Pokazivači

Memoriju u računaru možemo da zamislimo kao traku sa „kućicama“ u kojima su smeštene nule i jedinice, i gde svaka „kućica“ ima svoj redni broj.

Memorijska lokacija	…	5	6	7	8	9	10	11	12	…
Vrednost	…	0	0	1	1	1	1	0	0	…

Svaka „kućica“ sa cifrom 0 ili 1 se zove bit, a 8 bitova čine bajt. Znamo da je celobrojni tip *int* u programskom jeziku C/C++ veličine 4 bajta, tj. sačinjen je od 32 bita koji se u memoriji nalaze jedan pored drugog. Ukoliko znamo gde celobrojni tip *int* počinje u memoriji, mi na osnovu toga možemo da saznamo ceo broj, tj. dovoljno je da uzmemo naredna 32 bita (broj je u memoriji smešten u binarnom sistemu, tj. u sistemu sa osnovom 2).

U većini programskih jezika postoje osnovni tipovi promenljivih kao što su celobrojni brojevi, realni brojevi, nizovi, itd., međutim nama je omogućeno i da pravimo složenije strukture, slogove, koje će nam omogućiti lakši rad. Primer jednog sloga:

```
================================================================
01     slog Osoba
02            ime, prezime: array [20] of type char
03            godina: integer
04            JMBG: array [13] of type char
05     end slog
================================================================
```

Veličina sloga je zbir pojedinačnih veličina promenljivih u slogu.

Promenljive u slogu se u memoriji nalaze jedna pored druge. Na osnovu informacije gde počinje promenljiva tipa Osoba i na osnovu toga što znamo strukturu sloga Osoba mi možemo da saznamo vrednosti svih promenljivih koje taj slog sadrži. Na sledećoj šemi je prikazan primer kako je promenljiva tipa Osoba predstavljena u memoriji.

			Osoba													
			Ime: char[20]			Prezime: char[20]			Godina: integer			JMBG: char[13]				
Adresa	…	11	12	…	171	172	…	331	332	…	363	364	…	467	468	…
Vrednost	…	1	0	…	1	1	…	0	1	…	0	1	…	1	1	…

Pokazivači su strukture koje „pokazuju“ na neku promenljivu u memoriji, tj. oni kao vrednost čuvaju memorijsku lokaciju gde počinje struktura na koju pokazuju. U primeru iz gore navedene tabele pokazivač na tip podataka Osoba bi sadržao vrednost 12.

Pokazivači su promenljive veličine 4 bajta, te nekad nam je zgodnije da pri pozivu procedure prosledimo samo pokazivač na neki objekat, umesto celog objekta koji često zauzima mnogo više memorije. Pokazivač na neki objekat se dobija tako što uzmemo memorijsku lokaciju od promenljive, ovde ćemo tu vrednost označavati sa "address imePromenljive".

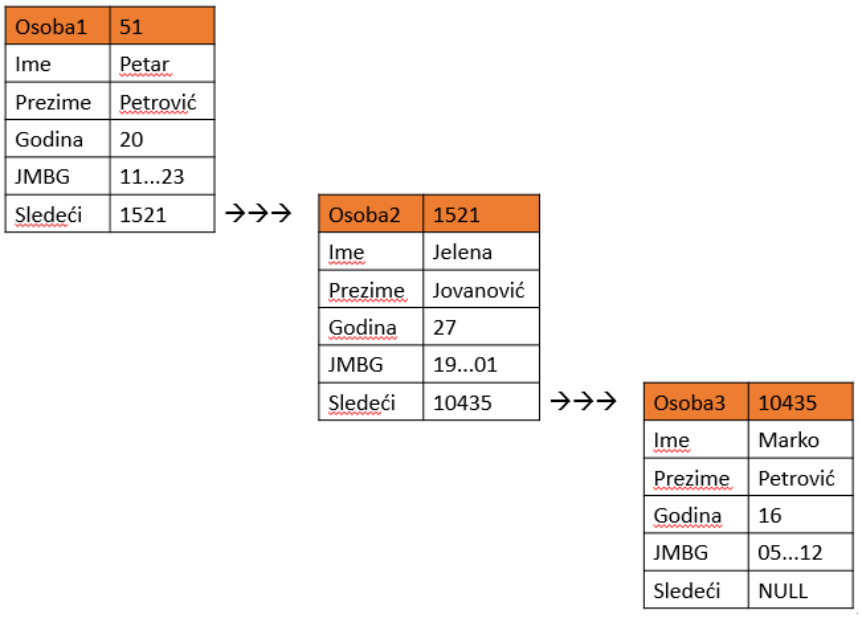
Nekada nismo sigurni koliki niz treba da napravimo pre izvršavanja programa, a ukoliko koristimo samo statičke promenljive, mi pre izvršavanja programa moramo da navedemo dužinu niza. Prednost pokazivača u odnosu na standardne tipove podataka je ta što se uz pomoć njih mogu praviti dinamičke promenljive, tj. promenljive koje su alocirane u toku izvršavanja programa.

Kao što je već rečeno, vrednost pokazivača je memorijska lokacija gde počinje struktura na koju pokazuje, međutim moguće je pristupiti i vrednostima te strukture. Ovde ćemo tu vrednost označavati sa *„value pokazivač"*. Ponovo ćemo za primer iskoristiti slog Osoba. Neka postoji promenljiva *prom* koja pokazuje na strukturu Osoba. Promenjiva tipa Osoba se dobija od pokazivača sa *value prom*, pa dalje tu promenljivu možemo da koristimo kao promenljivu tipa Osoba, tj. možemo da pristupamo elementima (value *prom).godina, (value prom).JMBG,* itd. Da ne bismo stalno pisali *(value prom).godina*, mi ćemo koristiti oznaku *prom->godina.*

Sa *„prom = new nazivStrukture"* ćemo označavati pravljenje objekta strukture *nazivStrukture* u memoriji, gde će posle poziva pokazivač *prom* pokazivati na taj novi objekat koji je napravljen u memoriji. Dok ćemo sa *„delete prom"* označavati brisanje objekta u memoriji na koji pokazuje pokazivač *prom.*

Liste

Neformalno ćemo reći da liste simuliraju nizove proizvoljne dužine. Liste su sačinjene od slogova koji su povezani vezama preko pokazivača, tj. svaki slog pokazuje na sledeći u listi. Primer se može videti na slici 1.



Slika 1

Atribut *Sledeci* sloga Osoba je pokazivač na slog Osoba, tj. on pokazuje gde se u memoriji nalazi sledeći element u listi. Kao što možemo da vidimo sa slike 1, Osoba1.Sledeći ima vrednost 1521 koja predstavlja poziciju u memoriji gde se nalazi promenljiva Osoba2. Kad je vrednost pokazivača stavljena na *NULL* to znači da ne pokazuje ni na šta.

Ovde ćemo kao primer rada sa listama pokazati kako se prave liste, gde ćemo svaki novi element postavljati na početak liste, a u zadacima ćemo kasnije odraditi komplikovanije zadatke. Napravićemo novi slog List koji će sadržati atribute *osoba* koji je tipa Osoba i *next* koji je pokazivač na tip Osoba. U atributu *next* ćemo čuvati informaciju gde se nalazi sledeći element u listi.

```
==============================================================
01    struct List
02          person: Person
03          next: pointer List
04    end struct
05
06    function AddElement( ref list, pointer newPerson )
07          currentList = list
08          list = new List // napravimo u memoriji novi objekat tipa List
09          list->person = value newPerson
10          list->next = currentList // povežemo prvi element sa prethodno
prvim
11    end function
==============================================================
```

Funkcija *AddElement* pravi novi element sa vrednostima atributa koji su prosleđeni funkciji i postavlja ga za prvi element liste, i povezuje taj novi napravljeni element sa prethodnim prvim elementom liste.

Kao što je rečeno, prednost listi u odnosu na nizove je ta što lista može da bude proizvoljne dužine i nije potrebno da navedemo dužinu liste pre izvržavanja programa. Međutim liste imaju i mane, jedna od bitnijih je ta što ne možemo da pristupimo svakom elementu u $O(1)$. Kod listi mi imamo samo pokazivač na prvi element u listi, tako da bismo pristupili n-tom elementu, mi moramo da idemo redom kroz sve element $1, 2, 3, \ldots, n$.