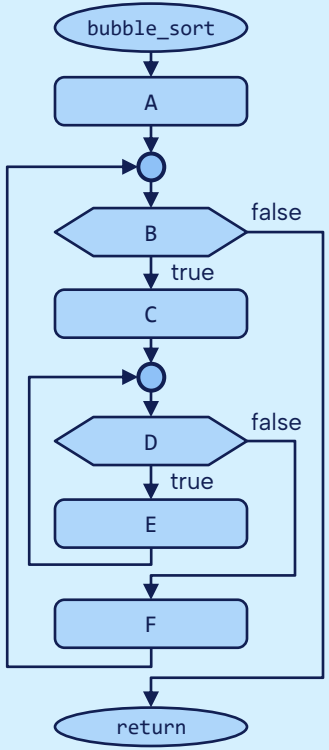
```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.      j = i - 1
4.      while (j >= 1 and A[j] > A[j+1]) do
5.         tmp = A[j]
6.         A[j] = A[j+1]
7.         A[j+1] = tmp
8.         j = j - 1
9.      end_while
10.     i = i + 1
11.  end_while
```



ZADATAK 3 – BUBBLE SORT

REŠENJE – GRAF TOKA KONTROLE

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.      j = i - 1
4.      while (j >= 1 and A[j] > A[j+1]) do
5.         tmp = A[j]
6.         A[j] = A[j+1]
7.         A[j+1] = tmp
8.         j = j – 1
9.      end_while
10.     i = i + 1
11.  end_while
```



ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (1)

- Broj iteracija spoljašnje petlje zavisi samo od dužine niza.
  - Za **n > 2**, telo spoljašnje petlje se izvršava tačno **n – 1** puta.
  - Za **n <= 1**, spoljašnja petlja se ne izvršava nijednom.
- Broj iteracija unutrašnje petlje zavisi od brojača okružujuće petlje **i**.
  - Ona može da se izvrši maksimalno **i – 1** puta i to samo ako je uvek zadovoljen uslov **A[j] > A[j+1]** (ako je element uvek veći od onog iza njega).
  - Za poslednju iteraciju spoljašnje petlje, kada je **i = n**, unutrašnja petlja može da ima maksimalno **n – 1** iteracija.
  - Unutrašnja petlja može da se izvrši minimalno **0** puta i to samo ako za prvi par elemenata koji se upoređuje nije zadovoljen uslov, odnosno ako element nije veći od onog iza njega.

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (2)

- Prvo se testira ugnežđena petlja, tako da se broj iteracija okružujuće petlje drži na minimumu:
  - Unutrašnja petlja će se izvršiti 0 puta ukoliko je niz već uređen rastuće. Spoljašnja petlja mora da se izvrši barem jednom, odnosno niz mora da ima barem 2 elementa.
  - Unutrašnja petlja će se izvršiti tačno 1 put ukoliko je potrebna samo jedna zamena elemenata. Slično, da bi se unutrašnja petlja uopšte izvršila, niz mora da ima barem 2 elementa.

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.     j = i - 1
4.     while (j>=1 and A[j]>A[j+1]) do
5.        tmp = A[j]
6.        A[j] = A[j+1]
7.        A[j+1] = tmp
8.        j = j – 1
9.     end_while
10.    i = i + 1
11.  end_while
```



ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (3)

- Maksimalan i prosečan broj iteracija nisu definisani u funkciji i specifikaciji. Dakle, možemo da uzmemo proizvoljnu pretpostavku, npr. da je predviđena maksimalna dužina niza 100.
  - To bi značilo da je maksimalan broj iteracija i spoljašnje i unutrašnje petlje 99.
  - U jednoj iteraciji spoljašnje petlje, u proseku je potrebno pomeriti pola elemenata niza, pa bi prosečan broj iteracija unutrašnje petlje bio 50.
  - Dakle, treba testirati izvršavanje 50, 98, 99 i 100 iteracija unutrašnje petlje.

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.     j = i - 1
4.     while (j>=1 and A[j]>A[j+1]) do
5.        tmp = A[j]
6.        A[j] = A[j+1]
7.        A[j+1] = tmp
8.        j = j – 1
9.     end_while
10.    i = i + 1
11.  end_while
```

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (4)

- Pošto broj iteracija unutrašnje petlje direktno zavisi od broja iteracija spoljašnje petlje, minimalan broj iteracija spoljašnje je jednak broju iteracija unutrašnje petlje.
  - Za 50 iteracija unutrašnje petlje, potreban je niz od 51 element, tako da je u poslednjoj iteraciji spoljašnje petlje potrebno pomeriti 50 elemenata.
  - Slično i za 98, 99 i 100 iteracija, potrebni su nizovi dužine 99, 100 i 101, respektivno. U poslednjoj iteraciji spoljašnje petlje potrebno je pomeriti 98, 99 i 100 elemenata, respektivno.
  - Ovo može da se postigne tako što će poslednji element u nizu zapravo biti i najmanji element, pa on mora da se pomeri na početak niza.

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.     j = i - 1
4.     while (j>=1 and A[j]>A[j+1]) do
5.       tmp = A[j]
6.       A[j] = A[j+1]
7.       A[j+1] = tmp
8.       j = j - 1
9.     end_while
10.    i = i + 1
11.  end_while
```

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (5)

- Test primeri za unutrašnju petlju:

TP	A	n	Očekivani izlaz	Pokrivene iteracije
1	{ 1, 2 }	2	{ 1, 2 }	0 iteracija unutrašnje petlje
2	{ 2, 1 }	2	{ 1, 2 }	1 iteracija unutrašnje petlje
3	{ 1, 2, ..., 50, 0 }	51	{ 0, 1, 2, ..., 50 }	prosečan broj iteracija unutrašnje petlje
4	{ 1, 2, ..., 98, 0 }	99	{ 0, 1, 2, ..., 98 }	ispod maksimalnog broja iteracija unutrašnje petlje
5	{ 1, 2, ..., 99, 0 }	100	{ 0, 1, 2, ..., 99 }	maksimalan broj iteracija unutrašnje petlje
6	{ 1, 2, ..., 100, 0 }	101	{ 0, 1, 2, ..., 100 }	iznad maksimalnog broja iteracija unutrašnje petlje



ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (6)

- Dalje, testira se okružujuća petlja tako da se broj iteracija ugnežđene petlje drži na proseku. Uzeta je pretpostavka da je maksimalna dužina niza 100, pa je maksimalan broj iteracija spoljašnje petlje 99. Dakle, potrebno je testirati sledeće situacije:
  - Spoljašnja petlja se izvršava 0 puta.
  - Spoljašnja petlja se izvršava 1 put.
  - Spoljašnja petlja se izvršava 50 puta.
  - Spoljašnja petlja se izvršava 98 puta.
  - Spoljašnja petlja se izvršava 99 puta.
  - Spoljašnja petlja se izvršava 100 puta.

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.     j = i - 1
4.     while (j>=1 and A[j]>A[j+1]) do
5.        tmp = A[j]
6.        A[j] = A[j+1]
7.        A[j+1] = tmp
8.        j = j - 1
9.     end_while
10.    i = i + 1
11.  end_while
```

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (7)

- Test primerima za unutrašnju petlju već je pokriveno izvršavanje 1, 50, 98, 99 i 100 iteracija spoljašnje petlje. Preostao je jedan slučaj:
  - Spoljašnja petlja se izvršava 0 puta, i to ukoliko niz ima samo jedan element.
  - Potrebno je dodati još jedan test primer.

```
BUBBLE_SORT(A, n)
1.   i = 2
2.   while (i <= n) do
3.     j = i - 1
4.     while (j>=1 and A[j]>A[j+1]) do
5.        tmp = A[j]
6.        A[j] = A[j+1]
7.        A[j+1] = tmp
8.        j = j – 1
9.     end_while
10.    i = i + 1
11.  end_while
```

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (8)

- Test primeri za sve petlje:

TP	A	n	Očekivani izlaz
1	{ 1, 2 }	2	{ 1, 2 }
2	{ 2, 1 }	2	{ 1, 2 }
3	{ 1, 2, ..., 50, 0 }	51	{ 0, 1, 2, ..., 50 }
4	{ 1, 2, ..., 98, 0 }	99	{ 0, 1, 2, ..., 98 }
5	{ 1, 2, ..., 99, 0 }	100	{ 0, 1, 2, ..., 99 }
6	{ 1, 2, ..., 100, 0 }	101	{ 0, 1, 2, ..., 100 }
7	{ 1 }	1	{ 1 }

- Da li postoji problem sa ovim skupom test primera?

ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (9)

- Za testiranje spoljašnje petlje iskorišćeni su testovi za unutrašnju petlju. Međutim, ti testovi su koristili uređene sekvence brojeva, osim poslednjeg elementa koji je trebalo pomeriti na početak niza.
- Problem je u tome što testiranje spoljašnje petlje podrazumeva da broj iteracija unutrašnje petlje bude prosečan.
    - U navedenim test primerima, samo se poslednji element pomera na početak, odnosno kada se unutrašnja petlja izvršava, izvršava se maksimalan broj puta.
    - Dakle, test primeri za dužine niza 51, 99, 100 i 101 se moraju izmeniti tako da prvih **n – 1** elemenata niza bude izmešano, da bi se obezbedio prosečan broj zamena.
    - Svakako, najmanji element niza treba da ostane na kraju kako bi se obezbedila i poslednja iteracija spoljašnje petlje kada se izvršava maksimalan broj pomeranja u unutrašnjoj petlji.



ZADATAK 3 – BUBBLE SORT

REŠENJE – BAJZEROVA METODA (10)

- Konačni skup test primera za sve petlje:

TP	A	n	Očekivani izlaz
1	{ 1 }	1	{ 1 }
2	{ 1, 2 }	2	{ 1, 2 }
3	{ 2, 1 }	2	{ 1, 2 }
4	{ 1, 2, 1, 2, ..., 1, 2, 0 }	51	{ 0, 1, 1, ..., 1, 2, 2, ..., 2}
5	{ 1, 2, 1, 2, ..., 1, 2, 0 }	99	{ 0, 1, 1, ..., 1, 2, 2, ..., 2}
6	{ 1, 2, 1, 2, ..., 1, 2, 0 }	100	{ 0, 1, 1, ..., 1, 2, 2, ..., 2}
7	{ 1, 2, 1, 2, ..., 1, 2, 0 }	101	{ 0, 1, 1, ..., 1, 2, 2, ..., 2}

ZADATAK 4 – POZITIVNI BROJEVI

POSTAVKA

- Data je implementacija funkcije **count** u programskom jeziku Java koja određuje broj pozitivnih elemenata u prosleđenom nizu i na osnovu njega modifikuje sam niz.

1. Nacrtati graf toka kontrole za datu funkciju.
2. Odrediti skup test primera kojim će se testirati petlje u funkciji korišćenjem Bajzerove metode. Za svaki test primer navesti ulazne podatke i očekivani izlaz. Obrazložiti izbor.

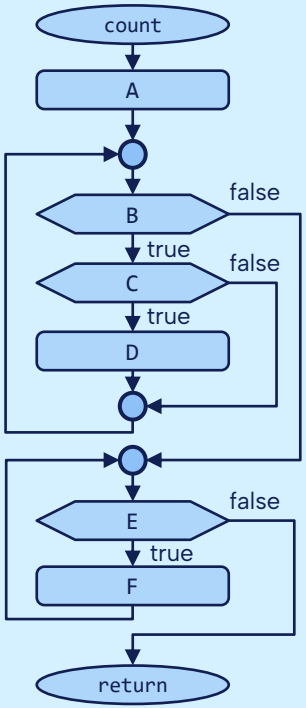
```
  public static void count(int[] a) {
1.    int count = 0;
2.    int n = a.length;
3.    for (int i = 0; i < n; i++) {
4.        if (a[i] > 0) count++;
5.    }
6.    for (int i = 0; i < count; i++) {
7.        a[i] += count;
8.    }
9. }
```



ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – GRAF TOKA KONTROLE

```
  public static void count(int[] a) {
1.    int count = 0;
2.    int n = a.length;
3.    for (int i = 0; i < n; i++) {
4.        if (a[i] > 0) count++;
5.    }
6.    for (int i = 0; i < count; i++) {
7.        a[i] += count;
8.    }
9. }
```



ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – BAJZEROVA METODA (1)

- Funkcija sadrži dve nadovezane petlje koje su zavisne, pošto broj iteracija druge petlje direktno zavisi od broja pozitivnih elemenata u nizu, koji se određuje u prvoj petlji.
- Prvo se testira druga (zavisna) petlja. Broj iteracija ove petlje je jednak broju pozitivnih elemenata u nizu.
  - Dakle, maksimalan broj iteracija ove petlje jednak je dužini niza **n**. Ovo važi ukoliko su svi elementi niza pozitivni.
  - Prosečan broj pozitivnih elemenata je jednak polovini veličine niza.

```
  public static void count(int[] a) {
1.    int count = 0;
2.    int n = a.length;
3.    for (int i = 0; i < n; i++) {
4.        if (a[i] > 0) count++;
5.    }
6.    for (int i = 0; i < count; i++) {
7.        a[i] += count;
8.    }
9. }
```

ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – BAJZEROVA METODA (2)

- Dakle, za drugu petlju, potrebno je testirati sledeće situacije:
  - Druga petlja se izvršava **0** puta.
  - Druga petlja se izvršava **1** put.
  - Druga petlja se izvršava **n / 2** puta.
  - Druga petlja se izvršava **n – 1** puta.
  - Druga petlja se izvršava **n** puta.
  - Druga petlja se izvršava **n + 1** puta.
- Ukoliko je ovu petlju potrebno izvršiti **k** puta, potrebno je da niz sadrži **k** pozitivnih elemenata.
- Ovu petlju nije moguće izvršiti **n + 1** puta, obzirom da je niz ograničen na **n** elemenata.



ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – BAJZEROVA METODA (3)

- Prilikom testiranja druge petlje, broj iteracija petlje od koje ona zavisi treba držati na minimumu.
  - Prva petlja se izvršava tačno onoliko puta koliko ima elemenata u nizu.
  - Dakle, prva petlja može najmanje da izvrši onoliko iteracija koliko će izvršiti i druga petlja.
- Dalje, testira se prva petlja:
  - Broj iteracija prve petlje je fiksiran na **n**, odnosno petlja se izvršava jednom za svaki element niza.
  - Kako maksimalan broj elemenata nije određen, može da se odabere neka proizvoljna granica za testiranje, npr. maksimalno 10 elemenata u nizu.



ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – BAJZEROVA METODA (4)

- Dakle, za prvu petlju, potrebno je testirati sledeće situacije:
  - Prva petlja se izvršava **0** puta.
  - Prva petlja se izvršava **1** put.
  - Prva petlja se izvršava **5** puta.
  - Prva petlja se izvršava **9** puta.
  - Prva petlja se izvršava **10** puta.
  - Prva petlja se izvršava **11** puta.

```
  public static void count(int[] a) {
1.    int count = 0;
2.    int n = a.length;
3.    for (int i = 0; i < n; i++) {
4.        if (a[i] > 0) count++;
5.    }
6.    for (int i = 0; i < count; i++) {
7.        a[i] += count;
8.    }
9. }
```

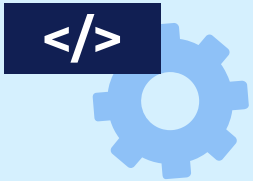


ZADATAK 4 – POZITIVNI BROJEVI

REŠENJE – BAJZEROVA METODA (5)

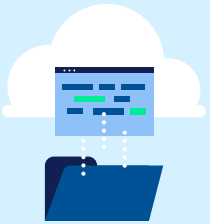
- Skup test primera za sve petlje:

TP	a	Očekivani izlaz	Br. iter. prve petlje	Br. iter. druge petlje
1	{ }	{ }	0	0
2	{ 1 }	{ 2 }	1	1
3	{ -1, -2, -3, 4, 5 }	{ 1, 0, -3, 4, 5 }	5	2
4	{ 1, 2, 3, 4, 5, 6, 7, 8, -9 }	{ 9, 10, 11, 12, 13, 14, 15, 16, -9 }	9	8
5	{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }	{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 }	10	10
6	{ 1, -2, 3, -4, 5, -6, 7, -8, 9, 10, 11 }	{ 8, 5, 10, 3, 12, 1, 14, -8, 9, 10, 11 }	11	7



03

LCSAJ TEHNIKA



LCSAJ SEKVENCE

- **LCSAJ** (*Linear Code Sequence And Jump*) je sekvenca naredbi koja se završava prenosom toka kontrole (skokom).
- Zove se i **JJ-path** (*Jump-to-Jump path*).
- Jedna sekvenca predstavljena je uređenom trojkom:

(*Start*, *Finish*, *Jump To*)

- *Start* – naredba koja predstavlja početak linearne sekvence
- *Finish* – naredba na kraju linearne sekvence kada se vrši skok
- *Jump To* – naredba na koju se „skače“



LCSAJ TEHNIKA

- LCSAJ tehnika testiranja prvo zahteva pronalaženje svih linearnih sekvenci, prema sledećim koracima:
    1. Pronađe se početna tačka u programu.
    2. Od početne tačke se definišu sve sekvence koda koje mogu da se izvrše do nekog skoka.
    3. Za svaku nađenu sekvencu koda se definiše uređena LCSAJ trojka.
    4. Za svako mesto gde je bilo moguće skočiti, postupak se ponavlja tako da to mesto skoka predstavlja novu početnu tačku.

- Nakon određivanja svih LCSAJ u programu, konstruišu se test primeri tako da se pokriju sve definisane sekvence.



ZADATAK 5 – GOTO SKOKOVI

POSTAVKA

- Dat je isečak programa u vidu pseudokoda u kojem se koristi *goto* naredba za bezuslovne skokove na delove programa označene labelama. Pretpostaviti da su X1, X2 i X3 ulazni parametri programa logičkog tipa, a da su promenljive A i B inicijalizovane vrednošću 0.

1. Odrediti sve LCSAJ za dati isečak programa.
2. Napisati minimalan skup test primera kojim će biti pokrivene sve LCSAJ iz prethodne tačke. Za svaki test primer navesti ulazne podatke, očekivani izlaz i LCSAJ koje su pokrivene test primerom.

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```



ZADATAK 5 – GOTO SKOKOVI

REŠENJE – SEKVENCE (1)

- Početna tačka programa je naredba na liniji 1.
- Naredbe koje se izvršavaju sekvencijalno do trenutka skoka, a koje počinju od linije 1 su:
  - 1, 2, skok sa linije 2 na liniju 7
    (ukoliko je X1 ispunjen)
  - 1, 2, 3, 4, skok sa linije 4 na liniju 7
    (ukoliko je X1 neispunjen, a X2 ispunjen)
  - 1, 2, 3, 4, 5, 6, skok sa linije 6 na liniju 10
    (ukoliko su i X1 i X2 neispunjeni)

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```

ZADATAK 5 – GOTO SKOKOVI

REŠENJE – SEKVENCE (2)

- Dakle, LCSAJ koje počinju od linije 1 su:

S	Start	Finish	Jump To
1	1	2	**7**
2	1	4	**7**
3	1	6	**10**

- Dalje, traže se nove sekvence čije je mesto početka linija na koju se moglo skočiti prethodnim sekvencama.



ZADATAK 5 – GOTO SKOKOVI

REŠENJE – SEKVENCE (3)

- Naredbe koje se izvršavaju sekvencijalno do trenutka skoka, a koje počinju od linije 7 su:
  - 7, 8, skok sa linije 8 na liniju 10
    (ukoliko je X3 ispunjen)
  - 7, 8, 9, 10, kraj programa
    (ukoliko je X3 neispunjen)

- Poslednja linija programa koja vraća tok kontrole pozivaocu ili prekida program predstavlja implicitni skok, a odredište skoka se označava sa –1.
- Dakle, nove LCSAJ su (7, 8, 10) i (7, 10, –1)

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```

ZADATAK 5 – GOTO SKOKOVI

REŠENJE – SEKVENCE (4)

- Jedina naredba koja se izvršava sekvencijalno do trenutka skoka, a koja počinje od linije 10 je:
  - 10, kraj programa
- Jedna naredba može biti ujedno i početak sekvence i kraj sekvence, odnosno mesto sa kog se vrši skok.
- Dakle, jedina nova LCSAJ je (10, 10, –1)

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```



ZADATAK 5 – GOTO SKOKOVI

REŠENJE – SEKVENCE (5)

- Dakle, LCSAJ koje počinju od linija 1, 7 i 10 su:

S	Start	Finish	Jump To
1	1	2	7
2	1	4	7
3	1	6	10
4	7	8	**10**
5	7	10	**–1**
6	10	10	**–1**

- Nova mesta skokova su 10 i –1. Linija 10 je već obrađena kao mesto početka, dok linija –1 predstavlja kraj programa, tako da je skup LCSAJ kompletiran.

ZADATAK 5 – GOTO SKOKOVI

REŠENJE – TEST PRIMERI (1)

- Uslovi koji moraju da budu zadovoljeni kako bi LCSAJ bile izvršene su:

S	Start	Finish	Jump To	Uslovi
1	1	2	7	X1 = true
2	1	4	7	X1 = false, X2 = true
3	1	6	10	X1 = false, X2 = false
4	7	8	10	X3 = true
5	7	10	-1	X3 = false
6	10	10	-1	-

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```

ZADATAK 5 – GOTO SKOKOVI

REŠENJE – TEST PRIMERI (2)

- Minimalni skup test primera koji pokriva sve LCSAJ (* označava da je vrednost parametra nebitna u test primeru):

TP	Ulazni podaci			Očekivani izlaz		Pokrivene sekvence
	X1	X2	X3	A	B	
1	true	true*	true	2	0	S-1, S-4, S-6
2	false	true	false	2	2	S-2, S-5, S-6
3	false	false	false*	2	1	S-3, S-6

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```

ZADATAK 5 – GOTO SKOKOVI

REŠENJE – PITANJA (1)

- Da li su ovim test primerima pokrivene sve odluke u programu?
- Da li pokrivanje svih odluka garantuje pokrivenost svih LCSAJ?
- Da li pokrivanje svih LCSAJ garantuje pokrivenost svih odluka?

TP	Ulazni podaci			Očekivani izlaz		Pokrivene sekvence
	X1	X2	X3	A	B	
1	true	true*	true	2	0	S-1, S-4, S-6
2	false	true	false	2	2	S-2, S-5, S-6
3	false	false	false*	2	1	S-3, S-6

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```



ZADATAK 5 – GOTO SKOKOVI

REŠENJE – PITANJA (2)

- Da li bi se LCSAJ i/ili test primeri promenili ukoliko bi se program izmenio u liniji 4, na sledeći način?

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2 AND X1) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```



ZADATAK 5 – GOTO SKOKOVI

REŠENJE – PITANJA (3)

- Zbog kontradiktornih uslova, druga sekvenca sada nije izvršiva!
- Minimalni broj test primera za pokrivanje izvršivih sekvenci ostaje 3.

S	Start	Finish	Jump To	Uslovi
1	1	2	7	X1 = true
2	1	4	7	**X1 = false**, X2 = true, **X1 = true**
3	1	6	10	X1 = false, X2 = false
4	7	8	10	X3 = true
5	7	10	-1	X3 = false
6	10	10	-1	-

```
1.   A = 1
2.   IF (X1) THEN GOTO L1
3.   B = 1
4.   IF (X2 AND X1) THEN GOTO L1
5.   A = 2
6.   GOTO L2
7.   L1: A = A + 1
8.   IF (X3) THEN GOTO L2
9.   B = B + 1
10.  L2: PRINT(A, B)
```

ZADATAK 6 – INSERTION SORT

POSTAVKA

- Data je implementacija funkcije **sort** u programskom jeziku Java koja sortira niz celih brojeva korišćenjem *Insertion Sort* algoritma. Pretpostaviti da funkcija **print** ispisuje prosleđeni niz na standardni izlaz.

1. Odrediti sve LCSAJ za datu funkciju. Za svaku sekvencu analizirati uslove koji omogućavaju njeno izvršavanje.
2. Napisati minimalan skup test primera kojim će biti pokrivene sve LCSAJ iz prethodne tačke. Za svaki test primer navesti ulazni niz, transformisani niz nakon poziva funkcije i LCSAJ koje su pokrivene test primerom.

```
   public static void sort(int[] arr) {
1.     int n = arr.length;
2.     for (int i = 1; i < n; i++) {
3.         int p = arr[i];
4.         int j = i - 1;
5.         while (j >= 0 && p < arr[j]) {
6.             arr[j + 1] = arr[j--];
7.             print(arr);
8.         }
9.         arr[j + 1] = p;
10.        print(arr);
11.    }
12.}
```

ZADATAK 6 – INSERTION SORT

REŠENJE – SEKVENCE (1)

- LCSAJ koje počinju od linije 1 su:

Start	Finish	Jump To
1	2	**12**
1	5	**9**
1	8	**5**

```
   public static void sort(int[] arr) {
1.     int n = arr.length;
2.     for (int i = 1; i < n; i++) {
3.         int p = arr[i];
4.         int j = i - 1;
5.         while (j >= 0 && p < arr[j]) {
6.             arr[j + 1] = arr[j--];
7.             print(arr);
8.         }
9.         arr[j + 1] = p;
10.        print(arr);
11.    }
12.}
```



ZADATAK 6 – INSERTION SORT

REŠENJE – SEKVENCE (2)

- LCSAJ koje počinju od linija 5, 9 i 12 su:

Start	Finish	Jump To
5	5	9
5	8	5
9	11	**2**
12	12	-1

```
   public static void sort(int[] arr) {
1.     int n = arr.length;
2.     for (int i = 1; i < n; i++) {
3.         int p = arr[i];
4.         int j = i - 1;
5.         while (j >= 0 && p < arr[j]) {
6.             arr[j + 1] = arr[j--];
7.             print(arr);
8.         }
9.         arr[j + 1] = p;
10.        print(arr);
11.    }
12.}
```

ZADATAK 6 – INSERTION SORT

REŠENJE – SEKVENCE (3)

- LCSAJ koje počinju od linije 2 su:

Start	Finish	Jump To
2	2	12
2	5	9
2	8	5

```
   public static void sort(int[] arr) {
1.     int n = arr.length;
2.     for (int i = 1; i < n; i++) {
3.         int p = arr[i];
4.         int j = i - 1;
5.         while (j >= 0 && p < arr[j]) {
6.             arr[j + 1] = arr[j--];
7.             print(arr);
8.         }
9.         arr[j + 1] = p;
10.        print(arr);
11.    }
12.}
```

ZADATAK 6 – INSERTION SORT

REŠENJE – SEKVENCE (4)

- Kompletan skup LCSAJ:

S	Start	Finish	Jump To
1	1	2	12
2	1	5	9
3	1	8	5
4	2	2	12
5	2	5	9
6	2	8	5
7	5	5	9
8	5	8	5
9	9	11	2
10	12	12	-1

ZADATAK 6 – INSERTION SORT

REŠENJE – SEKVENCE (5)

- Uslovi za izvršavanje LCSAJ:

SQ	S	F	J	Uslov
1	1	2	12	Niz ima samo jedan element.
2	1	5	9	Drugi element niza je veći od prvog.
3	1	8	5	Drugi element niza je manji od prvog.
4	2	2	12	Niz ima bar dva elementa.
5	2	5	9	Niz ima bar dva elementa i postoji element niza koji je veći od elementa iza njega.
6	2	8	5	Niz ima bar dva elementa i postoji element niza koji je manji od elementa iza njega.
7	5	5	9	Postoji element niza koji je veći od elementa iza njega.
8	5	8	5	Postoji element niza koji je manji od elementa iza njega.
9	9	11	2	Niz ima bar dva elementa.
10	12	12	-1	-

```
   public static void sort(int[] arr) {
1.      int n = arr.length;
2.      for (int i = 1; i < n; i++) {
3.          int p = arr[i];
4.          int j = i - 1;
5.          while (j >= 0 && p < arr[j]) {
6.              arr[j + 1] = arr[j--];
7.              print(arr);
8.          }
9.          arr[j + 1] = p;
10.         print(arr);
11.     }
12. }
```

ZADATAK 6 – INSERTION SORT

REŠENJE – TEST PRIMERI

- Minimalni skup test primera koji pokrivaju sve LCSAJ:

TP	Ulazni podaci	Očekivani izlaz	Pokrivene sekvence
1	{ 5 }	{ 5 }	S-1, S-10
2	{ 1, 2, 3 }	{ 1, 2, 3 }	S-2, S-9, S-5, S-4, S-10
3	{ 5, 4, 1, 2, 3 }	{ 1, 2, 3, 4, 5 }	S-3, S-7, S-9, S-6, S-8, S-4, S-10

```
   public static void sort(int[] arr) {
1.      int n = arr.length;
2.      for (int i = 1; i < n; i++) {
3.          int p = arr[i];
4.          int j = i - 1;
5.          while (j >= 0 && p < arr[j]) {
6.              arr[j + 1] = arr[j--];
7.              print(arr);
8.          }
9.          arr[j + 1] = p;
10.         print(arr);
11.     }
12. }
```

UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI
FAKULTET

PITANJA?

https://si3ts.etf.bg.ac.rs/

nikolas@etf.bg.ac.rs

