Datum:23.02.2015

Vježbe 1

Rekurzivne funkcije - rekurzija

Ovaj materijal ima za cilj da predstavi osnovne koncepte koji se odnose na pojam *rekurzije*. Naime, *rekurzija* predstavlja sposobnost funkcije da poziva samu sebe. Značaj i primjena rekurzivnih funkcija posebno je naglašena u određenim područjima programiranja koje se koriste za potrebe implementacije algoritama umjetne inteligencije. Pored toga, primjena rekurzivnih funkcija nije baš rijetka ni u svakodnevnom radu. Pred programere se ponekad postavljaju zadaci koji zbog svoje veličine zahtijevaju kompleksna rješenja. Kompleksni zadaci se u većini slučajeva mogu podijeliti na više manjih cjelina. Ideja rekurzije leži upravo u tome da se, koristeći rekurzivne algoritme, problem podijeli na manje dijelove koje je jednostavnije riješiti.

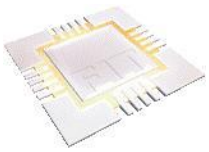
Da bi prethodno pomenuto postalo jasnije u nastavku je opisan mogući način implementacije rekurzije.

```
void imeFunkcije (argument) //definicija funkcije koja ne vraća vrijednost
{
      Iskaz1; //iskaz koji će se izvršiti pri svakom pozivu funkcije – uzlazni tok

      if (nekiUslov){               //provjera nekog uslova
         argument--;          //dekrementiranje vrijednosti
         imeFunkcije(argument);  //ponovni poziv funkcije -- REKURZIJA
      }                     //kraj if-a
      Iskaz2; //iskaz koji će se izvršiti kada uslov ne bude ispunjen – silazni tok
}  //kraj funkcije
```

Bitno je pomenuti da rekurzivne funkcije posjeduju nekoliko osobina o kojima treba voditi računa. Jedna od osobina rekurzivnih funkcija se odnosi na mogućnost da se nepažnjom dobije nekontrolisana ili beskonačna rekurzija. Najčešći uzrok beskonačne rekurzije je nepostojanje ili nepravilno definisanje tzv. *baznog slučaja* koji ima zadatak da u određenom momentu onemogući ponovno pozivanje funkcije, odnosno uzrokuje prekid rekurzije. Upravo zbog toga, za efikasno funkcionisanje, svaka rekurzivna funkcija zahtijeva postojanje najmanje jednog baznog slučaja.

Pošto su predstavljeni osnovni pojmovi rekurzije, vrijeme je za njihovu praktičnu primjenu.

Zadatak 1. Koristeći rekurziju, napisati program koji od korisnika traži da unese broj čija vrijednost mora biti u opsegu od 1 do 10. Unesenu vrijednost predati funkciji, koja treba da se izvršava onoliko puta kolika je vrijednost predanog argumenta (ako korisnik unese broj 5, funkcija treba da 5 puta pozove sama sebe). Pri svakom pozivu, funkcija treba da ispiše trenutnu vrijednost predanog argumenta.



```
#include <iostream>
using namespace std;

void rekFunkcija(int broj)  //zaglavlje rekurzivne funkcije
{
      if (broj>0){              //provjera uslova da li je broj veci od 0
            cout<<"Trenutna vrijednost varijable broj je:"<<broj<<endl;
            broj--;           //dekrementiranje vrijednosti argumenta broj
            rekFunkcija(broj);  //ponovni poziv funkcije - rekurzija

      }               //kraj if-a
}//kraj definicije funkcije

void main(){
      cout<<"\t\t ::REKURZIJA:: \n\n";
//broj koji unosi korisnik, a kojim se odredjuje broj rekurzivnih poziva
      int broj = 0;
      do{
            cout<<"Unesite broj od 1 do 10: ";
            cin>>broj;
        //onemogucavamo unos brojeva vecih od 10 ili manjih od 1
      }while(broj>10 || broj<1);
      cout<<"\nMAIN::prije poziva rekurzivne funkcije! \n\n";
      rekFunkcija(broj); //poziv funkcije
      cout<<"\nMAIN::poslije poziva rekurzivne funkcije! \n\n";
}
```

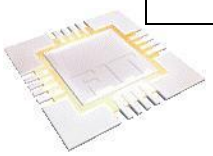
Kao što se može vidjeti, unutar `main` funkcije je deklarisana varijabla cjelobrojnog tipa (varijabla `broj`) koja služi za čuvanje vrijednosti koju je korisnik unio. Odmah nakon validnog unosa (1-10), vrijednost varijable `broj` se kao argument predaje funkciji `rekFunkcija`.

Na početku definicije funkcije `rekFunkcija` se provjera da li je vrijednost primljenog argumenta veća od nula (0). Ukoliko je uslov ispunjen, trenutna vrijednost argumenta se ispisuje, a nakon toga dekrementira (`broj--`). Neposredno nakon dekrementiranja vrši se ponovni poziv funkcije, te se na taj način implementira rekurzija. Ono što je bitno primijetiti je da se pri ponovnom pozivu funkcije kao argument predaje *umanjena* vrijednost argumenta `broj`, a ne vrijednost koju je korisnik proslijedio iz `main` funkcije. Dekrementiranjem vrijednosti primljenog argumenta se izbjegava mogućnost pojave beskonačne ili nekontrolisane rekurzije. U slučaju da se postavi komentar na liniju koda koja vrši dekrementiranje (`//broj--`) i ponovo pokrene program trebalo bi doći do pojave beskonačne rekurzije koja će nakon određenog broja ponavljanja biti automatski prekinuta.

U slučaju da prilikom sljedećeg poziva funkcije uslov (`broj>0`) ne bude ispunjen, tj. ukoliko vrijednost argumenta `broj` nije veća od nula (0), neće doći do ponovnog poziva funkcije čime je onemogućena dalja rekurzija.

U prethodnom primjeru, dekrementiranje argumenta `broj` se moglo izvršiti neposredno pri pozivu funkcije.

```
      if (broj>0){
            cout<<" Trenutna vrijednost varijable broj je:"<< broj <<endl;
            rekFunkcija(--broj);  // ili   rekFunkcija(broj-1);
      } //kraj if-a
```



Umanjivanje vrijednosti argumenta pri samom pozivu funkcije se baš ne preporučuje, ali to smo ipak uradili zbog demonstracije različitih načina rješavanja istog problema.

Šta bi se desilo da se kojim slučajem umjesto pre-dekrementiranja (`--broj`) koristilo post-dekrementiranje (`broj--`)? Naravno, prethodna promjena bi uzrokovala pojavu beskonačne rekurzije. Glavni razlog tome je činjenica da bi se pri svakom narednom pozivu funkcije predavala identična vrijednost argumenta zbog toga što se vrijednost argumenta post-dekrementira tek nakon poziva funkcije.

Za potpuno razumijevanje načina na koji funkcioniše rekurzija bitno je znati da se prilikom svakog novnog poziva funkcije (rekurzije) izvršava druga verzija (kopija) funkcije, a ne originalna funkcija. Analogno tome, lokalne varijable u drugoj verziji (kopiji) funkcije su apsolutno nezavisne od lokalnih varijabli originalne funkcije.

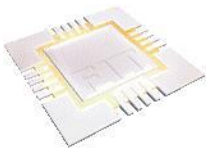
U slučaju da vas je malo zbunila prethodna konstatacija, nema potrebe da se brinete. Da bismo pojasnili izneseno koristit će se definicija funkcije iz prethodnog primjera. Dakle, `main` funkcija ostaje ista kao i u prethodnom primjeru, samo će na funkciji `rekFunkcija` biti napravljene sljedeće izmjene:

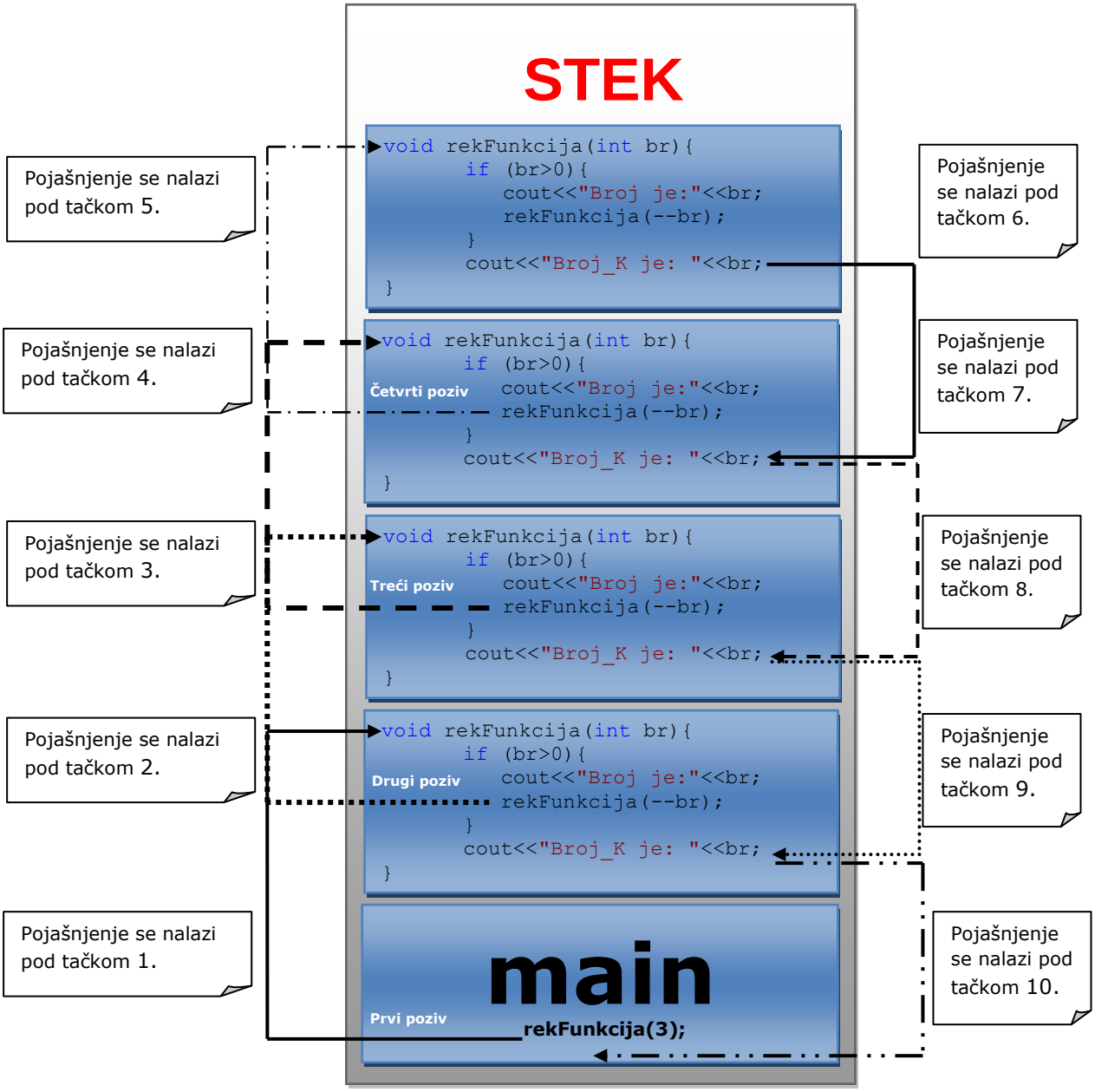
```
void rekFunkcija(int broj){
      if (broj>0){
            cout<<"Trenutna vrijednost varijable broj je:"<<broj<<endl;
            rekFunkcija(--broj); //<--skracena verzija
      }
      cout<<"Vrijednost varijable broj je: "<<broj<<endl; //<--nova linija
}
```

Pod uslovom da je varijabla `broj` inicijalizovana vrijednošću `3`, izlaz iz programa bi trebao biti sljedeći:

```
                ::REKURZIJA::

Unesite broj od 1 do 10: 3

MAIN::prije poziva rekurzivne funkcije!

Trenutna vrijednost varijable broj je:3
Trenutna vrijednost varijable broj je:2
Trenutna vrijednost varijable broj je:1
Vrijednost varijable broj je: 0
Vrijednost varijable broj je: 0
Vrijednost varijable broj je: 1
Vrijednost varijable broj je: 2

MAIN::poslije poziva rekurzivne funkcije!

Press any key to continue . . .
```

Ukoliko se prvi put susrećete sa pojmom rekurzije, vjerojatno ste se odmah zapitali: zašto se prilikom drugog ispisa (unutar iste funkcije) vrijednost varijable `broj` povećava? Odgovor na prethodno pitanje leži u činjenici da se pri svakom pozivu funkcije (rekurziji) na steku rezerviše novi memorijski prostor, što uzrokuje međusobnu nezavisnost lokalnih varijabli. Kako bi sve to bilo jasnije, izvršenje prethodnog programa će biti predstavljeno narednom slikom:





U nastavku slijedi detaljnije pojašnjenje načina izvršavanja programa, pa krenimo redom:

1. Unutar `main` funkcije korisniku je omogućeno da unese neki `broj`. Zbog veličine prethodnog dijagrama pretpostavljeno je da je korisnik unio vrijednost 3. Nakon unosa vrijednosti uslijedio je *prvi poziv* funkcije `rekFunkcija`. Samim pozivom, na steku je rezervisan prostor dovoljan za izvršenje nove funkcije[1].
2. Na samom početku izvršenja funkcije slijedi provjera vrijednosti primljenog parametra (u našem slučaju vrijednost je 3), tačnije provjerava se da li je vrijednost parametra veća od nule (`3>0`). Pošto je uslov zadovoljen, slijedi ispis vrijednosti (`Broj je: 3`), te ponovni (*drugi*) poziv funkcije sa pre-dekrementiranom vrijednošću parametra (nakon dekrementiranja vrijednost je 2).
3. *Drugi poziv* funkcije uzrokuje rezervisanje novog prostora na steku, te početak izvršenja funkcije. Pošto je vrijednost primljenog parametra (2) veća od nula,

[1] Pokretanjem programa na steku je rezervisan prostor za izvršenje `main` funkcije

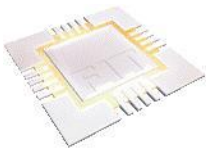
slijedi njen ispis (`Broj je: 2`), te ponovni (*treći*) poziv funkcije sa pre-dekrementiranom vrijednošću parametra (vrijednost parametra je sada 1).

4. Nakon rezervisanja prostora za izvršenje nove funkcije slijedi provjera uslova. Pošto je uslov ponovo zadovoljen (`1>0`), funkcija ispisuje poruku (`Broj je: 1`), te ponovo poziva funkciju sa pre-dekrementiranom vrijednošću parametra (ovoga puta 0).
5. *Četvrtim pozivom* vrijednost predanog parametra je 0. Pošto uslov nije ispunjen (`0>0`), neće doći do novog poziva funkcije, te funkcija ispisuje drugu poruku `Broj_K je: 0`.
6. Nakon ispisivanja prethodne poruke (`Broj_K je: 0`), posljednje pozvana funkcija (funkcija pozvana *četvrtim pozivom*) se završava i oslobađa prostor koji je bio alociran za potrebe njenog izvršenja.
7. Pošto je posljednja funkcija završila svoj životni ciklus, izvršenje programa se nastavlja na mjestu odakle je ona pozvana tj. na mjestu odakle je uslijedio četvrti poziv. Nakon linije koda koja je uzrokovala posljednji poziv slijedi ponovni ispis vrijednosti parametra. U trenutku poziva vrijednost parametra je bila 0, pa će funkcija ponovo ispisati poruku `Broj_K je: 0`. Treba napomenuti činjenicu da je funkcija (iz koje je uslijedio *četvrti poziv*) primila vrijednost parametra 1, ali je prilikom poziva funkcije (*četvrtog poziva*) izvršeno njegovo dekrementiranje. Nakon ispisa poruke, funkcija se završava i time oslobađa alocirani prostor.
8. Po završetku funkcije iz koje je uslijedio *četvrti poziv*, izvršenje programa se nastavlja na mjestu *trećeg poziva*. Identično prethodnom scenariju, funkcija ispisuje vrijednost parametra `Broj_K je: 1` i time završava svoje postojanje. Slijedi oslobađanje alociranog prostora i povratak na mjesto odakle je funkcija pozvana.
9. Prethodna funkcija je pozvana koristeći *drugi poziv*, te se izvršenje programa nastavlja neposredno ispod te linije. Funkcija ispisuje poruku `Broj_K je: 2` nakon čega slijedi dealokacija memorijskog prostora i povratak u `main` funkciju odakle je uslijedio *prvi poziv* funkcije.
10. Po povratku u `main` funkciju slijedi ispisivanje poruke „`MAIN::poslije poziva rekurzivne funkcije!`“ i time program završava.

Prethodni primjer oslikava princip funkcionisanja rekurzije, te njegovo razumijevanje predstavlja osnov za uspješno praćenje ostatka materijala. Ukoliko niste shvatili način izvršenja prethodnog programa svakako je preporučljivo da nekoliko puta preradite zadatak uz pomoć navedenih pojašnjenja. Ukoliko želite provjeriti da li ste shvatili prethodni primjer, pokušajte predvidjeti ispis sljedeće funkcije:

```
void rekFunkcija(int broj){
	if (broj>0){
		cout<<"Trenutna vrijednost varijable broj je:"<<broj<<endl;
		rekFunkcija(--broj);
	}
	else //<--dodato
		cout<<"Vrijednost varijable broj je: "<<broj<<endl;
}
```

Rješenje rekurzivne funkcije koje je korišteno u prethodnom dijelu materijala svakako izvršava svoj zadaću, ali ne predstavlja baš reprezentativan primjer. Naime, uobičajeno je da se bazni slučaj stavi na početku svake rekurzivne funkcije i na taj način onemogući nepotrebno izvršavanje funkcije. Malo reprezentativnije rekurzivno rješenje bi se moglo napisati na sljedeći način:



```
void rekFunkcija(int broj) {
	if (broj==0)//bazni slucaj
		return;
	cout<<"Trenutna vrijednost varijable broj je:"<<broj<<endl;
	rekFunkcija(broj-1);
}
```

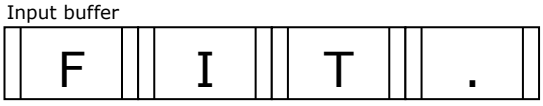
Naravno, odmah se nameće pitanje: kako je moguće da `void` funkcija vraća neku vrijednost? To svakako ne bi trebalo biti moguće, ali u prethodnom primjeru i nije vraćena bilo kakva vrijednost. U ovom slučaju, ključna riječ `return` ima zadatak samo da prekine izvršenje funkcije, a ne da vrati neku vrijednost.

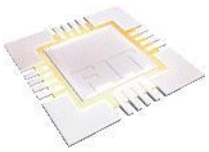
Zadatak 2. Koristeći rekurziju, napisati program koji korisniku omogućava da unese željeni tekst. Uslov za završetak programa je da uneseni tekst sadrži najmanje jednu tačku (.). Nakon unosa, program treba obrnutim redoslijedom ispisati sve znakove koji su uneseni do prvog znaka tačke.

```
#include <iostream>
using namespace std;

void prikaziUnazad(){
	char znak;
	cin>>znak;
	if(znak!='.'){
		prikaziUnazad();
		cout<<znak;
	}
	else
		cout<<"\nObrnutim redosljedom ste unijeli: ";
}
void main(){
	cout<<"\t\t::REKURZIJA::\n\n";
	cout<<"Unesite vas tekst: ";
	prikaziUnazad();
	cout<<"\n\nMAIN::POSLIJE poziva rekurzivne funkcije\n\n";
}
```

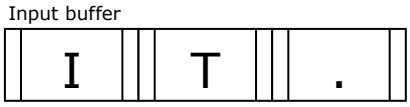
Prethodni program počinje ispisivanjem poruke kojom se od korisnika traži da unese željeni tekst, nakon čega se poziva funkcija `prikaziUnazad`. Pomenuta poruka je ispisana u `main` funkciji sa osnovnim ciljem da se prilikom rekurzije onemogući višestruko ponavljanje poruke.

Unutar funkcije `prikaziUnazad` deklarisana je lokalna varijabla `znak` koja će prilikom svake rekurzije čuvati jedan od karaktera koje je unio korisnik. Ovo možda izgleda malo zbunjujuće, pa je zbog toga bitno da razumijete način funkcionisanja *input buffera*. Naime, input buffer predstavlja dio memorije koji služi za čuvanje ulaznih vrijednosti sa tastature. Uz pretpostavku da je korisnik unio karaktere „`FIT.`“ i pritisnuo tipku `Enter`, sadržaj buffera će biti sljedeći:





Kada korisnik pritisne tipku `Enter`, komanda `cin` (govorimo o prvom izvršenju funkcije `prikaziUnazad`) će iz buffera preuzeti prvi karakter (u našem slučaju slovo **F** jer se u varijablu tipa `char` može pohraniti samo jedan znak) i pohraniti ga u varijablu `znak`. Odmah nakon toga se vrši provjera uslova tj. da li je `znak` (ne)jednak znaku tačka (**.**). Ako je uslov ispunjen (ako `znak` nije jednak znaku tačka) funkcija se poziva ponovo. Prilikom sljedećeg poziva funkcije sadržaj input buffera je sljedeći (pošto je prethodna funkcija iskoristila slovo F):



Na početku izvršenja sljedeće funkcije, komanda `cin` će preuzeti naredni karakter (slovo **I**) i pohraniti ga u varijablu `znak`. Ovdje se nameće pitanje: zašto komanda `cin`, prilikom sljedećeg izvršenja funkcije, nije tražila ponovni unos sa tastature? Odgovor na prethodno pitanje leži u činjenici da komanda `cin` neće zahtijevati interakciju sa korisnikom (misli se na unos) sve dok u input bufferu postoji bilo kakav sadržaj. Pošto je u našem slučaju sadržaj input buffera „IT.", komanda `cin` preuzima sljedeći znak tj. slovo **I**.

Izvršenje programa se na taj način nastavlja sve do momenta dok varijabla `znak` ne bude inicijalizovana vrijednošću karaktera tačka (**.**). U tom momentu, ponovni poziv funkcije je onemogućen i rekurzija se završava. Na osnovu iznesenog, moguće je postaviti nekoliko pitanja:

1. Šta će se desiti u slučaju kada se unese neki tekst bez tačke i pritisne tipka `Enter`?
2. Zašto se poruka: `Obrnutim redoslijedom ste unijeli sljedece znakove:`, koja se nalazi unutar `else` bloka, prikazuje prije teksta (obrnutog redoslijeda) kojeg je unio korisnik?

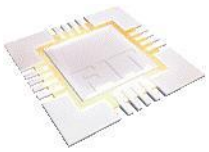
Zaista se nadam da ćete biti u stanju samostalno odgovoriti na postavljena pitanja.

Za prethodni program je karakteristično da ispisuje samo one znakove koji su uneseni do prvog znaka tačke. Svi znakovi uneseni poslije znaka tačke se ignorišu. Također, program ignoriše sve razmake među znakovima, što je moguće riješiti korištenjem funkcije `cin.get()`. Ova funkcija učitava sadržaj buffera znak po znak (čak i ako se radi o razmaku) i njihovu vrijednost dodjeljuje nekoj varijabli (u našem slučaju varijabli `znak`). Kao ilustraciju napravite sljedeće izmjene u definiciji funkcije `prikaziUnazad`:

```
//..
char znak;
cin.get(znak);//<--izmjena
if(znak!='.'){
//..
```

Funkcija `main` je ostala nepromijenjena, tako da odmah možete pokrenuti program kako biste vidjeli rezultat napravljenih izmjena. Za vježbu, prethodni program pokušajte prepraviti na način da uneseni tekst ne ispisuje unazad već onakvog kakav je unesen.

Zadatak 3. Koristeći rekurziju, napisati program koji od korisnika traži da unese neko slovo. Nakon unosa, program treba da ispiše sva slova koja se nalaze između unesenog i slova '`A`'. Pored oznake slova, program treba da ispiše i njima pripadajući `ASCII` kod.



Također, potrebno je ispisati odgovarajuću poruku u slučaju da korisnik unese znak koji ima ASCII vrijednost manju od slova 'A'.

Napomene:

- ✓ Kada se kaže manje ili veće slovo, naravno, misli se na vrijednost ASCII koda koji je asociran sa određenim slovom.
- ✓ ASCII kod za malo slovo **a** je 97, a za veliko slovo **A** je 65.

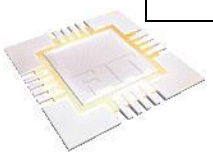
```
#include <iostream>
using namespace std;

void prikaziSlovo(char slovo){

	if(slovo>='A')
	{
		cout<<"\nSlovo je: \'"<<slovo<<"\' | ASCII code: "<<(int)slovo;
		slovo--;			//dekrementiranje
		prikaziSlovo(slovo);  //ponovni poziv funkcije
	}
	else
		cout<<"\n\nNije dozvoljeno prikazivanje znakova"
			" koja imaju ASCII kod manji od 65!"<<endl;
}
void main(){
	char slovo;
	cout<<"\t\t::REKURZIJA\n\n";
	cout<<"Unesite neko slovo: ";
	cin>>slovo;
	cout<<"\n\n::Pozivam funkciju::\n\n";
	prikaziSlovo(slovo);
	cout<<"\n\n::Ponovo u Main funkciji::\n\n";
}
```

Način izvršenja prethodnog programa bi trebao biti u potpunosti jasan. Jedina novina, u odnosu na prethodni dio materijala, je eksplicitna konverzija koja se koristi prilikom ispisa ASCII koda određenog karaktera. Pri korištenju eksplicitne konverzije, u zagradi prije varijable, navodi se tip podatka u koji želimo konvertovati određenu vrijednost. Ukoliko ste zaboravili šta podrazumijeva eksplicitna konverzija koja je obrađena u okviru predmeta Programiranje I pogledajte sljedeći primjer:

```
char znak='b';
int broj=(int)znak;
cout<<"Znak: "<<znak;  //ispisuje: b
cout<<"\nBroj: "<<broj; //ispisuje: 98
```

U slučaju da imate problema sa razumijevanje načina organizacije znakova i njima pripadajućih ASCII kodova, bilo bi dobro da napravite program koji će, za određeni opseg, ispisati znakove i njihove ASCII kodove. U nastavku je prikazan primjer programa koji korisniku omogućava da definiše opseg znakova koje želi ispisati.

```
#include <iostream>
using namespace std;
void main(){
	int poc=0;
	int kraj=0;
	do{
		system("cls");//cisti sadrzaj komandnog prozora
```



```
            cout<<"Unesite pocetnu vrijednost: ";
            cin>>poc;
            cout<<"Unesite krajnju vrijednost: ";
            cin>>kraj;
      }while(poc>=kraj);
      cout<<"------------------------------";
      //petlja se vrti od pocetne do krajnje vrijednosti
      for(poc;poc<=kraj;poc++)
            cout<<"\nZnak: "<<(char)poc<<" posjeduje ASCII kod: "<<poc;
      cout<<"\n------------------------------\n";
}
```

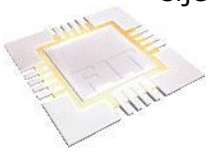
Za uspješno izvršenje prethodnog programa početna vrijednost mora biti manja od krajnje. Ukoliko to nije slučaj, program će od korisnika zatražiti ponovni unos. Upravo zbog toga je korištena komanda `system("cls")` koja ima osnovni zadatak da pri ponovnom unosu očisti sadržaj komandnog prozora u kojem se program izvršava.

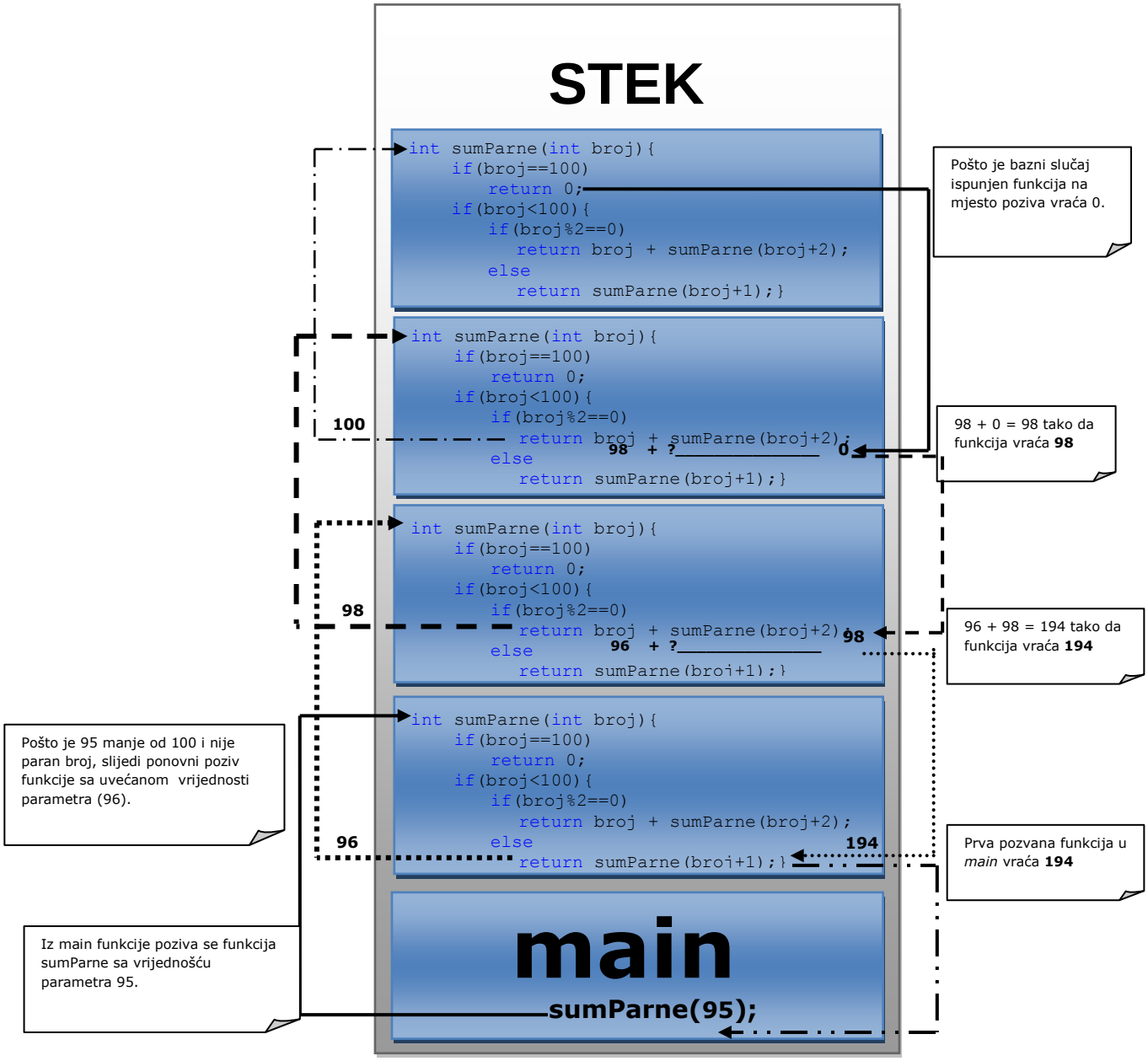
U prethodnom dijelu materijala je uglavnom bilo riječi o rekurzivnim funkcijama koji ne vraćaju nikakvu vrijednosti. Takve funkcije su dosta rijetke, a mi smo ih koristili samo za demonstraciju osnovnih karakteristika rekurzije. Zbog svega navedenog, nastavak poglavlja je posvećen rekurzivnim funkcijama koje vraćaju vrijednost i sa kojima ćete se mnogo češće susretati.

Zatadak 4. Napraviti program koji korisniku omogućava da unese neki broj. Nakon unosa, program treba da, koristeći rekurzivnu funkciju, izračuna sumu svih parnih brojeva koji se nalaze između unesenog i broja 100. Prilikom izrade programa voditi računa o tome da korisnik može unijeti broj koji je veći ili manji od broja 100.

```
#include <iostream>
using namespace std;

int sumParne(int broj){
      if(broj==100)//bazni slucaj
            return 0;
      if(broj<100){ //ako je broj manji od 100
            if(broj%2==0)//ako je paran
                  return broj + sumParne(broj+2);
            else//ako je neparan
                  return sumParne(broj+1);
      }
      else{//ako je broj veci od 100
            if(broj%2==0)//ako je paran
                  return broj + sumParne(broj-2);
            else//ako je neparan
                  return sumParne(broj-1);
      }
}
void main(){
      int broj=0;
      cout<<"Unesite neki broj: ";
      cin>>broj;
      int suma=sumParne(broj);
      cout<<"Suma parnih brojeva je: "<<suma<<endl;
}
```

Ako niste sigurni u način izvršenja prethodnog programa, pokušajte samostalno konstruisati sliku izvršenja programa na steku. Ukoliko ne uspijete, poslužite se sljedećom slikom.





Zbog veličine, prethodni dijagram je kreiran pod pretpostavkom da će korisnik unijeti broj koji je manji od broja 100. Upravo zbog toga, kod koji je prikazan u dijagramu, manipuliše samo brojevima koji su manji od 100. Također, u prethodnom programu, parne vrijednosti su uvećavane/umanjivane za dva čime se dvostruko smanjio broj rekurzivnih poziva. To je moguće učiniti prvenstveno zbog činjenice da je bazni slučaj paran broj što potvrđuje i beskonačna rekurzija dobivena sljedećom definicijom funkcije:

```
int sumParne(int broj){
      if(broj==101)//bazni slucaj neparan
            return 0;
      if(broj<101){ //ako je broj manji od 101
            if(broj%2==0)//ako je paran
                  return broj + sumParne(broj+2);
            else//ako je neparan
                  return sumParne(broj+1);
      }
```

```
	else{//ako je broj veci od 100
		if(broj%2==0)//ako je paran
			return broj + sumParne(broj-2);
		else//ako je neparan
			return sumParne(broj-1);
	}
}
```

Pokušajte nacrtati izvršenje funkcije na steku ili nakon baznog slučaja dodajte liniju koda koja će ispisivati vrijednost parametra `broj`.

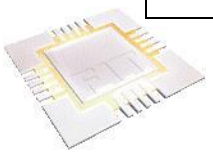
```
int sumParne(int broj){
	if(broj==101)
		return 0;
	cout<<"Broj je:"<<broj<<endl;
	if(broj<101){
		//...
```

Rekurziji kao i svakoj drugoj funkciji je moguće proslijediti niz vrijednosti, pa je zbog toga u nastavku prikazan program koji putem rekurzivne funkcije izračunava sumu vrijednosti svih članova niza.

```
#include <iostream>
using namespace std;

int sumiraj(int niz[], int max){
	if(max==0)
		return niz[max];
	return niz[max] + sumiraj(niz,max-1);
}

void main(){
	const int max = 5;
	int niz[max];
	for(int i=0;i<max;i++){
		cout<<"Unesite "<<i+1<<". element:";
		cin>>niz[i];
	}
	cout<<"Suma svih clanova je: "<<sumiraj(niz,max-1)<<endl;
}
```

Prethodni program je dosta jednostavan. Jedino se postavlja pitanje: zašto se prilikom poziva funkcije `sumiraj` vrijednost konstante `max` umanjila za jedan? Pokušajte samostalno odgovoriti na postavljeno pitanje. Ukoliko ste pronašli logičan odgovor, prethodnom programu dodajte rekurzivnu funkciju koja će računati sumu svih parnih članova niza. Ako ste u dilemi kako napraviti pomenutu funkciju, u nastavku je prikazan kompletan primjer.

```
#include <iostream>
using namespace std;

int sumiraj(int niz[], int max){
	if(max==0)
		return niz[max];
	return niz[max] + sumiraj(niz,max-1);
}
int sumirajParne(int niz[], int max){
	if(max<0)
		return 0;
```



```
	if(niz[max]%2==0)
		return niz[max] + sumirajParne(niz,max-1);
	else
		return sumirajParne(niz,max-1);
}
void main(){
	const int max = 5;
	int niz[max];
	for(int i=0;i<max;i++){
		cout<<"Unesite "<<i+1<<". element:";
		cin>>niz[i];
	}
	cout<<"Suma svih clanova je: "<<sumiraj(niz,max-1)<<endl;
	cout<<"Suma parnih brojeva je: "<<sumirajParne(niz,max-1)<<endl;
}
```

Za kraj ovog materijala, samostalno pokušajte napisati program koji će korištenjem rekurzije omogućiti konverziju unsenog broja u njegov binarni ekvivalent. Također, korištenjem `bitset`-a možemo izvršiti provjeru ispravnosti rada rekurzivne funkcije.

```
#include <iostream>
#include <bitset>
using namespace std;
//ukoliko ne razumijete rjesenje ovog zadatka ponovo pogledajte
//zadatak u kome smo uneseni tekst ispisivali unazad
void PretvoriUBinarni(int broj){
	if(broj<=1) {//bazni slucaj
		cout<<broj;
		return;
	}
	int bit = broj%2;
	broj/=2;
	PretvoriUBinarni(broj);
	//kako bismo dobili ispravan prikaz rezultata
	//bite ispisujemo u silaznom toku rekurzije
	cout<<bit;
}
void main(){
	int broj=0;
	char crtice[] = "\n---------------------------------\n";
	do{
		cout<<crtice<<"BINARNA KONVERZIJA::REKURZIJOM"<<crtice;
		cout<<"Unesite broj: ";
		cin>>broj;
//bitset omogucava da dobijemo binarni ekvivalent odredjene vrijednosti
//vrijednost,broj 8 predstavlja broj bita u koje ce unesena vrijednost biti
//konvertovana, pa ako zelite konvertovati vece vrijednosti povecajte broj
//bita
		bitset<8> binarno(broj);

		cout<<crtice<<"BROJ:\t\t"<<broj<<"\nBINARNO:\t";
		PretvoriUBinarni(broj);
		cout<<"\nPROVJERA:\t"<<binarno;
		cout<<crtice;
	}while(broj>0);
}
```

Za vježbu, pokušajte rekurzivnom funkcijom uraditi konverziju binarne vrijednost u decimalnu.