Programiranje 2

Laboratorijske vježbe

LV2
Polje pokazivača, pokazivači i višedimenzionalna polja

Fakultet elektrotehnike računarstva i informacijskih tehnologija Osijek
Kneza Trpimira 2b
www.ferit.unios.hr

Uvod

Svaka varijabla predstavlja spremnik određenog tipa podatka koji zauzima određeni broj bajtova u memoriji računala. U varijablu se može spremiti samo jedna vrijednost, a to je numerička konstanta i ovisno o tipu podatka može biti cjelobrojna ili realna. Ovisno o tipu podatka varijable, koristit će se određeni broj bitova za prikaz numeričke konstante, te različite kombinacije bitova predstavljaju cjelobrojnu ili realnu vrijednosti. Svaka varijabla ima memorijsku adresu kojom se definira njezin početak u memoriji računala. Memorijska adresa ili referenca svake varijable može se dohvatiti pomoću adresnog operatora (&).

Pokazivač je varijabla u koju se može spremiti samo jedna vrijednost, a to je memorijska adresa varijable. Kako je pokazivač varijabla potrebno je naglasiti kako i on zauzima određeni broj bajtova u memoriji i da počinje na određenoj memorijskoj adresi. Uobičajeno je da deklarirani pokazivač zauzima 4 bajta memorije neovisno kojim je tipom podatka deklariran, ali to ovisi od kompilatora i sustava za koji se programira. Memorijska adresa je predstavljena dugačkim cijelim pozitivnim brojem, može se prikazati u skraćenom heksadekadskom obliku. Opći oblik deklaracije pokazivača:

```
tip_podatka *ime_jednostrukog_pokazivaca;
```

Tip podatka pokazivača treba odgovarati tipu podatka varijable na koju će se pokazivač usmjeriti. Prilikom deklaracije pokazivača kaže se da pokazivač pokazuje na tip podatka na koji je deklariran. Jedino pokazivač koji je deklariran kao *void** pokazivač (generički pokazivač), može se usmjeriti na bilo koji tip podatka, ali je naknadno potrebno promijeniti tip podatka generičkom pokazivaču pomoću operatorom pretvaranja (engl. *cast operator*) u tip podatka varijable na koju je usmjeren kako bi ste pristupilo vrijednosti unutar varijable. Operator pretvaranja privremeno mijenja tip podatka varijabli u izrazu u liniji kôda gdje se taj izraz nalazi, a prije i poslije primjene operatora pretvaranja varijabla ima svoj tip podatka kako je u početku bila deklarirana.

Primjer 1: Pridruživanje memorijske adrese varijable generičkom pokazivaču, te indirektni pristup sadržaju varijable preko generičkog pokazivača.

```
#include<stdio.h>

int main(void) {
	int a = 5;
	void *genPok = &a;

	printf("a = %d\n", *(int*)genPok);

	return 0;
}
```

Operator indirektnog pristupa ili dereferenciranja (*) u deklaraciji označava da je riječ o pokazivačkoj varijabli, dok njegova primjena nad pokazivačem poslije deklaracije označava indirektni pristup sadržaju varijable koji se nalazi u memorijskoj adresi koja je spremljena u pokazivač.
Kada se želi indirektno preko pokazivačke varijable promijeniti sadržaj ili pročitati sadržaj varijable na koju je pokazivač usmjeren, upravo na temelju tipa podatka kojim je deklariran pokazivač kompilator

će znati prilikom prevođenja koliko bajtova podataka treba prepisati ili pročitati iz varijable indirektno pomoću pokazivača.

Prilikom deklaracije pokazivača dobra je praksa inicijalizirati pokazivač na određenu memorijsku adresu, npr. na vrijednost simboličke konstante *NULL*, ako se nema potreba odmah usmjeriti pokazivač na varijablu. *NULL* vrijednost je konstanta s vrijednosti memorijske adrese nula koja je definirana u nekoliko standardnih biblioteka (*<stdio.h>*, *<stdlib.h>*). Kod većine operacijskih sustava, programima je zabranjen pristup memoriji na adresi nula jer je ta memorijska adresa rezervirana od strane operacijskog sustava. Memorijska adresa nula signalizira kako pokazivač nije usmjeren prema dostupnoj memorijskoj adresi.

Primjer 2: Promjena sadržaja varijable pokazivačem primjenom indirektnog pristupa.

```c
#include<stdio.h>

int main(void) {

	int a = 5;
	int *pok = &a;
	//int *pok = NULL; inicijalizacija pokazivaca
	//pok = &a; naknadna dodjela vrijednosti pokazivacu

	printf("a = %d\n", a);
	printf("a = %d\n", *pok);
	*pok = 6;
	printf("a = %d\n", a);
	printf("a = %d\n", *pok);

	return 0;
}
```

Preporuka je ne koristiti operator indirektnog pristupa nad pokazivačem koji nije inicijaliziran ili koji nije konkretno usmjeren na varijablu, zato što takvi ne inicijalizirani i ne usmjereni pokazivači prilikom deklaracije mogu sadržavati nasumičnu vrijednost, tj. nasumičnu memorijsku adresu, te pristupanju sadržaju takve memorijske adrese može izazvati neočekivano ponašanje programa.

Polja i pokazivači su u srodstvu, gdje ime polja predstavlja memorijsku adresu cijelog polja, odnosno memorijsku adresu prvog elementa polja. Ime polja je ujedno i konstantni jednostruki pokazivač na prvi element polja kojeg nije moguće preusmjeriti.

Primjer 3: Pristup elementima jednodimenzionalnog polja pomoću pokazivača koji je usmjeren na prvi element polja.

```c
#include<stdio.h>

int main(void) {

	int polje[] = { 1, 2, 3, 4, 5 };
	int *pok = polje;
	int n = sizeof(polje)/sizeof(int);
	/*izraz je ispravan samo ako je polje statički zauzeto*/
	int i;

	for (i = 0; i < n; i++)
	{
		if (i == 0) {
			printf("polje[%d", polje[i]);
			//printf("polje[%d", *(polje + i));
		}
		else if (i < n - 1) {
			printf(" %d ", polje[i]);
			//printf(" %d ", *(polje + i));
		}
		else {
			printf("%d]\n", polje[i]);
			//printf("%d]\n", *(polje + i));
		}
	}

	for (i = 0; i < n; i++)
	{
		if (i == 0) {
			printf("pok[%d", *(pok + i));
			//printf("polje[%d", pok[i]);
		}
		else if (i < n - 1) {
			printf(" %d ", *(pok + i));
			//printf(" %d ", pok[i]);

		}
		else {
			printf("%2d]\n", *(pok + i));
			//printf("%2d]\n", pok[i]);
		}
	}

	return 0;
}
```

Kako su pokazivači i polja u srodstvu za pristup elementima polja pomoću imena polja ili pokazivača koji je usmjeren na prvi element polja, može se koristiti notacija polja kao i pokazivačka notacija u jednom i drugom slučaju. No potrebno je napomenuti kako u određenim slučajevima pokazivačka

notacija je efikasnija ili u najgorem slučaju jednaka kao notacija polja. Uglate zagrade (*[]*) i operator zvjezdica (*) vrše istu operaciju, a to je dereferenciranje, odnosno dohvaćanje sadržaja na određenoj memorijskoj adresi. Notacija polja *polje[i]* je ekvivalentna pokazivačkoj notaciji **(polje + i)*. Kod notacije polja *polje[i]* ili pokazivačke notacije **(polje + i)*, ime polja predstavlja memorijsku adresu prvog elementa polja dok se indeksom (*i*) radi pomicanje od memorijske adrese prvog elementa polja na memorijsku adresa dobivenu rješenjem izraza (*i * sizeof(int)*), a kada se napravi pomak na željenu memorijsku adresu, uglate zagrade kao i zvijezdica vrše dereferenciranje odnosno dohvaćanje konkretne vrijednosti na toj memorijskoj adresi. Analogno objašnjenje vrijedi i za izraze *pok[i]* i **(pok + i)*.

Dvostruki pokazivač

Dvostruki pokazivač je varijabla u koju se može spremiti samo jedna vrijednost, a to je memorijska adresa jednostrukog pokazivača.

Opći oblik deklaracije dvostrukog pokazivača:

```
tip_podatka **ime_dvostrukog_pokazivaca;
```

Pokazivačka varijabla dvostrukog pokazivača počinje na određenoj memorijsko adresi, uobičajeno zauzima 4 bajta memorije, također neovisno kojim je tipom podatka deklarirana, ali to isto ovisi o kompilatoru i sustavu za koji se prevodi. Tip podatka dvostrukog pokazivača mora odgovarati tipu podatka jednostrukog pokazivača na koji će se dvostruki pokazivač usmjeriti.

Primjena dvostrukog pokazivača je indirektni pristup sadržaju memorijske adrese koja je spremljena u jednostrukom pokazivaču, odnosno sadržaju varijable čija je memorijska adresa spremljena u jednostrukom pokazivaču.

Operator indirektnog pristupa ili dereferenciranja (**) u deklaraciji označava da je riječ o pokazivačkoj varijabli dvostrukog pokazivača, dok njegova primjena nad dvostrukim pokazivačem poslije deklaracije, označava indirektni pristup sadržaja neke varijable preko jednostrukog pokazivača.

Kao što je bila preporuka kod jednostrukih pokazivača, tako vrijedi isto kod dvostrukih pokazivača, a to je da se pokazivačku varijablu inicijalizira na neku početnu vrijednost, to je najčešće *NULL* vrijednost. Osobito pripaziti da se ne koristi operator indirektnog pristupa nad pokazivačkom varijablom koja nije inicijalizirana ili usmjerena.

Primjer 4: Indirektni pristup sadržaju varijable preko dvostrukog pokazivača.

```c
#include<stdio.h>

int main(void) {

	int a = 5;
	int *jedPok = &a;
	int **dvoPok = &jedPok;

	printf("a == %d\n", a);
	printf("a == *jedPok == %d\n", *jedPok);
	printf("a == **dvoPok == %d\n", **dvoPok);

	printf("&a == %ul\n", &a);
	printf("&a == jedPok == %ul\n", jedPok);
	printf("&a == *dvoPok == %ul\n", *dvoPok);
	printf("&jedPok == dvoPok == %ul\n", dvoPok);

	return 0;
}
```

Dvostruki pokazivač se jedino može usmjeriti na jednostruki pokazivač. Sadržaj varijable dvostrukog pokazivača je memorijska adresa jednostrukog pokazivača. Kako bi se pomoću dvostrukog pokazivača indirektno dohvatio sadržaj memorijske adrese koja je pohranjena unutar jednostrukog pokazivača, dvostruki pokazivač je potrebno dva puta dereferencirati. Kada se prvi puta dereferencira dvostruki pokazivač, dobiva se memorijska adresa koja je spremljena unutar jednostrukog pokazivača. Kada se drugi puta dereferencira dvostruki pokazivač dobiva se konkretni sadržaj koji se nalazi unutar memorijske adrese spremljene u jednostrukom pokazivač, a to je sadržaj varijable na koju je usmjeren jednostruki pokazivač.

Primjena dvostrukog pokazivača je široka, ponajviše u radu sa strukturama podataka, kao što je primjerice povezani popis ili prilikom dinamičkog zauzimanja memorijskog prostora za dvodimenzionalno polje jer ti koncepti traže spremanje memorijske adrese jednostrukog pokazivača i naknadni pristup podacima na određenim memorijskim adresama. Upravo zbog tih koncepata, primjena dvostruki pokazivač je neizostavna, te na temelju dvostrukog pokazivača i njegovim dereferenciranjem se može pristupiti konkretnim vrijednostima u memoriji računala.

Polje pokazivača

Kao što se može kreirati polje određenog tipa podatka gdje je svaki element varijabla istog tipa, tako se može kreirati polje pokazivača, gdje je svaki element pokazivač na tip podatke kojim se deklariralo polje pokazivača.

Opći oblik deklaracije polja pokazivača:

```
tip_podatka *ime_polja_pokazivaca[BR_ELEMENATA];
```

Tip podatka polja pokazivača mora odgovarati tipu podatka na koji će pokazivači biti usmjereni. Ime polja pokazivača predstavlja memorijsku adresu cijelog polja, odnosno memorijsku adresu prvog elementa polja koji je jednostruki pokazivač. Ime polja pokazivača je ujedno dvostruki konstantni pokazivač na prvi element polja pokazivača.

Primjer 5: Usmjeravanje pokazivača iz polja pokazivača na pojedini element cjelobrojnog polja.

```
#include<stdio.h>
int main(void) {
	int polje[] = { 1, 2, 3, 4, 5 };
	int *pok[5] = { NULL };
	int n = sizeof(polje)/sizeof(int);
	/*izraz je ispravan samo ako je polje statički zauzeto*/
	int i;
	for (i = 0; i < n; i++) {
		*(pok + i) = &polje[i];
		//pok[i] = (polje + i);
	}
	for (i = 0; i < n; i++) {
		if (i == 0) {
			printf("pok[%d", *(*(pok + i)));
			//printf("pok[%d", *pok[i]);
		}
		else if (i < n - 1) {
			printf(" %d ", *(*(pok + i)));
			//printf(" %d ", pok[i][0]);
		}
		else {
			printf("%d]\n", *(*(pok + i)));
			//printf("%d]\n", *(*(pok + i) + 0));
		}
	}
	return 0;
}
```

Deklaracija *int *pok[5];* predstavlja polje pokazivača, gdje je svaki element jednostruki pokazivač na cjelobrojni tip podatka tipa *int* i u ovo se polje može spremiti pet memorijske adrese varijabli tipa *int*. Svaki element polja pokazivača biti će usmjeren na jedan element cjelobrojnog polja i to na način da se dohvati memorijska adresa pojedinog elementa cjelobrojnog polja i dodjeli se određenom pokazivaču iz polja pokazivača. To je prikazano prvom (*for*) petljom.
Kako bi se memorijska adresa mogla dodijeliti pojedinom pokazivaču iz polja pokazivača, prvo je potrebno pristupiti određenom pokazivaču koji se nalazi na određenom indeksu, taj se indeks također

koristi za pristup elementu u cjelobrojnom polju. Za to se koristi sintaksa **(pok + i)* ili *pok[i]*, gdje je *pok* dvostruki pokazivač u kojemu se nalazi memorijska adresa prvog elementa u polju pokazivača, a to je memorijska adresa jednostruki pokazivač pod indeksom (*i == 0*). Kako obične zagrade imaju najveći prioritet izvršavanja, prvo se pomoću sintakse (*pok + i*) pozicionira na memorijsku adresu određenog elementa polja, ovisno o indeksu (*i*). Zvjezdicom (***) ili uglatom zagradom (*[]*) se zatim vrši dereferenciranje i na određenom indeksu se dohvaća jednostruki pokazivač u koji se sada može spremiti memorijska adresa. Kada se pomoću određene vrijednosti indeksa dohvati pokazivač iz polja pokazivača, pridružuje mu se memorijska adresa varijable iz cjelobrojnog polja za koju se koristi isti indeks. Ovim načinom se dobilo slijedno pridruživanje memorijske adrese elementa cjelobrojnog polja u jednostruke pokazivače unutar polja pokazivača.

Za pristup elementima cjelobrojnog polja pomoću polja pokazivača, može se koristiti notacija polja ili pokazivačka notacija, prikazano drugom (*for*) petljom. Prilikom indirektnog pristupa elementima cjelobrojnog polja pomoću polja pokazivača prvo je potrebno primijetiti kako je ime polja pokazivača konstantni dvostruki pokazivač. To znači da bi se indirektno pristupilo elementima cjelobrojnog polja potrebno je dva puta dereferencirati ime polja pokazivača uz primjenu indeksa kojim se dohvaća određeni pokazivač iz polja pokazivača. U dvostrukom pokazivaču (*pok*) nalazi se memorijska adresa prvog jednostrukog pokazivača, a sintaksa (*pok + i*) predstavlja pomicanje i uvid u memorijsku adresu pojedinog jednostrukog pokazivača iz polja pokazivača na temelju indeksa (*i*). Nakon što se pozicioniralo na određenu memorijsku adresu jednostrukog pokazivača pomoću indeksa (*i*) sintaksom **(pok + i)* ili *pok[i]* radi se prvo dereferenciranje koje omogućava pristup jednostrukom pokazivaču i uvid u njegov sadržaj, a to je memorijska adresa varijable iz cjelobrojnog polja. Drugim dereferenciranjem **(*(pok + i))* ili **pok[i]* pristupa se sadržaju u memorijskoj adresi koja je spremljena unutar trenutnog dohvaćenog jednostrukog pokazivača, te se time omogućio indirektni pristup varijabli iz cjelobrojnog polja.

Izrazi *pok[i][0]* i **(*(pok + i) + 0)* su također legitimne sintakse kojima se dohvaća vrijednost u cjelobrojnom polju preko polja pokazivača. Prvim indeksom se dohvaća jednostruki pokazivač iz polja pokazivača, a drugim indeksom (0) se dohvaća tj. označava samo jednu varijablu u cjelobrojnom polju. Važno je za primijetiti kako se drugi indeks koristi kada se želi dohvatiti element podpolja o čemu će se nešto više govoriti u narednom dijelu, dok se u ovom primjeru koristi samo jedna varijabla umjesto podpolja. Kod prethodnih sintaksi **(*(pok + i))*, **pok[i]* nije se pisala 0, već se ona implicitno nalazila kao da je eksplicitno bila pisana **(*(pok + i) + 0)* ili **(pok[i] + 0)*.

Koncept polja pokazivača može se koristiti prilikom pridruživanja i dohvaćanja redova dvodimenzionalnog polja.

Primjer 6: Pridruživanje i dohvaćanja redova dvodimenzionalnog polja pomoću polja pokazivača.

```
#include<stdio.h>

int main(void) {

	int M[][3] = { { 1, 2, 3 },
		       { 4, 5, 6 },
		       { 7, 8, 9 },
		       { 10, 11, 12 } };
	int *pok[4] = { NULL };
	int i, j;

	for (i = 0; i < 4; i++) {
		*(pok + i) = &M[i][0];
		//pok[i] = M[i];
		//*(pok + i) = M[i];
		//*(pok + i) = *(M + i);
	}

	for (i = 0; i < 4; i++) {
		for (j = 0; j < 3; j++) {
			printf("%3d", *(*(pok + i) + j));
			//printf("%3d", pok[i][j]);
			//printf("%3d", *(pok[i] + j);

		}
		printf("\n");
	}

	return 0;
}
```

Kako je dvodimenzionalno polje zapravo polje podpolja, onoliko koliko dvodimenzionalno polje ima podpolja, toliko treba imati pokazivača u polju pokazivača. Pojedini pokazivač iz polja pokazivača bit će usmjeren na pojedini prvi element svakog reda, odnosno na prvi element svakog podpolja dvodimenzionalnog polja. To je vidljivo u prvoj (*for*) petlji, gdje se sintaksom **(pok + i)* ili *pok[i]* dohvaća pojedini jednostruki pokazivač iz polja pokazivača i u trenutno dohvaćeni jednostruki pokazivač pridružuje se memorijska adresa trenutno dohvaćeno prvog elementa podpolja jednom od tri prikazane sintakse, npr. *&M[i][0]*.
Prilikom indirektnog dohvaćanja elemenata dvodimenzionalno polja pomoću polja pokazivača prvo je potrebno pristupiti jednostrukom pokazivaču na određenom indeksu i zatim dereferencirati taj pokazivač kako bi se dohvatila memorijska adresa prvog elementa u određenom podpolju. Dodatnim dereferenciranjem pokazivača dohvaća se konkretni element unutar podpolja. To je vidljivo unutar ugniježđenih (*for*) petlji, gdje se sintaksom **(pok + i)* ili *pok[i]* pristupa sadržaju dohvaćenog jednostrukog pokazivača unutar kojeg se nalazi memorijska adresa prvog elementa reda dvodimenzionalnog polja određenim indeksom (*i*). Sintaksom *(*(pok + i) + j)* ili *(pok[i] + j)* šeće se po memorijskim adresama pojedinog elementa podpolja primjenom indeksa (*j*). Da bi se dohvatila konkretna vrijednost u podpolju potrebno je nakon pozicioniranja na memorijsku adresu trenutnog

elementa indeksom (*j*) još jednom dereferencirati tu memorijsku adresu sintaksom **(*(pok + i) + j)*, *pok[i][j]*, ili **(pok[i] + j)*.

Polje pokazivača se može koristiti za usmjeravanje i dohvaćanje elemenata jednodimenzionalnih polja različitih duljina.

Primjer 7: Pristup poljima znakova koje tvore string različite duljine pomoću polja pokazivača.

```
#include<stdio.h>

int main(void) {

	char *mjeseci[] = { "sijecanj", "veljaca", "ozujak", "travanj", "svibanj",
			    "lipanj", "srpanj", "kolovoz", "rujan", "listopad",
			    "studeni", "prosinac" };
	int i;

	for (i = 0; i < 12; i++) {
		printf("%d\t%s\n", i + 1, *(mjeseci + i));
		//printf("%d\t%s\n", i + i, mjeseci[i]);
	}

	return 0;
}
```

Prilikom deklaracije polja pokazivača izostavio se broj elemenata tj. duljina polja i prepustilo se određivanje duljine polja od strane kompilatora koji će odrediti duljinu polja pokazivača na temelju broja pokazivača.
Kako je unutar inicijalizatora navedeno dvanaest string vrijednosti, toliko će biti i pokazivača na znakovni tip *char* unutar polja pokazivača. Za svaku string vrijednost zauzet će se memorijski prostor i pojedini pokazivač iz polja pokazivača biti će usmjeren na prvi element pojedinog stringa.
Kako bi se dohvatio pojedini string i ispisao pomoću polja pokazivača prvo je potrebno pristupiti jednostrukom pokazivaču na određenom indeksu unutar polja pokazivača. Pristupanje pojedinom pokazivaču radi se pomoću sintakse **(mjeseci + i)*, ili *mjeseci[i]*.
Kako je (*mjeseci*) dvostruki pokazivač, on pokazuje na memorijsku adresu prvog jednostrukog pokazivača, a indeksom (*i*) pomiče se po polju pokazivača, tj. po memorijskim adresama jednostrukih pokazivača unutar polja pokazivača. Kada se odabere memorijska adresa jednog od pokazivača, potrebno je dereferencirati pokazivač (*mjeseci*) kako bi se pristupilo jednostrukom pokazivaču iz polja pokazivača. Tako pristupljenom jednostrukom pokazivaču dobiva se uvid u njegov sadržaj, a to je memorijska adresa početka trenutnog stringa, tj. memorijska adresa prvog elementa stringa na koji je usmjeren trenutno dohvaćeni pokazivač.
Sada je moguće ispisati taj string na standardni izlaz na način da se funkciji *printf()* preda memorijska adresa prvog elementa stringa, a na mjestu specifikatora pretvorbe (*%s*) ispisat će se cjelokupni string.
Dodatnim dereferenciranjem dvostrukog pokazivača (*mjeseci*) pristupilo bi se sadržaju memorijske adrese koja je spremljena u trenutno dohvaćenom jednostrukom pokazivaču, a to je prvi element stringa, tj. prvi znak stringa. Dodatnim indeksom u kombinaciji s dvostrukim dereferenciranjem pokazivača (*mjeseci*), omogućilo bi se kretanje i dohvaćanje ostalih znakova koje tvore trenutno dohvaćeni string. Primjer takve sintakse je **(*(mjeseci + i) + j)*, *mjeseci[i][j]*, ili **(mjeseci[i] + j)*.

Pokazivači i višedimenzionalna polja

Najjednostavniji oblik višedimenzionalnih polja predstavlja dvodimenzionalno polje, a dvodimenzionalno polje je zapravo jednodimenzionalno polje čiji je svaki element jednodimenzionalno polja. Opći oblik deklaracije dvodimenzionalnog polja:

```
tip_podatka ime_dvodimenzionalnog_polja[BR_PODPOLJA][BR_EL_PODPOLJA];
                                        ili
tip_podatka ime_dvodiemnzionalnog_polja[BR_REDOVA][BR_STUPACA];
```

Dvodimenzionalno polje može se smatrati tablicom koja ima određeni broj redova i stupaca.

int polje2D[i][j]	**Stupac 0**	**Stupac 1**	**Stupac 2**
Red 0	polje2D[0][0]	polje2D [0][1]	polje2D [0][2]
Red 1	polje2D [1][0]	polje2D [1][1]	polje2D [1][2]
Red 2	polje2D [2][0]	polje2D [2][1]	polje2D [2][2]

Svakom podatku u dvodimenzionalnom polju pristupa se na temelju imena dvodimenzionalnog polja i dvaju indeksa unutar uglatih zagrada, npr. *polje2D[i][j]*, gdje je naziv polja *polje2D*, a (*i, j*) su indeksi koji jedinstveno identificiraju položaj pojedinih elemenata unutar dvodimenzionalnom polju.

Primjer 8: Pristup elementima dvodimenzionalnog polja.

```
#include<stdio.h>

int main(void) {

	int polje2D[][3] = { {1, 2, 3},
			     {4, 5, 6},
			     {7, 8, 9},
			     {10, 11, 12} };
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("polje2D[%d][%d]\t%d\n", i, j, polje2D[i][j]);
		}
	}

	return 0;
}
```

Elementi statički zauzetog dvodimenzionalnog polja su u memoriji spremljeni sekvencijalno (slijedno), red za redom.

polje2D[0][0]	**polje2D[0][1]**	**polje2D[0][2]**	**...**	**polje2D[3][1]**	**polje2D[3][2]**	**polje2D[3][3]**

Kao i kod jednodimenzionalno polja, ime dvodimenzionalnog polja predstavlja memorijsku adresu cijelog polja, odnosno memorijsku adresu prvog elementa polja (prvo jednodimenzionalno polje), odnosno ime dvodimenzionalnog polja je konstantni pokazivač na prvi element polja koji nije moguće preusmjeriti.
Elementi polja se jednostavno mogu dohvatiti i promijeniti primjenom pokazivačke notacije, neki primjeri pokazivačke notacije prikazani sljedećom tablicom:

Sintaksa u C-u	Opis rezultata sintakse
polje2D	adresa dvodimenzionalnog polja, odnosno prvog jednodimenzionalnog polja
polje2D + i	adresa i-tog jednodimenzionalnog polja
***(polje2D + i)**	adresa prvog elementa i-tog jednodimenzionalnog polja
***(polje2D + i) + j**	adresa j-tog elementa u i-tom jednodimenzionalnom polju
***(*(polje2D + i) + j)**	j-ti elementa u i-tom jednodimenzionalnom polju

U sljedećoj tablici pokazani su različiti načini dohvaćanja elemenata dvodimenzionalnog polja pomoću notacije polja, pokazivačke notacije i kombinirane notacije:

	Notacija polja	Pokazivačka notacija	Kombinirana notacija
Dohvaćanje vrijednosti elemenata	polje2D[0][0]	**polje2D	*polje2D[0]
	polje2D[i][j]	*(*(polje2D + i) + j) *(polje2D + i * br_stupaca + j) *(*polje2D + i)	*(polje2D[i] + j)
Dohvaćanje memorijske adrese elemenata	&polje2D[0][0]	polje2D	polje2D[0]
	&polje2D[i][j]	(*(polje2D + i) + j) (polje2D + i * br_stupaca + j) (*polje2D + i)	(polje2D[i] + j)

Primjer 9: Dohvaćanje elemenata dvodimenzionalnog polja različitim notacijama.

```
#include<stdio.h>
int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("polje2D[%d][%d]\t%d\t", i, j, polje2D[i][j]);
//printf("*(*(polje2D + %d) + %d)\t%d\t", i, j, *(*(polje2D + i) + j));
//printf("*(polje2D[%d] + %d)\t%d\t", i, j, *(polje2D[i] + j));
		}
		printf("\n");
	}
	return 0;
}
```

Primjer 9b: Dohvaćanje elemenata dvodimenzionalnog polja različitim notacijama.

```
#include<stdio.h>
int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int *pokNa2D = *(polje2D + 0);
	//int *pokNa2D = *polje2D ili polje2D[0];
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
	printf("*(pokNa2D + %d * %d + %d\t%d\t", i, 3, j, *(pokNa2D + i * 3 + j));
		}
		printf("\n");
	}
	return 0;
}
```

U ovom se primjeru koristi predložena sintaksa **(pokNa2D + i * 3 + j)* i ona vrijedi samo ako je dvodimenzionalno polje statički zauzeto jer su tada elementi slijedno spremljeni unutar memorije računala. Potrebno je usmjeriti jednostruki pokazivač na prvi element prvog podpolja dvodimenzionalnog polja te predloženom formulom pristupati pojedinim elementima dvodimenzionalnog polja.

Primjer 10: Dohvaćanje adrese elemenata dvodimenzionalnog polja različitim notacijama.

```
#include<stdio.h>

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("&polje2D[%d][%d]\t%p\t", i, j, &polje2D[i][j]);
//printf("(*(polje2D + %d) + %d)\t%p\t", i, j, (*(polje2D + i) + j));
//printf("(polje2D[%d] + %d)\t%p\t", i, j, (polje2D[i] + j));
		}
		printf("\n");
	}
	return 0;
}
```

Primjer 10b: Dohvaćanje adrese elemenata dvodimenzionalnog polja različitim notacijama.

```
#include<stdio.h>
int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int *pokNa2D = *(polje2D + 0);
	//int *pokNa2D = *polje2D ili polje2D[0];
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
	printf("(pokNa2D + %d * %d + %d\t%d\t", i, 3, j, (pokNa2D + i * 3 + j));
		}
		printf("\n");
	}
	return 0;
}
```

U ovom se primjeru koristi predložena sintaksa *(pokNa2D + i * 3 + j)* i ona vrijedi samo ako je dvodimenzionalno polje statički zauzeto jer su tada elementi slijedno spremljeni unutar memorije računala. Potrebno je usmjeriti jednostruki pokazivač na prvi element prvog podpolja dvodimenzionalnog polja te predloženom formulom pristupati pojedinim memorijskim adresama elemenata dvodimenzionalnog polja.

Pokazivač na polje

Moguće je kreirati pokazivač koji će pokazivati na cijelo polje, a ne samo na prvi element polja. Takav pokazivač se naziva pokazivač na polje. Opća deklaracija:

```
tip_podatka (*ime_pokazivaca_na_polje)[BR_ELEMENATA];
```

Pokazivač kao takav je usmjeren na kompletno polje, a ne na prvi element polja. Izraz (**ime_pokazivaca_na_polje*) predstavlja pokazivač na polje koji također u svojoj deklaraciji mora sadržavati tip podatka i broj elemenata polja, odnosno duljinu polja koja mora odgovora tipu podatka i duljini polja na koje će biti usmjeren. Pokazivač na polje u sebi sadrži jednu memorijsku adresu koja predstavlja početak polja na koje je usmjeren pokazivač na polje.
Ako bi se izostavile zagrade oko pokazivača, tada bi deklaracija pokazivača postala deklaracija polja pokazivača.
Potrebno je razumjeti kako su pokazivač na prvi element polja i pokazivač na cijelo polje dva potpuno različita koncepta.

Primjer 11: Pridruživanje memorijske adrese prvog elementa polja jednostrukom pokazivaču, te pridruživanje memorijske adrese cijelog polja pokazivaču na polje.

```c
#include<stdio.h>

int main(void) {

	int polje[] = { 1, 2, 3, 4, 5 };
	int *p = polje;
	int (*pPolje)[5] = &polje;

	printf("Adresa prije inkrementa\n");
	printf("p = %lu\n", p);
	printf("pPolje = %lu\n", pPolje);

	p++;
	pPolje++;

	printf("Adresa poslije inkrementa\n");
	printf("p = %lu\n", p);
	printf("pPolje = %lu\n", pPolje);

	printf("*pPolje = %lu\n", *pPolje);
	printf("**pPolje = %d\n", **pPolje);

	return 0;
}
```

Pokazivač *p* je usmjeren na prvi element polja, dok je pokazivač na polje *pPolje* usmjeren na cijelo polje. Obratiti pozornost kako je dodijeljena memorijska adresa cijelog polja primjenom adresnog operatora (*&*) uz ime cjelobrojnog polja. Osnovni tip podatka pokazivača *p* je *int**, odnosno pokazivač na *int* tip podatka, a osnovni tip pokazivača *pPolje* je pokazivač na pet elemenata tipa *int*. Prvim ispisom sadržaja ovih pokazivača dobile su se iste memorijske adrese, odnosno ispisom sadržaja pokazivača *p* dobiva se memorijska adresa prvog elementa polja što je ujedno memorijska adresa početka polja, dok ispisom pokazivača na polje *pPolje* dobije se memorijska adresa cijelog polja, odnosno početka polja koja opet odgovara prvom elementu polja. Kako se pokazivačka aritmetika izvodi relativno prema osnovnom tipu podatka, npr. za pokazivač na *int* tip podatka *(memorijska adresa + sizeof(int))*, povećanjem pokazivača za jedan, dohvaća se sljedeća memorijska adresa koja slijedi nakon prvog elementa *(1000 + 4bajta) = 1004*. Ponovnim ispisom sadržaja pokazivača *p* dobiva se memorijska adresa drugog elementa, odnosno radi se povećanje od 4 bajta zbog tipa podatka na koji je pokazivač usmjeren. Ispisom pokazivača na polje *pPolje* dobiva povećanje od 20 bajtova, upravo zbog toga što polje sadrži pet elemenata tipa *int*, što je ukupno 20 bajtova *(memorijska adresa + sizeof(int) * (sizeof(polje)/sizeof(int)))*, odnosno *(10000 + 4 * (20/4)) = (10000 + 4 * 5) = (10000 + 20) = 10020*.
U zadnjem ispisu pokazivač na polje, *pPolje* je dereferenciran s čime se njegovim ispisom dobije memorijska adresa prvog elementa, što je vrlo važno za primijetiti. *pPolje* pokazuje na polje od pet elemenata, odnosno na cijelo polje, dok **pPolje* pokazuje na prvi element polja, odnosno memorijsku adresu prvog elementa polja. Dodatnim dereferenciranjem pokazivača na polje *pPolje* s dodatnom zvjezdicom ***pPolje* pristupa se vrijednosti prvog elementa polja *polje*. Ovo predstavlja važan koncept koji će se koristiti za pristup dvodimenzionalnim poljima.
Ime dvodimenzionalnog polja je konstantni pokazivač na prvi element a prvi element je jednodimenzionalno polje, stoga ime dvodimenzionalnog polja je pokazivač na polje.

Primjer 12: Ispisivanje memorijske adrese pojedinog jednodimenzionalnog polja, te ispisivanje memorijske adrese prvog elementa pojedinog jednodimenzionalnog polja.

```c
#include<stdio.h>

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int i;

	printf("Ispis memorijske adrese pojedinog podpolja\n");
	for (i = 0; i < 4; i++)
	{
		printf("polje2D[%d]\t%lu\n", i, polje2D + i);
	}

	printf("\nIspis memorijske adrese prvog elementa podpolja\n");
	for (i = 0; i < 4; i++)
	{
			printf("polje2D[%d]\t%lu\n", i, *(polje2D + i));
			//printf("polje2D[%d]\t%lu\n“, i, polje2D[i]);
	}

	return 0;
}
```

Ime dvodimenzionalnog polja *polje2D* je pokazivač na polje i konstantni pokazivač na prvi element dvodimenzionalnog polja koje je jednodimenzionalno polje. U prvoj *for* petlji pomoću pokazivača na polje *polje2D* dohvaća se svako jednodimenzionalno polje i ispisuje se memorijska adresa dohvaćenog jednodimenzionalnog polja primjenom sintakse (*polje2D + i*). Svakim povećanjem indeksa (*i*) u pokazivačkoj aritmetici radi se pomak od 12 bajtova te se tako dohvaća memorijska adresa sljedećeg jednodimenzionalnog polja. Može se reći kako izraz pokazivačke aritmetike (*polje2D + i*) pokazuje na *i*-to jednodimenzionalno polje. U drugoj *for* petlji, izraz (*polje2D + i*) se dereferencira i dohvaća se memorijska adresa prvog elementa u svakom jednodimenzionalnom polju primjenom sintakse **(polje2D + i)* ili *polje2D[i]*.
Dereferenciranjem izraza pokazivača na polje *(polje2D + i)* dobiva se memorijska adresa prvog elementa tog određenog polja. To je zato što je osnovni tip podatka na koji je usmjeren pokazivač na polje *polje2D* pokazivač na tri elemenata tipa *int*, dok je osnovni tip podatka dereferenciranog pokazivača na polje **(polje2D + i) int**, tj. pokazivač na *int*. Izraz (*polje2D + i*) i izraz **(polje2D + i)* pokazuju na istu memorijsku adresu, ali im je različit tip podatka na koji pokazuju.

Primjer 13: Ispisivanje elemenata dvodimenzionalnog polja korištenjem pokazivačke notacije s objašnjenjem pokazivača na polje.

```c
#include<stdio.h>

int main(void) {

	int polje2D[][3] = { {1, 2, 3},
			     {4, 5, 6},
			     {7, 8, 9}},
			     {10, 11, 12} };
	int i, j;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("polje2D[%d][%d]\t%d\n", i, j, *(*(polje2D + i) + j));
			//printf("polje2D[%d][%d]\t%d\n", i, j, polje2D[i][j]);
		}
		printf("\n");
	}

	return 0;
}
```

Razlaganjem izraza **(*(polje2D + i) + j)* slijedi objašnjenje. Kako *polje2D* predstavlja pokazivač na polje, izraz (*polje2D + i*) predstavlja dohvaćanje memorijske adrese pojedinog *i*-tog jednodimenzionalnog polja u nizu. Dereferenciranjem pokazivača na polje izrazom *(*polje2D + i*) ili *polje2D[i]* dohvaća se memorijska adresa prvog elementa unutar pojedinog *i*-tog jednodimenzionalnog polja. Izrazom (**(polje2D + i) + j*) ili (*polje2D[i] + j*) omogućava se prolaženje po memorijskim adresama pojedinog *j*-tog elementa unutar dohvaćenog *i*-tog jednodimenzionalnog polja. Izrazom **(*(polje2D + i) + j)* ili *polje2D[i][j]* pristupa se konkretnoj vrijednosti *j*-tog elementa unutar dohvaćenog *i*-tog jednodimenzionalnog polja.

Primjer 14: Usmjeravanje pokazivača na polje na prvi element dvodimenzionalno polje.

```
#include<stdio.h>

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int i, j;
	int(*p)[3] = polje2D;
	//p = polje2D;

	for (i = 0; i < 4; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("polje2D[%d][%d]\t%d\t", i, j, *(*(p + i) + j));
		}
		printf("\n");
	}

	return 0;
}
```

Isto objašnjenje kao i u prethodnom primjeru.

Dinamičko zauzimanje memorije za dvodimenzionalno polje

Prilikom statičkog zauzimanja memorije za dvodimenzionalno polje uvidjelo se kako su elementi dvodimenzionalnog polja koji su jednodimenzionalna polja slijedno spremljeni u memoriji jedno iza drugoga bez prekida. Upotrebom funkcija za dinamičko zauzimanje memorije kao što su *malloc()* i *calloc()* nema potrebe da unutarnja polja budu slijedno spremljena u memoriji jedno iza drugog zbog načina na koji će se zauzimati memorija za unutarnja polja. U tome slučaju treba obratiti pozornost kada je potreba za kopiranjem određenog dijela memorije koji predstavlja elemente dvodimenzionalnog polja u drugi jer se tada mora individualno pristupiti različitim dijelovima memorije kako bi se kopirali ti elementi, a ne od jednom kada bi se koristilo statičko zauzimanje memorije.

Primjenom različitih koncepata koji su se prethodno obradili moguće je kreirati dvodimenzionalnog polja dinamičkim zauzimanjem memorije. Za to je potrebno poznavanje koncepta dvostrukog pokazivača, polja pokazivača i dinamičkog zauzimanja memorije.

Dinamičko zauzimanje ne kontinuirane memorije za dvodimenzionalno polje

Primjer 15: Dinamičko zauzimanje ne kontinuirane memorije za dvodimenzionalno cjelobrojno polje (potpuno rukovanje memorijom), ujedno prikazano popunjavanje i dohvaćanje elemenata dvodimenzionalnog polja pokazivačkom notacijom.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define RED 4
#define STUPAC 3

int main(void) {

	srand((unsigned)time(NULL));

	int **polje2D = NULL;
	int i,j;

	polje2D = (int**)calloc(RED, sizeof(int*));
	if(polje2D == NULL){
		return 1;
	}

	for (i = 0; i < RED; i++) {
		*(polje2D + i) = (int*)calloc(STUPAC, sizeof(int));
		if(*(polje2D + i) == NULL){
			return 1;
		}
	}
```

```
	for (i = 0; i < RED; i++) {
		for (j = 0; j < STUPAC; j++) {
		*(*(polje2D + i) + j) = 1 + (float)rand() / RAND_MAX * (11 - 1);
		}
	}

	for (i = 0; i < RED; i++) {
		for (j = 0; j < STUPAC; j++) {
		printf("*(*(polje2D + %d) + %d)\t%d\t", i, j, *(*(polje2D + i) + j));
		}
		printf("\n");
	}

	for (i = 0; i < RED; i++) {
		free(*(polje2D + i));
	}

	free(polje2D);

	polje2D = NULL;

	return 0;
}
```

Da bi se dinamičkim zauzimanjem memorije moglo kreirati dvodimenzionalno polje, prvo je potrebno kreirati dvostruki pokazivač, u primjeru je dvostruki pokazivač imenovan *polje2D*. Zatim se pomoću funkcija *malloc()* ili *calloc()* dinamički zauzima memorijski prostor koji će predstavljati polje pokazivača. Izrazom *polje2D = (int**)calloc(4, sizeof(int*))*, pomoću funkcije *calloc()* zauzima se memorijski prostor za polje pokazivača od četiri elementa, svaki element je pokazivači na *int* tip podatka. Funkcije za dinamičko zauzimanje memorije vraćaju memorijsku adresu tipa *void** stoga je povratnu vrijednost funkcije *calloc()* potrebno promijeniti (engl. *cast*) iz generičkog pokazivača *void** u dvostruki pokazivač na tip podatka *int*.

To je potrebno napraviti jer funkcija za dinamičko zauzimanje memorije vraća memorijsku adresu prvog elementa koji je jednostruki pokazivač na tip podatka *int* i kao takva se može spremiti jedino u dvostruki pokazivač koji pokazuje na jednostruki pokazivač na tip *int*. Također to je potrebno napraviti jer se generičkim pokazivačem *void** ne može pristupiti indirektno podatku na nekoj lokaciji već ga je potrebno promijeniti u pokazivač na konkretni tip podatka.

Svaki pokazivač iz polja pokazivača predstavlja početak jednog reda dvodimenzionalnog polja, ali ti pokazivači trenutno nisu usmjereni na početke redova, stoga je potrebno posebno dinamički zauzeti memorijski prostor koji će predstavljati redove dvodimenzionalnog polja. Izrazom **(polje2D + i) = (int*)calloc(3, sizeof(int))* se zauzima memorijski prostor koji će predstavljati redove dvodimenzionalnog polja i nakon svakog pojedinog zauzimanje memorije, pojedinom se pokazivaču iz polja pokazivača funkcijom *calloc()* pridružuje memorijska adresa prvog elementa dinamički zauzete memorije. Ponovno je potrebno napraviti promjenu tipa podatka i to iz generičkog pokazivača u jednostruki pokazivač na tip podatka *int* jer se vraća memorijska adresa prvog elementa reda koji je tip podatka *int* i kao takav se može spremiti samo u jednostruki pokazivač te se preko jednostrukog pokazivača može pristupiti elementima tog reda.

Nakon što se završi s upotrebom dinamički zauzetog memorijskog prostora, potrebno je i osloboditi memoriju. Prvo je potrebno osloboditi memorijski prostor koji predstavljaju redove dvodimenzionalnog polja i to izrazom *free(*(polje2D + i))* gdje se pomoću dvostrukog pokazivača *polje2D* prvim dereferenciranjem dohvaća sadržaj pojedinog pokazivača unutar polja pokazivača, a to su memorijske adrese početka svakog reda dvodimenzionalnog polja i predaje se funkciji *free()* koja oslobađa memoriju na hrpi koja je dinamički zauzeta funkcijama *malloc()* ili *calloc()*. Nakon što se oslobodi memorija za pojedini red dvodimenzionalnog polja, potrebno je osloboditi memoriju za dinamički zauzetog polja pokazivača, a to se postiže izrazom *free(polje2D)* jer se u dvostrukom pokazivaču *polje2D* nalazi memorijska adresa prvog elementa polja pokazivača.

Nakon što se završilo oslobađanje memorije, dobra je praksa pokazivač postaviti na vrijednost *NULL* do njegove sljedeće upotrebe unutar kôda upravo iz sigurnosnih razloga kako se ne bi u nastavku kôda ilegalno pristupilo memorijskom prostoru za koji se oslobodila memorija.

Dinamičko zauzimanje kontinuirane memorije za dvodimenzionalno polje

Postoje dva pristupa za dinamičko zauzimanje kontinuirane memorije za dvodimenzionalno polje. Prvim pristupom se zauzima memorija za vanjsko polje i nakon toga se zauzima memorija za unutarnja polja tj. redove. Drugim pristupom se dinamički zauzima memorija za cjelokupno polje.

Primjer 16: Dinamičko zauzimanje kontinuirane memorije za dvodimenzionalno cjelobrojno polje (potpuno rukovanje memorijom) prvim pristupom, ujedno prikazano popunjavanje i dohvaćanje elemenata dvodimenzionalnog polja pokazivačkom notacijom.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define RED 4
#define STUPAC 3

int main(void) {

	srand((unsigned)time(NULL));

	int **polje2D = NULL;
	int i,j;

	polje2D = (int**)calloc(RED, sizeof(int*));
	if(polje2D == NULL){
		return 1;
	}

	*polje2D = (int*)calloc(RED * STUPAC, sizeof(int));
		if(*polje2D) == NULL){
			return 1;
		}

	for (i = 1; i < RED; i++) {
		*(polje2D + i) = *polje2D + i * STUPAC;
	}
```

```
        for (i = 0; i < RED; i++) {
                for (j = 0; j < STUPAC; j++) {
                *(*(polje2D + i) + j) = 1 + (float)rand() / RAND_MAX * (11 - 1);
                }
        }

        for (i = 0; i < RED; i++) {
                for (j = 0; j < STUPAC; j++) {
                printf("*(*(polje2D + %d) + %d)\t%d\t", i, j, *(*(polje2D + i) + j));
                }
                printf("\n");
        }

        free(*polje2D);
        free(polje2D);

        polje2D = NULL;

        return 0;
}
```

Za prvi pristup koristi se dvostruki pokazivač. S prvom *calloc()* funkcijom dinamički se zauzima memorija za polje pokazivača i povratna vrijednost *calloc()* funkcije se sprema u dvostruki pokazivač. Povratna vrijednost *calloc()* funkcije je memorijska adresa prvog jednostrukog pokazivača iz polja pokazivača. Svaki pokazivač iz polja pokazivača će pokazivati na prvi element reda. S drugom *calloc()* funkcijom se dinamički zauzima memorija za sve elemente dvodimenzionalnog polja u kontinuitetu. Povratna vrijednost *calloc()* funkcije je memorijska adresa prvog elementa jednodimenzionalnog polja koji predstavlja prvi element u prvom redu dvodimenzionalnog polja i sprema se u prvi jednostruki pokazivač iz polja pokazivača. U prvoj *for()* petlji, svakom se pokazivaču iz polja pokazivača pridružuje memorijska adresa elementa iz kontinuiranog dijela memorije koji predstavlja prvi element pojedinog reda, potrebno je primijetiti kako se krenulo od indeksa (1) jer se s drugom *calloc()* funkcijom pridružila memorijska adresa prvog elementa reda u prvi pokazivač iz polja pokazivača. Upravo izrazom *(*polje2D + i * STUPAC*) omogućuje se ispravno dohvaćanje memorijskih adresa pojedinog elementa iz kontinuiranog dijela memorije koji predstavlja prvi element u svakom redu dvodimenzionalnog polja i njihovo spremanje i očekivane jednostruke pokazivače iz polja pokazivača na temelju indeksa (i).

Razlaganjem izraza *(*polje2D + i * STUPAC*), dio izraza (**polje2D*) predstavlja sadržaj prvog jednostrukog pokazivača iz polja pokazivača i to je memorijska adresa početka zauzetog kontinuiranog dijela memorije, tj. memorijska adresa prvog elementa u prvom redu dvodimenzionalnog polja. Zatim dio izraza (*i * STUPAC*) predstavlja cjelobrojni dio koji će sudjelovati u pokazivačkoj aritmetici i predstavlja pomak za određeni broj elemenata tj. pomak za određeni broj memorijskih adresa. Kako se vrijednost brojača (*i*) povećava, množi se s brojem stupaca te sudjeluje u pokazivačkoj aritmetici. Tako se stvara pomak od početne memorijske adrese zauzetog kontinuirano dijela memorije te se dohvaća memorijska adresa elemenata koji predstavljaju prve elemente pojedinih redova dvodimenzionalnog polja.

Nakon što se odradi posao s dvodimenzionalnom poljem, potrebno je osloboditi memoriju i to na način da se prvo oslobodi memorija koja je zauzeta za kontinuirani dio memorije, odnosno redova dvodimenzionalnog polja, a tek nakon tog oslobađanja potrebno je osloboditi memoriju od polja pokazivača. Dobra je praksa pokazivač postaviti na vrijednost *NULL* do njegove sljedeće upotrebe unutar kôda upravo iz sigurnosnih razloga kako se ne bi u nastavku kôda ilegalno pristupilo memorijskom prostoru za koji se oslobodila memorija.

Primjer 17: Dinamičko zauzimanje kontinuirane memorije za dvodimenzionalno cjelobrojno polje (potpuno rukovanje memorijom) drugim pristupom, ujedno prikazano popunjavanje i dohvaćanje elemenata dvodimenzionalnog polja pokazivačkom notacijom.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define RED 4
#define STUPAC 3

int main(void) {

	srand((unsigned)time(NULL));

	int *polje2D = NULL;
	int i,j;

	polje2D = (int*)calloc(RED * STUPAC, sizeof(int));
	if(polje2D == NULL){
		return 1;
	}

	for (i = 0; i < RED; i++) {
		for (j = 0; j < STUPAC; j++) {
		*(polje2D + i * STUPAC + j) = 1 + (float)rand() / RAND_MAX * (11 - 1);
		}
	}

	for (i = 0; i < RED; i++) {
		for (j = 0; j < STUPAC; j++) {
printf("*(polje2D + %d*%d + %d)\t%d\t", i, STUPAC, j, *(polje2D + i * STUPAC + j));
		}
		printf("\n");
	}

	free(polje2D);

	polje2D = NULL;

	return 0;
}
```

Za drugi pristup koristi se jednostruki pokazivač. Funkcijom *calloc()* dinamički se zauzima memorijski prostor za dvodimenzionalno polje kao kontinuirani dio memorije koje je predstavljeno jednodimenzionalnim poljem. Broj elemenata u jednodimenzionalnom polju odgovara broju elemenata u dvodimenzionalnom polju. Važno je za primijetiti kako se kod ovog pristupa prilikom pristupanja elementima polja ne koristi dvostruko dereferenciranje, već jednostruko i još se mora izrazom **(polje2D + i * STUPAC + j)* izračunati koliko je svaki element udaljen jedan od drugoga.
Razlaganjem izraza **(polje2D + i * STUPAC + j)*, dio izraza *polje2D* predstavlja memorijsku adresu prvog elementa polja i ujedno je to element prvog reda. Dio izraza (*i * STUPAC*) izračunava se cjelobrojna vrijednost koja će sudjelovati u pokazivačkoj aritmetici za pozicioniranje na memorijsku adresu elementa koji predstavlja prvi element u pojedinom redu, dok se (*j*) u izrazu koristi da bi se pozicioniralo na memorijsku adresu elementa koji se nalazi u određenom redu. Nakon što se izračunavanjem spomenutog izraza dobije memorijska adresa određenog elementa vrši se dereferenciranje i pristupanju sadržaju te memorijske adrese.

Nakon što se odradi posao s dvodimenzionalnom poljem koji je predstavljen jednodimenzionalnim poljem u kontinuitetu, potrebno je osloboditi memoriju i to na način da se samo oslobodi memorija koja je zauzeta za kontinuirani dio memorije, odnosno za jednodimenzionalno polje. Dobra je praksa pokazivač postaviti na vrijednost *NULL* do njegove sljedeće upotrebe unutar kôda upravo iz sigurnosnih razloga kako se ne bi u nastavku kôda ilegalno pristupilo memorijskom prostoru za koji se oslobodila memorija.

Predavanje dvodimenzionalnog polja funkciji

Predavanje dvodimenzionalnog polja funkciji nije jednostavan zadatak, jer je potrebno obratiti pozornost na koji se način polje predaje funkciji i na koji se način pristupa elementima u predanom polju unutar funkcije. Nije isto predaje li se statički zauzeto polje ili dinamički zauzeto polje, također nije isto na koji se način polje dinamički zauzelo te se nakon toga predaje funkciji.

Primjer 18: Primjer predavanja statički zauzetog dvodimenzionalnog polja funkciji, primjenom jednostrukog pokazivača.

```
#include<stdio.h>

void ispis2D(int*, int, int);

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };

	ispis2D(&polje2D[0][0], 4, 3);

	return 0;
}

void ispis2D(int *p, int m, int n) {
	int i, j;

	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++) {

			printf("%3d", *(p + i * n + j));
		}
		printf("\n");
	}
}
```

Unutar memorije računala, elementi dvodimenzionalnog polja nalaze se slijedno jedan iza drugoga i stoga je moguće primjenom jednostrukog pokazivača proći kroz sve elemente dvodimenzionalnog polja. U ovome primjeru, funkciji se predaje memorijska adresa prvog elementa koji se nalazi unutar prvog jednodimenzionalnog polja. Zatim se izrazom **(p + i * n + j)* pristupa pojedinim elementima dvodimenzionalnog polja. Važan koncept kod ovog izraza je u tome što se promjenom varijable (*i*) pristup pojedinom redu, dok se promjenom varijable (*j*) pristupa pojedinim elementima reda. Varijabla (*n*) predstavlja broj elemenata unutar pojedinog reda iliti broj stupaca.
Obratiti pozornost kako se uz memorijsku adresu prvog elementa u prvom redu dvodimenzionalnog polja predaje i broj redova i stupaca, što je obavezno, jer u C programskom jeziku ne postoji mehanizam kojim bi se otkrilo unutar funkcije koliko polje ima elemenata, te je potrebno eksplicitno navesti u pozivu funkcije te pripremiti formalne parametre koji će primiti tu informaciju prilikom poziva funkcije.

Primjer 19: Primjer predavanja statički zauzetog dvodimenzionalnog polja funkciji, primjenom dvostrukog pokazivača.

```c
#include<stdio.h>

void ispis2D(int**, int, int);

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };
	int *pok[4] = { NULL };
	int i;

	for (i = 0; i < 4; i++) {
		*(pok + i) = &polje2D[i][0];
	}

	ispis2D(pok, 4, 3);

	return 0;
}

void ispis2D(int **p, int m, int n) {
	int i, j;

	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++) {

			printf("%3d", *(*(p + i) + j));
		}
		printf("\n");
	}
}
```

U ovome se primjeru dvodimenzionalno polje predaje funkciji tako da se prvo kreira polje pokazivača čiji će se elementi usmjeriti na prvi element pojedinog reda dvodimenzionalnog polja. Kako bi se polje pokazivač moglo predati funkciji, parametar funkcije koji će primiti memorijsku adresu prvog elementa polja pokazivača mora biti dvostruki pokazivač jer se funkciji predaje memorijska adresa prvog jednostrukog pokazivača iz polja pokazivača. Nedostatak ovog koncepta je u tome što je potrebno kreirati polje pokazivača, statički ili dinamički, te usmjeriti sve pokazivače iz polja pokazivača na prve elemente svakog reda dvodimenzionalnog polja što uvodi dodatne međukorake u procesu predavanja dvodimenzionalnog polja funkciji.
Obratiti pozornost kako se uz memorijsku adresu prvog elementa iz polja pokazivača predaje i broj redova i stupaca, što je obavezno, jer u C programskom jeziku ne postoji mehanizam kojim bi se otkrilo unutar funkcije koliko polje ima elemenata, te je potrebno eksplicitno navesti u pozivu funkcije te pripremiti formalne parametre koji će primiti tu informaciju prilikom poziva funkcije.

Primjer 20: Primjer predavanja statički zauzetog dvodimenzionalnog polja funkciji, primjenom pokazivača na polje.

```c
#include<stdio.h>

void ispis2D(int(*)[3], int, int);

int main(void) {

	int polje2D[][3] = { { 1, 2, 3 },
			     { 4, 5, 6 },
			     { 7, 8, 9 },
			     { 10, 11, 12 } };

	ispis2D(polje2D, 4, 3);

	return 0;
}

void ispis2D(int (*p)[3], int m, int n) {
	int i, j;

	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++) {

			printf("%3d", *(*(p + i) + j));
		}
		printf("\n");
	}
}
```

U ovome se primjeru dvodimenzionalno polje predaje funkciji tako da se u funkciji za parametar koji će primiti prvi element dvodimenzionalnog polja koristi pokazivač na polje. Nedostatak ovog koncepta je u tome što je potrebno parametru funkcije koji je pokazivač na polje navesti unaprijed koliko elemenata sadrži red dvodimenzionalnog polja. I zbog toga se ovaj način predavanja dvodimenzionalnog polja funkciji ne može generalizirati za dvodimenzionalno polje različitog broja elemenata podpolja.

Obratiti pozornost kako se uz memorijsku adresu prvog podpolja dvodimenzionalnog polja predaje i broj redova i stupaca, što je obavezno, jer u C programskom jeziku ne postoji mehanizam kojim bi se otkrilo unutar funkcije koliko polje ima elemenata, te je potrebno eksplicitno navesti u pozivu funkcije te pripremiti formalne parametre koji će primiti tu informaciju prilikom poziva funkcije.

Primjer 21: Primjer predavanja dinamički zauzetog dvodimenzionalnog polja funkciji, primjenom dvostrukog pokazivača.

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int** zauzimanje(int, int);
void popunjavanje(int**, int, int);
void ispis2D(int**, int, int);
int** oslobadjanje(int**, int);

int main(void) {

	srand((unsigned)time(NULL));
	int **polje2D = NULL;

	polje2D = zauzimanje(4, 3);

	if(polje2D == NULL){
		return 1;
	}

	popunjavanje(polje2D, 4, 3);
	ispis2D(polje2D, 4, 3);

	polje2D = oslobadjanje(polje2D, 4);

	return 0;
}

int** zauzimanje(int m, int n) {

	int **polje2D = NULL;

	polje2D = (int**)calloc(m, sizeof(int*));

	if(polje2D == NULL){
		return NULL;
	}

	for (int i = 0; i < m; i++) {
		*(polje2D + i) = (int*)calloc(n, sizeof(int));

		if(*(polje2D + i) == NULL){
			return NULL;
		}
	}
	return polje2D;
}

void popunjavanje(int **p, int m, int n) {
	int i, j;

	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++) {
			*(*(p + i) + j) = 1 + (float)rand() / RAND_MAX * (11 - 1);
		}
	}
}
```

```
void ispis2D(int **p, int m, int n) {
	int i, j;

	for (i = 0; i < m; i++)
	{
		for (j = 0; j < n; j++)
		{
			printf("%3d", *(*(p + i) + j));
		}
		printf("\n");
	}
}

int** oslobadjanje(int **p, int m) {
	int i;

	for (i = 0; i < m; i++)
	{
		free(*(p + i));
	}

	free(p);

	return NULL;
}
```

Ovim primjerom je pokazano kako primijeniti funkcije za rad s dvodimenzionalnim poljima. Funkcijom *zauzimanje()* se dinamički zauzima memorijski prostor za dvodimenzionalno polje, funkcijom *popunjavanje()* se popunjava dvodimenzionalno polje, funkcijom *ispis()* se ispisuje dvodimenzionalno polje u matričnom obliku, a funkcijom *oslobadjanje()* se oslobađa memorija od dinamički zauzetog dvodimenzionalno polja. Potrebno je primijetiti kako funkcija *zauzimanje()* vraća memorijsku adresu primjenom dvostrukog pokazivača, što označava da funkcija vraća memorijsku adresu jednostrukog pokazivača, tj. memorijsku adresu prvog jednostrukog pokazivača iz polja pokazivača. Kod funkcije *oslobadnjanje()* potrebno je primijetiti kako je povratna vrijednost isto memorijska adresa. Jedina namjena ove funkcije je da oslobodi dinamičku zauzetu memoriju i vrati *NULL* memorijsku adresu u točku svoga poziva i s tom vrijednosti postavi inicijalni pokazivač koji se prethodno koristio i pokazivao na memorijsku prostor koji je bio dinamički zauzet na *NULL* memorijsku adresu. Ovim načinom se izbjeglo da pokazivač ostane tzv. viseći pokazivač koji pokazuje na memorijski prostor koji je oslobođen što je potencijalno opasno, a postavljanjem iskorištenog pokazivača na *NULL* memorijsku adresu je dobra praksa kojom se izbjegavaju neželjene posljedice ilegalnom pristupu nekoj memorijskoj adresi preko pokazivača koja je oslobođena od upotrebe.

Zadaci za vježbu:

1. Napisati C program koji omogućava unos broja *n*, $(1 \leq n \leq 7)$ koji predstavlja dane u tjednu. Ukoliko je unesen broj koji ne odgovara broju dana u tjednu, ispisati poruku "*Krivo uneseni broj za dan*" i zatražiti novi unos broja *n*. Za inicijalizaciju imena dana koristiti polje pokazivača. Ispisati traženi dan. Koristiti isključivo pokazivačku notaciju!

2. Proširiti prvi zadatak tako da se provjerio koji dan ima najviše suglasnika, te ispisati naopako dan s najvećim brojem suglasnika. Koristiti isključivo pokazivačku notaciju!

3. Napisati C program koji omogućava unos broja *n*, $(1 \leq n \leq 12)$ koji predstavlja mjesece u godini. Ukoliko je unesen broj koji ne odgovara broju mjeseca u godini, ispisati poruku "*Krivo uneseni broj za mjesec*" i zatražiti novi unos broja *n*. Za inicijalizaciju imena mjeseci koristiti polje pokazivača. Ispisati traženi mjesec. Koristiti isključivo pokazivačku notaciju!

4. Proširiti treći zadatak tako da se provjeri koji mjesec ima najveću srednju vrijednost znakova u imenu, te tako pronađenom mjesecu ispisati naopako ime. Koristiti isključivo pokazivačku notaciju!

5. Napisati C program koji omogućava unos broja *n* $(2 \leq n \leq 8)$ koji predstavlja broj redova i stupaca matrice, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za realnu matricu *mat* (primjenom dinamičkog zauzimanja ne kontinuirane memorije za dvodimenzionalno polje), te ju popuniti pseudo-slučajnim vrijednostima iz intervala [-1.25, 5,75], popunjavanje matrice obaviti pomoću funkcije. Ispisati matricu *mat* u matričnom obliku, ispis matrice obaviti pomoću funkcije. Pronaći najveći element matrice *mat* iznad sporedne i glavne dijagonale i ispisati rezultat na standardnom izlazu, pronalaženje najvećeg elementa obaviti pomoću funkcije koja će ujedno vratiti najveći pronađeni element, te povratnu vrijednost funkcije ispisati u glavnom dijelu programa. Na kraju programa osloboditi dinamički zauzetu memoriju, oslobađanje obaviti pomoću funkcije. Koristiti isključivo pokazivački notaciju!

6. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$ i stupaca *n* $(5 \leq n \leq 25)$, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja ne kontinuirane memorije za dvodimenzionalno polje), te ju popuniti na sljedeći način: kako bi se unijele vrijednosti u matricu prvo je potrebno posebno kreirati i popuniti jednodimenzionalno cjelobrojno polje od 8 elementa s generiranim pseudo-slučajnu vrijednosti (0 i 1). Tako popunjeno cjelobrojno polje predstavlja binarni broj koji je potrebno pretvoriti u dekadski ekvivalent koji će se upisati u matricu *mat* (pripaziti ako je generirani negativan binarni broj). Npr. ako je matrica 2×5, morat će se posebno 10 puta popuniti cjelobrojno polje s P-S vrijednostima matrice iz kojeg će se izračunati dekadski ekvivalent koji će se tada upisati u određeni element unutar matrice *mat*. Izračunati srednju vrijednost matrice, te ju ispisati u glavnom dijelu programa. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!

7. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$ i stupaca *n* $(5 \leq n \leq 25)$, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja kontinuirane memorije za dvodimenzionalno polje prvim pristupom), te ju popuniti na sljedeći način: kako bi se unijele vrijednosti u matricu prvo je potrebno posebno kreirati i popuniti jednodimenzionalno cjelobrojno polje od 8 elementa s generiranim pseudo-slučajnu vrijednosti (0 i 1). Tako popunjeno cjelobrojno polje predstavlja binarni broj koji je potrebno pretvoriti u dekadski ekvivalent koji će se upisati u matricu *mat* (pripaziti ako je generirani negativan binarni broj). Npr. ako je matrica 2×5, morat će se posebno 10 puta popuniti cjelobrojno polje s P-S vrijednostima matrice iz kojeg će se izračunati

dekadski ekvivalent koji će se tada upisati u određeni element unutar matrice *mat*. Pronaći najveću vrijednost matrice, te u glavnom dijelu programa ispisati koliko se puta pojavljuje unutar matrice. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!

8. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$ i stupaca *n* $(5 \leq n \leq 25)$, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja kontinuirane memorije za dvodimenzionalno polje drugim pristupom), te ju popuniti na sljedeći način: kako bi se unijele vrijednosti u matricu prvo je potrebno posebno kreirati i popuniti jednodimenzionalno cjelobrojno polje od 8 elementa s generiranim pseudo-slučajnu vrijednosti (0 i 1). Tako popunjeno cjelobrojno polje predstavlja binarni broj koji je potrebno pretvoriti u dekadski ekvivalent koji će se upisati u matricu *mat* (pripaziti ako je generirani negativan binarni broj). Npr. ako je matrica 2×5, morat će se posebno 10 puta popuniti cjelobrojno polje s P-S vrijednostima matrice iz kojeg će se izračunati dekadski ekvivalent koji će se tada upisati u određeni element unutar matrice *mat*. Pronaći najmanju vrijednost matrice, te u glavnom dijelu programa ispisati koliko se puta pojavljuje unutar matrice. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!

9. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$, a *n* iznosi 32 i predstavlja stupce matrice, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja ne kontinuirane memorije za dvodimenzionalno polje), te ju popuniti na sljedeći način: u matricu *mat* unosi se onoliko realnih vrijednosti koliko ima redova. Za svaki red matrice potrebno je generirati pseudo-slučajno realnu vrijednosti iz (-2, -1] i [1, 2) te takvu realnu vrijednost pretvoriti u binarni ekvivalent pomoću IEEE 754 zapis realnog broja jednostrukom preciznosti (pripaziti ako je generirani realni broj negativan). Odabir negativnog ili pozitivnog broja odabrati pseudo-slučajnim odabirom. Svaki red matrice ima 32 elementa koji predstavljaju podpolje u koje se trebaju spremiti (0 i 1) od pretvorenog realnog broja. Pronaći i u glavnom dijelu programa ispisati red matrice koji ima najviše jedinica. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!

10. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$, a *n* iznosi 32 i predstavlja stupce matrice, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja ne kontinuirane memorije za dvodimenzionalno polje), te ju popuniti na sljedeći način: u matricu *mat* unosi se onoliko realnih vrijednosti koliko ima redova. Za svaki red matrice potrebno je generirati pseudo-slučajno realnu vrijednosti iz (-1, 0] i [0, 1) te takvu realnu vrijednost pretvoriti u binarni ekvivalent pomoću IEEE 754 zapis realnog broja jednostrukom preciznosti (pripaziti ako je generirani realni broj negativan). Odabir negativnog ili pozitivnog broja odabrati pseudo-slučajnim odabirom. Svaki red matrice ima 32 elementa koji predstavljaju podpolje u koje se trebaju spremiti (0 i 1) od pretvorenog realnog broja. Pronaći i u glavnom dijelu programa ispisati red matrice koji ima najviše nula. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!

11. Napisati C program koji omogućava unos broja redova *m* $(2 \leq m \leq 20)$, a *n* iznosi 32 i predstavlja stupce matrice, ponavljati sve dok unos ne odgovara traženom intervalu. Dinamički zauzeti memorijski prostor za cjelobrojnu matricu *mat* (primjenom dinamičkog zauzimanja ne kontinuirane memorije za dvodimenzionalno polje), te ju popuniti na sljedeći način: u matricu *mat* unosi se onoliko realnih vrijednosti koliko ima redova. Za svaki red matrice potrebno je generirati pseudo-slučajno realnu vrijednosti iz (-∞, -2] i [2, ∞) te popuniti cjelobrojni red s (0 i 1) koji predstavljaju IEEE 754 zapis realnog broja te takvu realnu vrijednost pretvoriti u binarni ekvivalent

pomoću IEEE 754 zapis realnog broja jednostrukom preciznosti (pripaziti ako je generirani realni broj negativan). Odabir negativnog ili pozitivnog broja odabrati pseudo-slučajnim odabirom. Svaki red matrice ima 32 elementa koji predstavljaju podpolje u koje se trebaju spremiti (0 i 1) od pretvorenog realnog broja. Pronaći i u glavnom dijelu programa ispisati red matrice koji ima jednak broj jedinica i nula. Ispisati matricu *mat* u matričnom obliku. Svaku radnju obaviti isključivo funkcijama! Koristiti isključivo pokazivački notaciju!