

POTPROGRAMI

Definicija potprograma

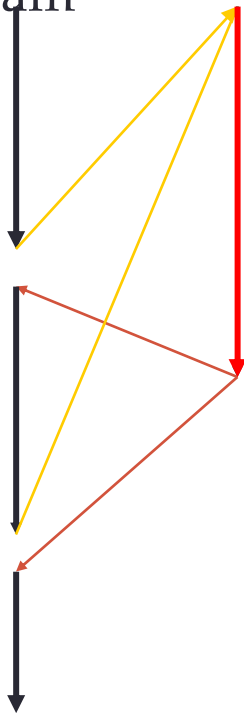
Parametri potprograma

Rekurzije

Organizacija memorije u toku izvršenja programa

Pojam potprograma

Glavni program Potprogram



Elementi potprogram

Potprogram karakterišu četiri osnovna elementa:

- ime potprograma
- lista fiktivnih parametara
- telo potprograma
- sredina u kojoj potprogram definisan.

Tipovi potprograma

- Funkcijski potprogrami (funkcije)
 - Imaju tačno jednu povratnu vrednost
- Potprogrami opšteg tipa (procedure)
 - Mogu imati više (ili ni jednu) povratnu vrednost

Definicija potprograma u *C-like* jezicima

- U programskim jezicima koji nasleđuju sintaksu programskog jezika C postoji samo jedna vrsta potprogram (funkcije)
- Definicija funkcije u programskom jeziku C:

```
<tip> ime(<lista_fiktivnih_parametara>)  
{  
    //telo potprograma  
}
```

- **<tip>** – tip rezultata funkcije
 - Ukoliko funkcija ne vraća nikakav rezultat, kao povratni tip se koristi tip **void**

Naredba povratka iz potprograma

- Naredba za povratak iz programa ima 2 formata:
 - `return <izraz>;` - za funkcije koje vraćaju rezultat
 - `return;` - za funkcije tipa `void`

Poziv potprograma

`ime(<lista_stvarnih_parametara>)`

- Lista fiktivnih i lista stvarnih argumenata mora da se slaže po broju i tipovima argumenata.
 - Stvarni parametar je izraz istog tipa kao odgovarajući fiktivni parametar.
- Primeri poziva potprograma:

```
int a, b, min1, min2, min3;
```

...

```
min1 = min(a,b) ;
```

```
min2 = min(a, 5) ;
```

```
min3 = min(5*a, b+7) ;
```

Parametri potprograma

Parametri potprograma

- Semantički posmatrano prenos parametara u potprogram može da bude po jednom od tri sematička modela.

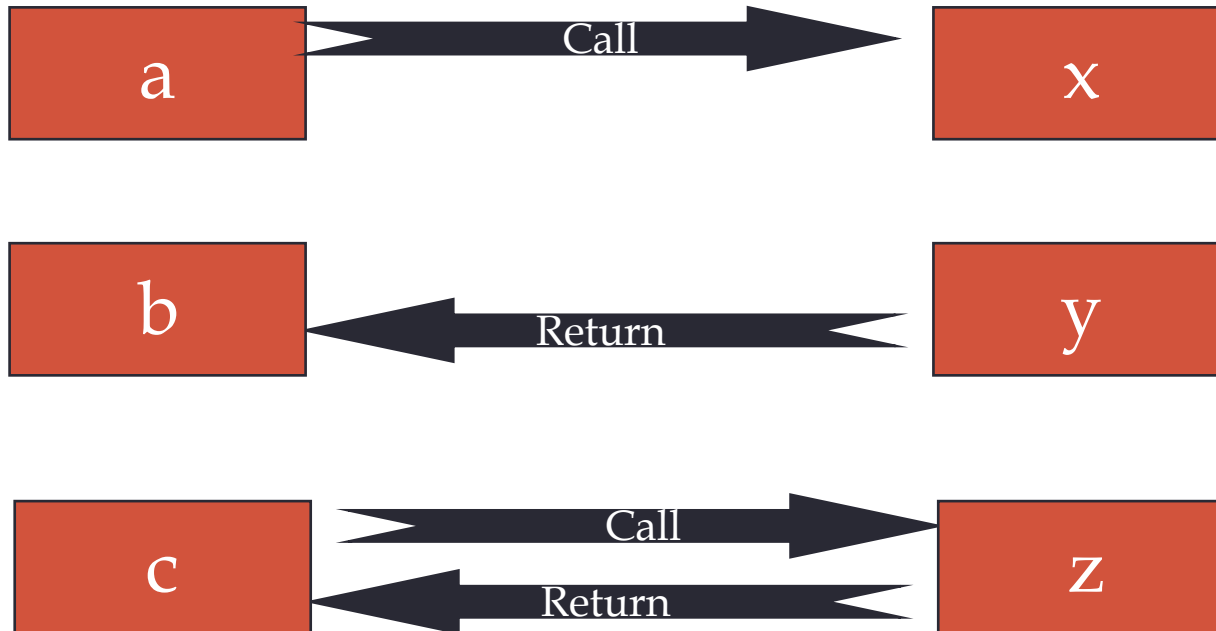
Potprogram može da:

- od glavnog programa primi vrednost parametra (**ulazni**),
- da mu preda vrednost (**izlazni**),
- da primi vrednost i preda rezultat glavnom programu (**ulazno-izlazni**).

Parametri potprograma

SUB(a,b,c)

```
procedure SUB(x : in INTEGER, y:  
out INTEGER, z: inout INTEGER);
```



Prenos parametara

- U teoriji programskih jezika najčešće korišćene su dve metode prenosa parametara:
 - **Po vrednosti**
 - U trenutku poziva rezerviše se novi prostor u memoriji za fiktivni parametar, i u taj prostor upisuje vrednost stvarnog parametra.
 - **Po referenci**
 - Fiktivni parametar je drugo ime za stvarni parametar koji se navodi u pozivu

Prenos parametara po vrednosti

Call by value

Glavni program



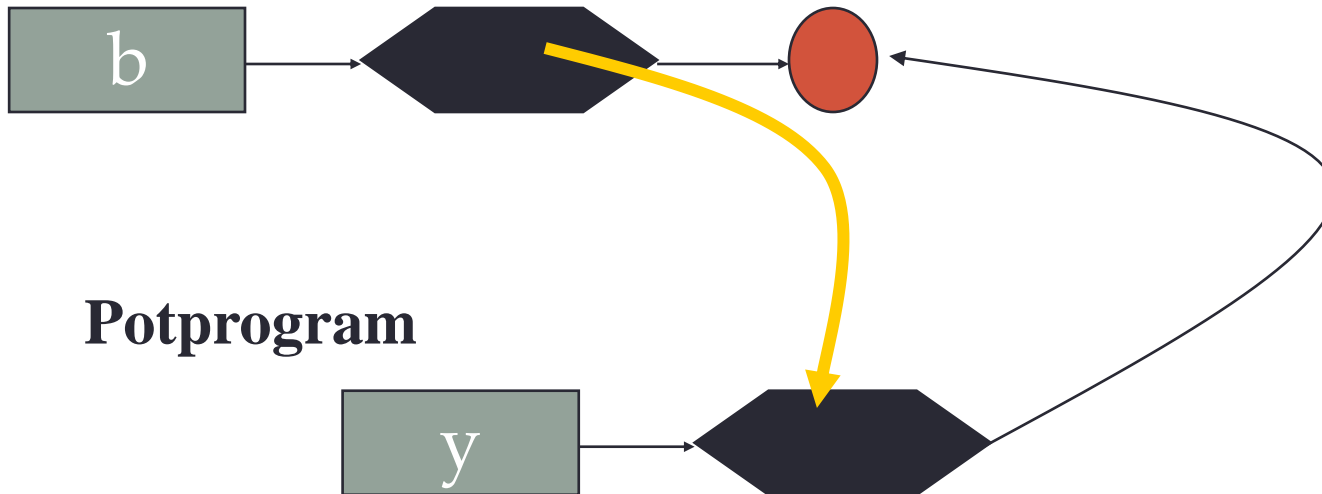
Potprogram



Prenos parametara po referenci

Call by reference

Glavni program



Druge tehnike prenosa

- Prenos po rezultatu (call by result)
- Prenos po vrednosti i rezultatu (call by Value-Result)
- Prenos po imenu (Call by Name)

Call by Name

```
real procedure SUMA(x,j,n);  
  value n; real x; integer j,n;  
begin  
  real S;  
  S := 0.0;  
  for j := 1 step 1 until n do  
    S := S + x;  
  SUMA := S  
end.
```

Call SUMA(A, I, 100)

```
begin  
  real S;  
  S := 0.0;  
  for I := 1 step 1 until 100 do  
    S := S + A;  
  SUMA := S  
end.
```

Call by Name

```
real procedure SUMA(x,j,n);  
  value n; real x; integer j,n;  
begin  
  real S;  
  S := 0.0;  
  for j := 1 step 1 until n do  
    S := S + x;  
  SUMA := S  
end.
```

Call SUMA(A[I], I, 100)

```
begin  
  real S;  
  S := 0.0;  
  for I := 1 step 1 until 100 do  
    S := S + A[I];  
  SUMA := S  
end.
```

Call SUMA(A[I]*A[I], I, 100)

```
begin  
  real S;  
  S := 0.0;  
  for I := 1 step 1 until 100 do  
    S := S + A[I] * A[I];  
  SUMA := S  
end.
```


Prenos parametara u programskom jeziku C

- Parametri se prenose isključivo po vrednosti.
- Kad funkcija treba da menja vrednost parametra, funkciji se prenose pokazivači.

Prenos parametara u programskom jeziku C++

- Postoje oba načina prenosa:
 - Po vrednosti
 - Po referenci
- Programer odlučuje da li će se prenos parametara vršiti po vrednosti ili po referenci

```
void f(int i, int &j) // i po vrednosti,  
                     // j po referenci  
{  
    i++;  
    j++;  
}
```

Prenos parametara u Javi

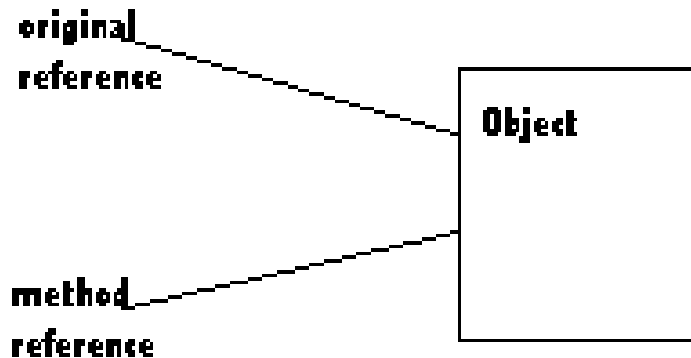
- Postoji jedino prenos parametara po vrednosti.
- Tipovi podataka u Javi:
 - Vrednosni
 - Svi primitivni tipovi podataka
 - Referentni
 - Polja
 - Enumeracije
 - Klase
 - Interfejsi

Prenos parametara vrednosnog tipa u Javi

- Pri prenosu se rezerviše novi prostor za vrednost parametra i tu upisuje vrednost stvarnog argumenta
- Izmena vrednosti parametra se ne odražava na stvarni argument

Prenos parametara referentnog tipa u Javi

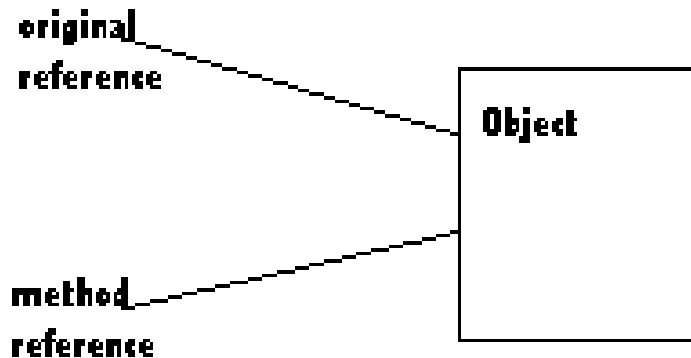
- Prilikom poziva funkcije se kreira nova referenca, ali ne i novi objekat.



- Promena objekta u metodi, znači promenu originalnog objekta.
- Ukoliko se referentnom parametru u metodi dodeli drugi objekat, promena tog objekta se neće videti u pozivajućem modulu.

Prenos parametara referentnog tipa u Javi

- Prilikom poziva funkcije se kreira nova referenca, ali ne i novi objekat.



- Promena objekta u metodi, znači promenu originalnog objekta.
- Ukoliko se referentnom parametru dodeli u metodi dodeli drugi objekat, promena tog objekta se neće videti u pozivajućem modulu.

Prenos parametara referentnog tipa u Javi

```
void f(MyClass p)
{
    p.Change();      //stvarni parametar se menja
}
```

```
void g(MyClass p)
{
    p = new MyClass();
    p.Change();      //stvarni parametar nepromenjen
}
```

Parametri u programskom jeziku C#

- Parametri u programskom jeziku C# mogu biti:
 - Ulazni – podrazumevani, navode se bez eksplicitnog navođenja vrste prenosa,
 - Izlazni - ispred definicije parametra stoji ključna reč **out**,
 - Ulazno-izlazni – ispred definicije parametra stoji ključna reč **ref**.
- Način prenosa:
 - Ulazni – po vrednosti
 - Izlazni i ulazno-izlazni – po referenci

Parametri vrednosnog tipa u jeziku C#

- Ulazni – prenosi se po vrednosti
 - Pravi se kopija podatka u memoriji
 - Promene parametra unutar metode se ne vide van nje
 - Kao stvarni parametri se mogu koristiti promenljive, konstante ili izrazi odgovarajućeg tipa
- Izlazni ili ulazno-izlazni - prenosi se po referenci
 - Promene parametra unutar metode se vide u pozivajućem modulu
 - Štedi se memorijski prostor
 - Ulazni parametar se ne može koristiti u metodi pre nego što mu se dodeli vrednost
 - Stvarni parametar u pozivu može biti samo promenljiva

Parametri referentnog tipa u jeziku C#

- Ulazni parametar – prenos po vrednosti
 - Kreira se kopija reference, ali ne i podatka
 - Metoda može da menja objekat (ta promena će biti vidljiva spolja), ali ne može parametru da dodeli neki drugi objekat.
- Izlazni ili ulazno-izlazni parametar – prenos po referenci
 - Koristi se ista referenca kao i u pozivajućem modulu
 - Metoda može parametru da dodeli sasvim drugi objekat.
 - Ukoliko je parametar označen kao **out**, stvarni argument može imati i vrednost **null** pre poziva.

Parametri referentnog tipa u jeziku C#

```
void f(MyClass p)
{
    p.Change();
}
```

```
void g(ref MyClass p)
{
    p.DoSomething();
    p = new MyClass();
    p.Change();
}
```

```
void q(out MyClass p)
{
    p = new MyClass();
    p.Change();
}
```

Podrazumevane vrednosti parametara

- U nekim jezicima (C++,C#,...) se mogu navesti podrazumevane vrednosti argumenata.

- Definicija funkcije sa podrazumevanim vrednostima parametara:

```
void f( float a=0.5f, int k=0 ) ...
```

- Podrazumevane vrednosti parametara se koriste ukoliko u pozivu funkcije odgovarajući stvarni parametar nije naveden.

- Dozvoljeni pozivi funkcije f:

```
f( 15.5f, 2 );
```

```
f( 15.5f );
```

```
f( 2 );           // a dobija vrednost 2
```

```
f();
```

Podrazumevane vrednosti parametara

- Podrazumevane vrednosti mogu imati samo poslednji parametri u listi.

- Dozvoljena definicija funkcije sa podrazumevanim vrednostima parametara:

```
void f( float a, int k=0 )
```

- Nedozvoljena definicija funkcije sa podrazumevanim vrednostima parametara:

```
void f( float a=0.5f, int k )
```

Funkcije sa promenljivim brojem parametara istog tipa

- Neki jezici (C#, Java,...) omogućavaju da se definišu funkcije sa proizvoljnim brojem parametara istog tipa.

Funkcije sa promenljivim brojem parametara u programskom jeziku C#

- Ključna reč **params** omogućava da se funkcija poziva sa različitim brojem parametara istog tipa.
- U listi parametara samo poslednji može da ima ispred sebe ovu ključnu reč
- Nakon ključne reči params obavezno stoji niz
- Funkcija takav parametar vidi kao niz, a u pozivu se navode nezavisni stvarni parametri

Funkcije sa promenljivim brojem parametara u programskom jeziku C#

- Primer:

```
public static float prosek(params int[] ocene)
{
    float s=0;
    for (int i=0; i<ocene.Length; i++)
        s += ocene[i];
    return s/ocene.Length;
}
```

//poziv metode:

... prosek(7, 8, 7, 9, 6) ...

Funkcije sa promenljivim brojem parametara u programskom jeziku Java

- Parametar koji se u pozivu može zameniti nizom parametara se u programskom jeziku Java obeležava tako što se iza imena tipa parametra navede znak ...
- Primer:

```
public static float prosek(int... ocene)
{
    float s=0;
    for (int i=0; i<ocene.length; i++)
        s += ocene[i];
    return s/ocene.length;
}
```

//poziv metode:

```
... prosek(7, 8, 7, 9, 6) ...
```

Rekurentnti potprogami

Rekurentni potprogrami

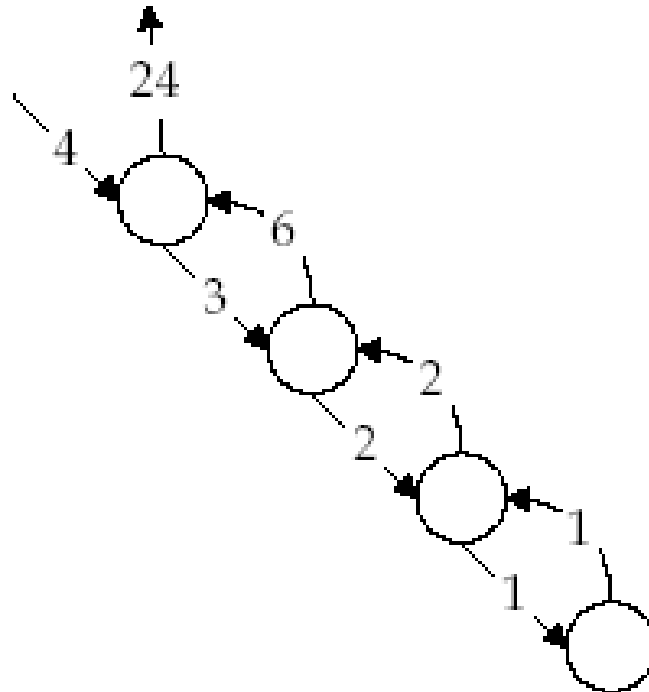
- Neki programski jezici dozvoljavaju poziv potprograma u telu samog tog potprograma. Takvi potprogrami nazivaju se rekurzivnim i pogodno su sredstvo za rešavanje problema koji su matematički rekurzivno definisani.
- Tipičan primer rekurzivnog potprograma je potprogram za izračunavanje faktoriijela:

$$n! = \begin{cases} n \cdot (n-1)! & \text{za } n \geq 1 \\ 1 & \text{za } n = 0 \end{cases}$$

Funkcija za izračunavanje faktoriijela

```
int fact(int n)
{
    if ( n < 2 )
        return 1;
    return n * fact( n - 1 );
}
```

Graf poziva funkcije

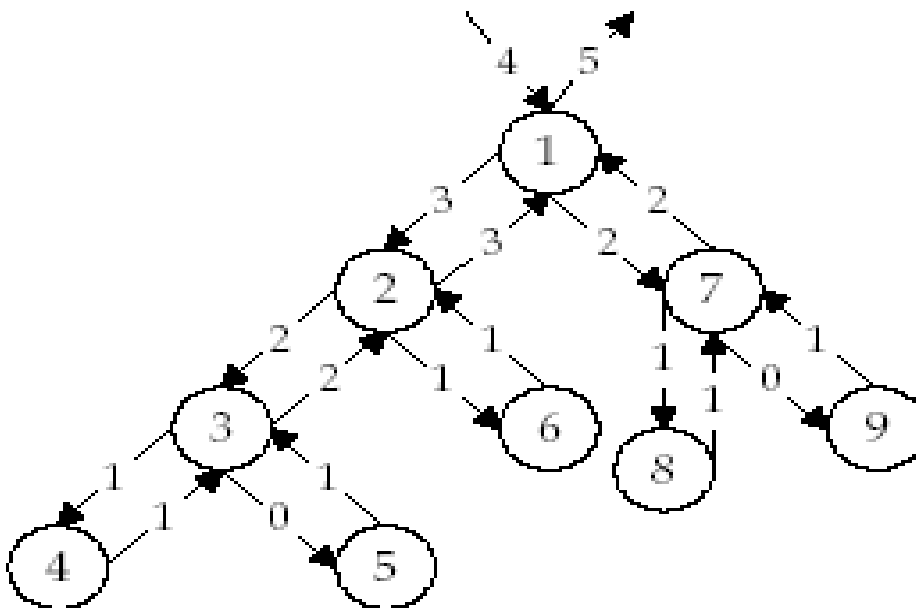


*Graf poziva funkcije **fact** za $n = 4$*

Fibonačijev niz

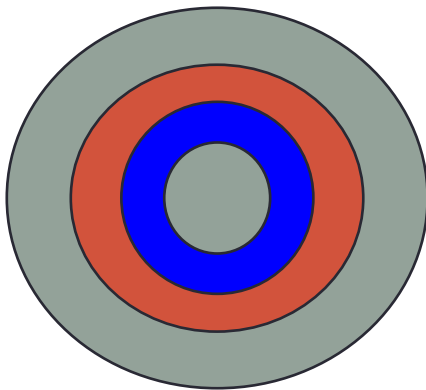
$$f_{(n)} = \begin{cases} f_{(n-1)} + f_{(n-2)}, n > 1 \\ 1, n = 1 \\ 1, n = 0 \end{cases}$$

```
int fib(int n)
{
    if ( n < 2 )
        return 1;
    return fib(n-1) + fib(n-2);
}
```



Hanojeve kule

```
void Hanoi(int n, int a, int b, int c)
{
    if (n>0)
    {
        Hanoj(n-1, a, c, b);
        PrebaciDisk(a, b);
        Hanoj(n-1, c, b, a);
    }
}
```



18.446.744.073.709.551.615

za n=64

Organizacija memorije u toku izvršenja programa

Organizacija memorije

- Memorija dodeljena jednom programu može da se подели na sledeće delove:
 - Deo sa generisanim kodom
 - Polje podataka
 - Deo sa aktivacionim slogovima potprograma

Struktura aktivacionog sloga

Rezultati koje vraća potprogram

Stvarni parametri

Adresa povratka

*Upravljački linkovi i linkovi za
pristup podacima iz okoline*

Kontekst procesora

Lokalni podaci

Privremene promenljive

Struktura aktivacionog sloga

- Temporalne promenljive – Promenljive koje generiše kompilator prilikom evaluacije aritmetičkih izraza
- Lokalni podaci potprograma
- Polje gde se pamti status procesora prilikom prenošenja upravljanja na potprogram
- Opcioni linkovi do nelokalnih podataka koji su sadržani u drugim aktivacionim slogovima.
- Opcioni linkovi do aktivacionog sloga programa iz kojeg je pozvan potprogram.
- Stvarni argumenti potprograma
- Polje preko kojeg se vraćaju rezultati u glavni program.

Strategije alokacije memorije

- Statička alokacija svih objekata u vreme kompilacije.
- Alokacija pomoću steka
- Dinamička alokacija pomoću Heap-a.

Statička alokacija memorije

- U toku kompiliranja programa generiše se po jedan aktivacioni slog za svaki potprogram tako da se isti aktivacioni slog koristi kod svakog poziva potprograma.
- Nema mogućnosti za rekurzivne pozive procedura
- Strukture podataka se ne mogu kreirati dinamički.
- Primjenjuje se kod implementacije jezika Fortran 77.

Primer statičke alokacije memorije

PROGRAM CONSUME

CHARACTER * 50 BUF

INTEGER NEXT

CHARACTER C, PRODUCE

DATA NEXT /1/, BUF /' '/

6

C = PRODUCE ()

BUF(NEXT:NEXT) = C

NEXT = NEXT + 1

IF (C .NE. ' ') GO TO 6

WRITE (*, '(A)') BUF

END

CHARACTER FUNCTION PRODUCE ()

CHARACTER * 80 BUFFER

INTEGER NEXT

SAVE BUFFER, NEXT

DATA NEXT /81/

IF (NEXT .GT. 80) THEN

READ (*, '(A)') BUFFER

NEXT = 1

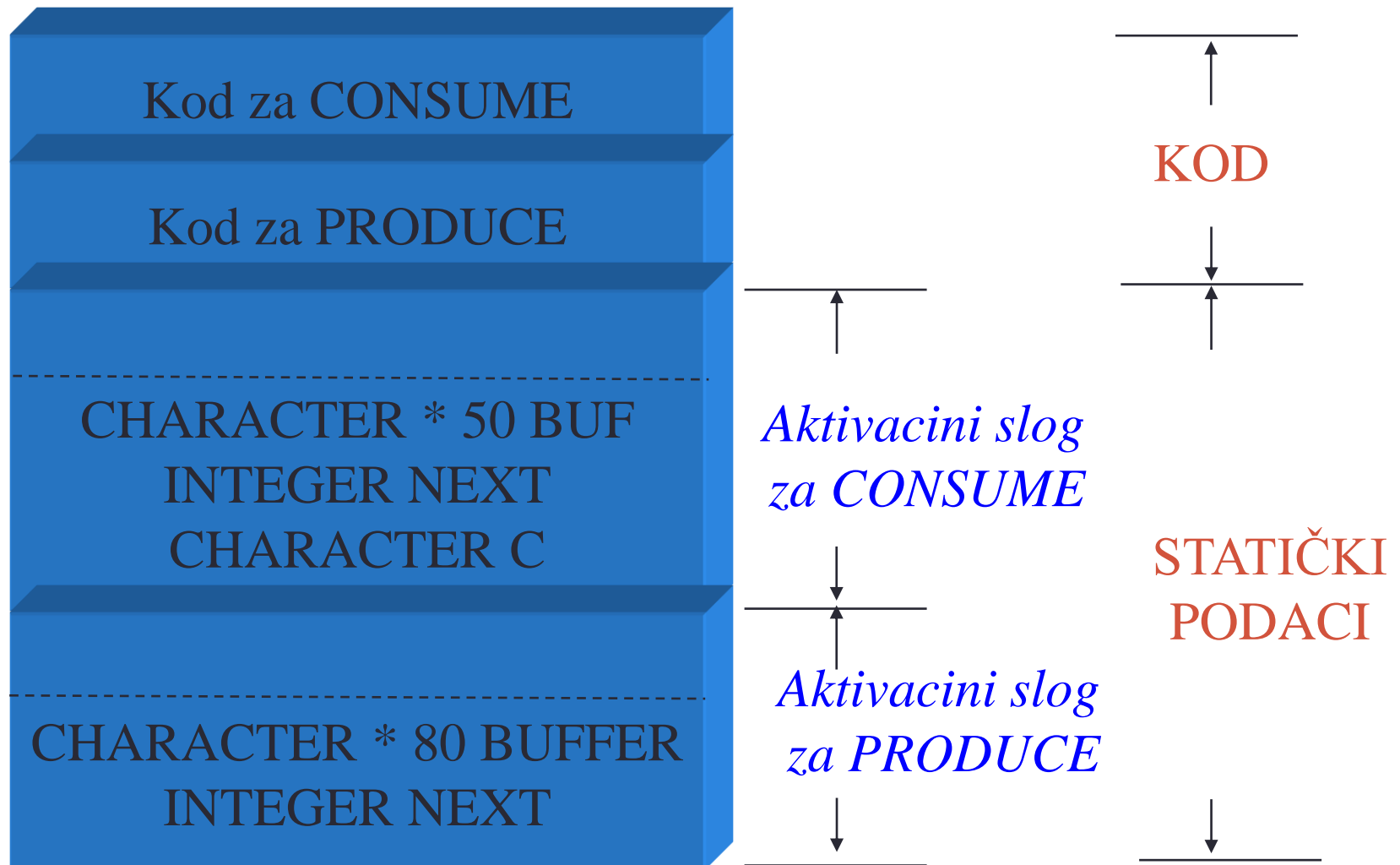
END IF

PRODUCE = BUFFER(NEXT:NEXT)

NEXT = NEXT + 1

END

Statička raspodela memorije za program iz primera



STEK alokacija

- Memorija predviđena za pamćenje aktivacionih slogova je organizovana kao stek.
- Prilikom svakog poziva potprograma u stek se ubacuje njegov aktivacioni slog.
- Slog se izbacuje iz steka kada se potprogram završi.
- Ova tehnika omogućava rekurzivne pozive potprograma. U tom slučaju u steku se nalazi više aktivacionih slogova istog potprograma i svaki sadrži odgovarajuće podatke.
- U vreme kompiliranja programa zna se samo veličina aktivacionog sloga ali ne i dubina rekurzije (koliko će aktivacionih slogova jednog potprograma biti generisano).

STEK alokacija



Aktivacioni slogovi

Dinamičke strukture procedura

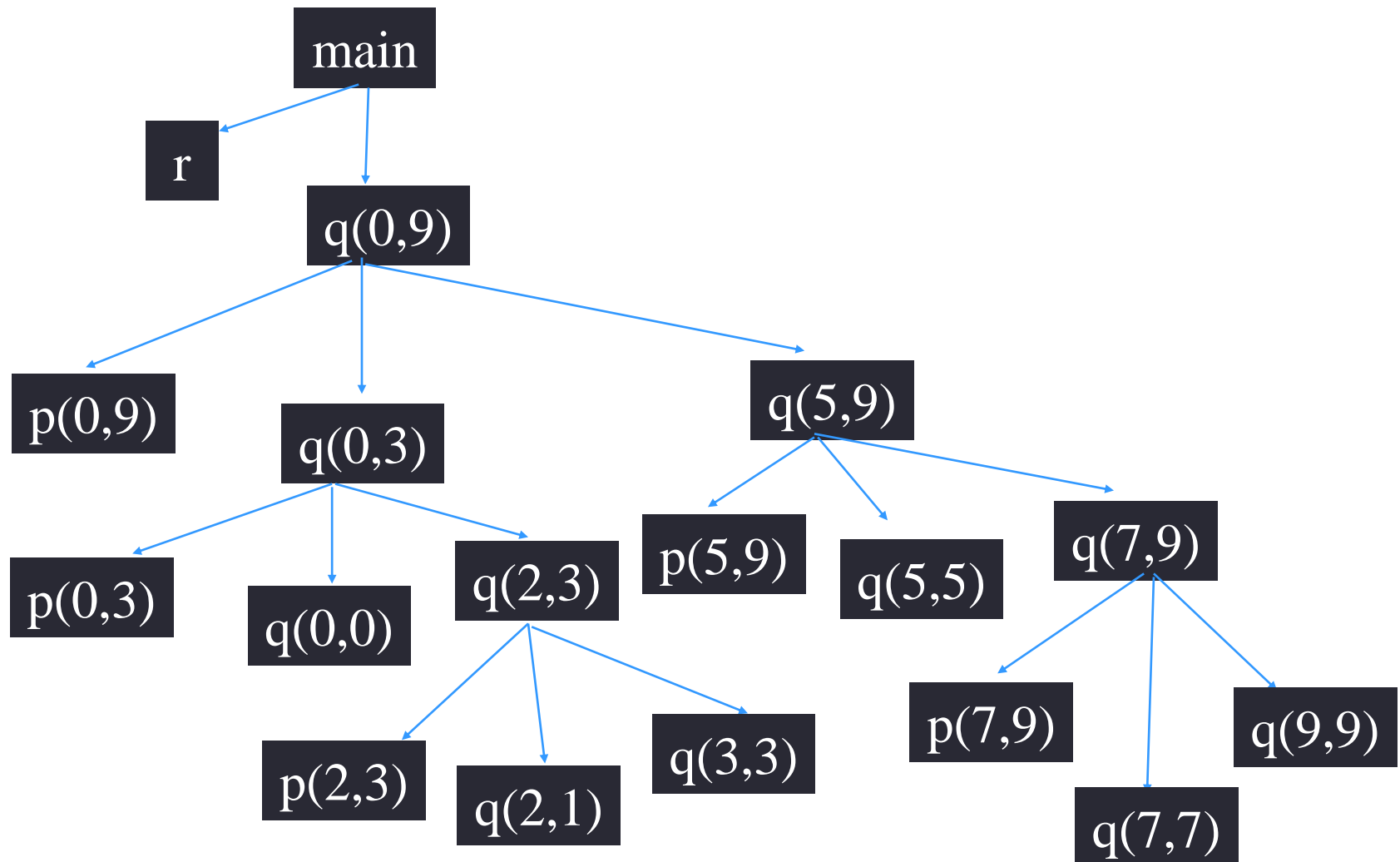
Primer

```
void quiksort (int m, int n)
{
    if (n>m)
    {
        i = partition (m,n) ;
        quiksort(m, i-1) ;
        quiksort(i+1, n) ;
    }
}
```

Primer

```
int a[10];  
void main ()  
{  
    readarray();  
    quiksort(0,9);  
}
```

Aktivaciono stablo za sort



STEK

AKTIVACIONO
STABLO

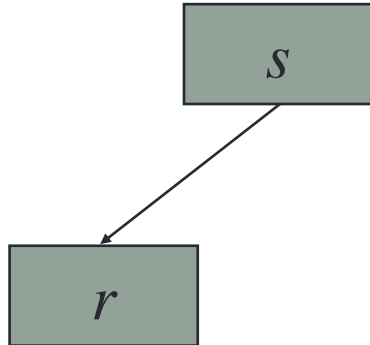


AKTIVACIONI
SLOGOVI U STEKU

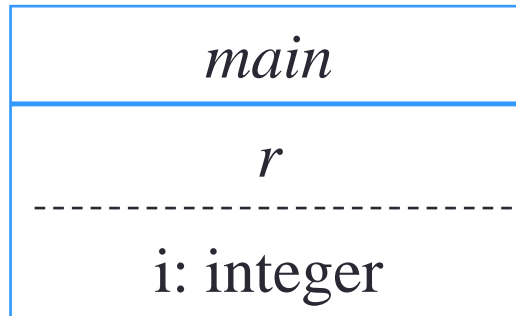


STEK alokacija za primer programa SORT

AKTIVACIONO
STABLO

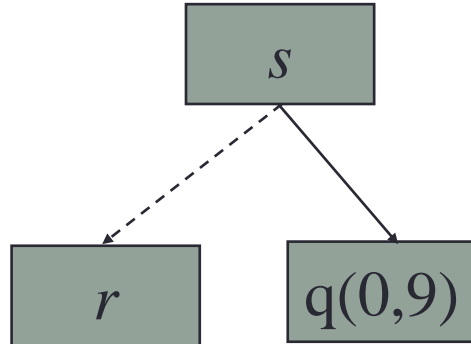


AKTIVACIONI
SLOGOVI U STEKU

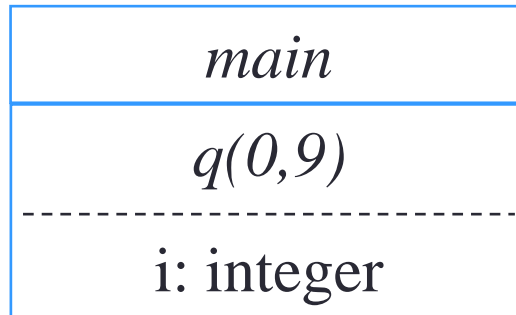


STEK alokacija za primer programa SORT

AKTIVACIONO
STABLO

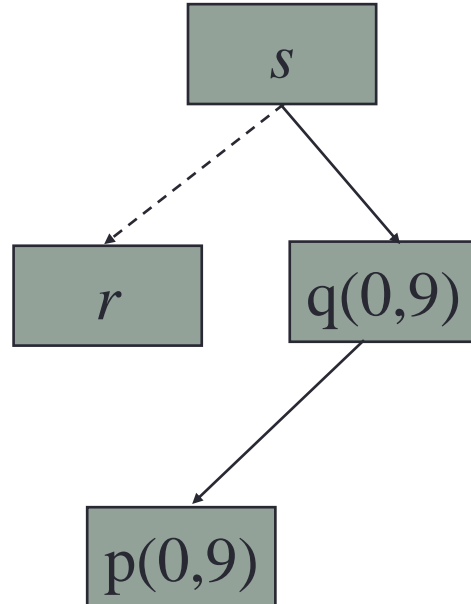


AKTIVACIONI
SLOGOVI U STEKU

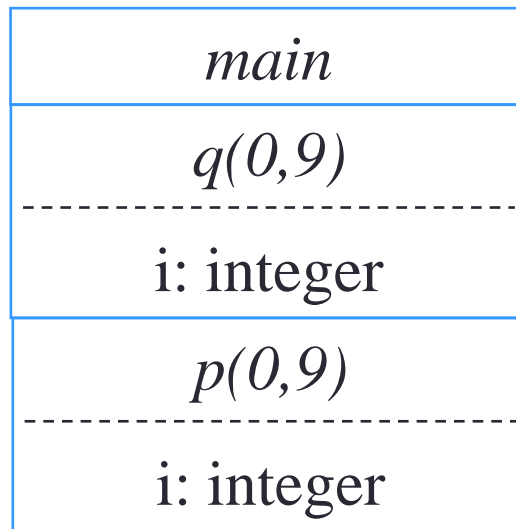


STEK alokacija za primer programa SORT

AKTIVACIONO STABLO

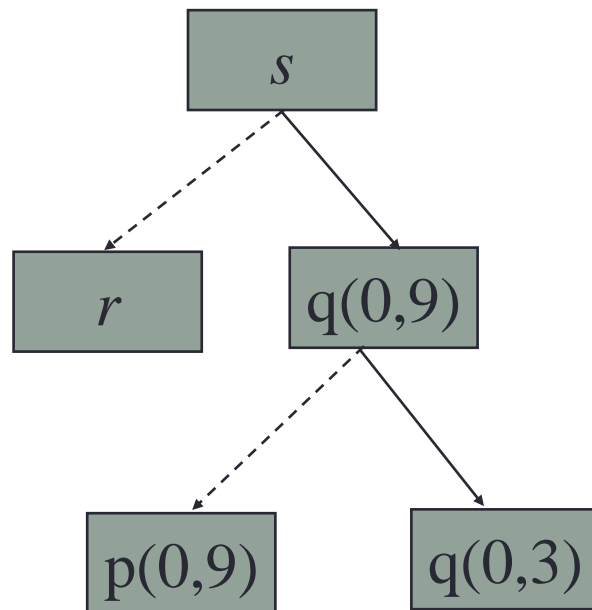


AKTIVACIONI SLOGOVI U STEKU

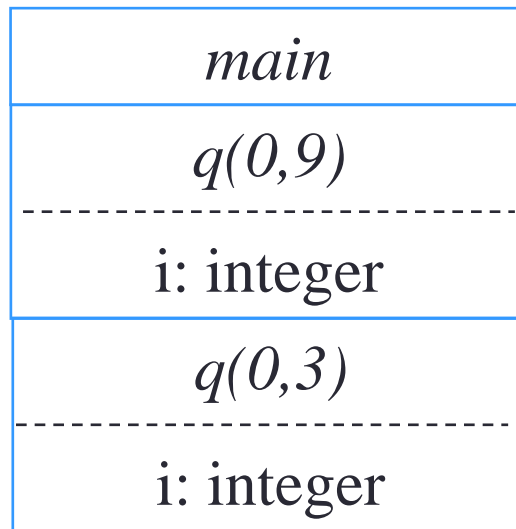


STEK alokacija za primer programa SORT

AKTIVACIONO STABLO



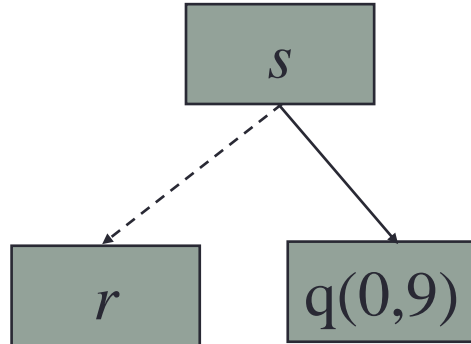
AKTIVACIONI SLOGOVI U STEKU



Dinamička alokacija

Aktivacioni slogovi se smeštaju u dinamičku memoriju i ne izbacuju se kao kod stek alokacije.

AKTIVACIONO
STABLO



AKTIVACIONI
SLOGOVI U HEAP-u

