Programiranje 2

Laboratorijske vježbe

LV6
Strukture i funkcije

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

Uvod

Primjenom funkcija kôd postaje modularniji, pregledniji i smanjuje se ponavljanje kôda, te samim time omogućava se generaliziranje i ponovna iskoristivost određene logike u programu. Ovim laboratorijskim vježbama pokazat će se način predaje strukture kao argument funkciji, i što funkcija može vratiti kao vrijednost.

Prijenos parametara po vrijednosti

Funkciji se kao parametar može proslijediti varijabla strukture na isti način kao bilo koja druga varijabla. Jedan od načina prosljeđivanja varijable strukture kao parametar funkciji je po vrijednosti. Ovim načinom se stvara lokalna kopija strukture unutar funkcije te se nad članovima te lokalne kopije vrše operacije.

Primjer 1: Prijenos varijable strukture funkciji po vrijednosti, lokalna kopija.

```
#include<stdio.h>

typedef struct kompleksni {
	float re;
	float im;
}KOMPLEKSNI;

KOMPLEKSNI zbroj(const KOMPLEKSNI, const KOMPLEKSNI);
KOMPLEKSNI unos(void);
void ispis(const KOMPLEKSNI);

int main(void) {

	KOMPLEKSNI br1 = { 0 };
	KOMPLEKSNI br2 = { 0 };
	KOMPLEKSNI rez = { 0 };

	br1 = unos();
	br2 = unos();
	rez = zbroj(br1, br2);
	ispis(rez);

	return 0;
}

KOMPLEKSNI zbroj(const KOMPLEKSNI zbrojBr1, const KOMPLEKSNI zbrojBr2) {

	KOMPLEKSNI zbrojRez = { 0 };

	zbrojRez.re = zbrojBr1.re + zbrojBr2.re;
	zbrojRez.im = zbrojBr1.im + zbrojBr2.im;

	return zbrojRez;
}

KOMPLEKSNI unos(void) {

	KOMPLEKSNI br = { 0 };

	printf("Unesite realnu komponentu broja\n");
	scanf("%f", &br.re);
	printf("Unesite imaginarnu komponentu broja\n");
	scanf("%f", &br.im);

	return br;
}
```

```
void ispis(const KOMPLEKSNI rez) {

	if (rez.im < 0) {
		printf("%.2f - %.2f*i\n", rez.re, rez.im * -1);
	}
	else {
		printf("%.2f + %.2f*i\n", rez.re, rez.im);
	}
}
```

Kada se funkciji kao argument predaje varijabla strukture, a formalni parametar funkcije je deklaracija varijable strukture istog tipa kao i argument koji se šalje, članovi formalnog parametra primaju kopije vrijednosti članova argumenta koji se šalje u funkciju. To znači kako se u funkciji radi s kopijom varijable strukture, te kako izmjena vrijednosti članova kopije strukture neće utjecati na originalnu strukturu koja je predana funkciji. Dobra je praksa navesti ključnu riječ *const* ispred deklaracije formalnih parametara, ako će se sadržaj formalnog parametra koristiti samo za čitanje podataka, bez da se izmjenjuje sadržaj unutar formalnog parametra. Ključnom riječi *const* sadržaj varijable se postavlja na konstantnu vrijednost.

Prijenos parametara po adresi

Drugi način prosljeđivanja varijable strukture kao parametar funkciji je po adresi. Ovim načinom se unutar funkcije omogućava direktna izmjena sadržaja članova varijable strukture jer se funkciji proslijedila memorijska adresa varijable strukture.

Primjer 2: Prijenos memorijske adrese varijable strukture funkciji.

```c
#include<stdio.h>

typedef struct kompleksni {
	float re;
	float im;
}KOMPLEKSNI;

KOMPLEKSNI zbroj(const KOMPLEKSNI * const, const KOMPLEKSNI * const);
void unos(KOMPLEKSNI * const, KOMPLEKSNI * const);
void ispis(const KOMPLEKSNI * const);

int main(void) {

	KOMPLEKSNI br1 = { 0 };
	KOMPLEKSNI br2 = { 0 };
	KOMPLEKSNI rez = { 0 };

	unos(&br1, &br2);
	rez = zbroj(&br1, &br2);
	ispis(&rez);

	return 0;
}

KOMPLEKSNI zbroj(const KOMPLEKSNI * const zbrojBr1,
const KOMPLEKSNI * const zbrojBr2) {

	KOMPLEKSNI zbrojRez = { 0 };

	zbrojRez.re = zbrojBr1->re + zbrojBr2->re;
	zbrojRez.im = zbrojBr1->im + zbrojBr2->im;

	return zbrojRez;
}

void unos(KOMPLEKSNI * const zbrojBr1, KOMPLEKSNI * const zbrojBr2) {

	printf("Unesite realnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->re);
	printf("Unesite imaginarnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->im);
	printf("Unesite realnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->re);
	printf("Unesite imaginarnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->im);
}
```

```
void ispis(const KOMPLEKSNI * const rez) {

	if (rez->im < 0) {
		printf("%.2f - %.2f*i\n", rez->re, rez->im * -1);
	}
	else {
		printf("%.2f + %.2f*i\n", rez->re, rez->im);
	}
}
```

Kada se funkciji kao argument predaje memorijska adresa varijable strukture, a formalni parametar funkcije je deklaracija pokazivača na strukturu istog tipa kao i argument koji se šalje, formalni parametar prima memorijsku adresu argumenta. To znači kako se u funkciji radi s originalnom varijablom strukture pristupajući joj indirektno preko pokazivača na strukturu, te kako izmjena vrijednosti članova varijable strukture preko pokazivača na strukturu direktno utječe na originalnu strukturu koja se nalazi izvan funkcije. Dobra je praksa navesti ključnu riječ *const* ispred deklaracije formalnog parametara, ako će se članovi varijable strukture na memorijskoj adresi koja se nalazi u formalnom parametru koristiti samo za čitanje podataka. Ako se ključna riječ *const* postavi iza znaka zvjezdice (*) i ispred identifikatora formalnog parametra, tada taj pokazivač postaje konstantni pokazivač na memorijsku adresu koja je proslijeđena kao argument funkciji, te takav pokazivač nije moguće unutar funkcije preusmjeriti na neku drugu memorijsku adresu.

Pokazivač kao povratni tip podatka

Funkcija kao povratni tip podatka može vratiti pokazivač, odnosno memorijsku adresu određenog osnovnog i/ili složenog tipa podatka, ili memorijsku adresu početka nekog memorijskog bloka.

Primjer 3: Dinamičkog zauzimanja memorije za strukturu i vraćanja memorijske adrese strukture pomoću korisnički kreirane funkcije.

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

typedef struct kompleksni {
	float re;
	float im;
}KOMPLEKSNI;

KOMPLEKSNI* zbroj(const KOMPLEKSNI * const, const KOMPLEKSNI * const);
void unos(KOMPLEKSNI * const, KOMPLEKSNI * const);
void ispis(const KOMPLEKSNI * const);

int main(void) {

	KOMPLEKSNI br1 = { 0 };
	KOMPLEKSNI br2 = { 0 };
	KOMPLEKSNI *rez = NULL;

	unos(&br1, &br2);
	rez = zbroj(&br1, &br2);
	ispis(rez);
	free(rez);
	rez = NULL;

	return 0;
}

KOMPLEKSNI* zbroj(const KOMPLEKSNI * const zbrojBr1,
const KOMPLEKSNI * const zbrojBr2) {

	KOMPLEKSNI *zbrojRez = NULL;

	zbrojRez = (KOMPLEKSNI*)calloc(1, sizeof(KOMPLEKSNI));

	zbrojRez->re = zbrojBr1->re + zbrojBr2->re;
	zbrojRez->im = zbrojBr1->im + zbrojBr2->im;

	return zbrojRez;
}
```

```
void unos(KOMPLEKSNI * const zbrojBr1, KOMPLEKSNI * const zbrojBr2) {

	printf("Unesite realnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->re);
	printf("Unesite imaginarnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->im);
	printf("Unesite realnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->re);
	printf("Unesite imaginarnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->im);
}


void ispis(const KOMPLEKSNI * const rez) {

	if (rez->im < 0) {
		printf("%.2f - %.2f*i\n", rez->re, rez->im * -1);
	}
	else {
		printf("%.2f + %.2f*i\n", rez->re, rez->im);
	}
}
```

Unutar funkcije *zbroj()* dinamički je zauzet memorijski prostor za varijablu strukture *KOMPLEKSNI*. Nakon što su izvršene operacije zbrajanja nad članovima formalnih parametara, te su rezultati zbrajanja realnih i imaginarnih članova spremljeni u članove varijable strukture za koju je dinamički zauzet memorijski prostor, pomoću funkcije *zbroj()* vraća se memorijska adrese početka dinamički zauzetog memorijskog prostora za varijablu strukture *KOMPLEKSNI*.

Primjer 4: Pronalazak strukture KOMPLEKSNI unutar polja struktura KOMPLEKSNI koja ima najveću vrijednost modula, vraćanje memorijske adrese pronađene strukture i daljnja obrada strukture na temelju prosljeđivanja memorijske adrese strukture KOMPLEKSNI u funkcije.

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

#define DG -100.0
#define GG 100.0

typedef struct kompleksni {
	float re;
	float im;
}KOMPLEKSNI;

int duljinaPolja(void);
KOMPLEKSNI* zauzimanjePolje(const int);
void popunjavanjePolja(KOMPLEKSNI* const, const int);
KOMPLEKSNI* pretrazivanjePolja(const KOMPLEKSNI* const, const int);
void ispisBroja(const KOMPLEKSNI* const);
float modulBroja(const KOMPLEKSNI* const);
KOMPLEKSNI* oslobadjanjePolja(KOMPLEKSNI*);
```

```c
int main(void) {

	int n = 0;
	KOMPLEKSNI* poljeKompleksnih = NULL;
	KOMPLEKSNI* maksModulBroj = NULL;

	printf("Unos duljine polja!\n");
	n = duljinaPolja();
	printf("Duljina polja: %d\n", n);

	printf("Zauzimanje polja kompleksnih brojeva!\n");
	poljeKompleksnih = zauzimanjePolje(n);

	if (poljeKompleksnih == NULL) {
		printf("Neuspjesno zauzimanje memorije!");
		return 1;
	}

	srand((unsigned)time(NULL));

	printf("Popunjavanje polja kompleksnih brojeva!\n");
	popunjavanjePolja(poljeKompleksnih, n);

	printf("Pretrazivanje polja kompleksnih brojeva!\n");
	maksModulBroj = pretrazivanjePolja(poljeKompleksnih, n);

	printf("Ispis kompleksnog broja!\n");
	ispisBroja(maksModulBroj);

	printf("Oslobadjanje memorije od polja kompleksnih brojeva!\n");
	poljeKompleksnih = oslobadjanjePolja(poljeKompleksnih);

	return 0;
}

int duljinaPolja(void) {

	int temp = 0;

	printf("Unesite duljinu polja!\n");

	do {
		scanf("%d", &temp);

		if (temp <= 0) {
			printf("Krivi unos, unesite nenegativnu vrijednost koja je i\
			veca od nule!\n");
		}
	} while (temp <= 0);

	return temp;
}
```

```
KOMPLEKSNI* zauzimanjePolje(const int duljinaPolja) {

	KOMPLEKSNI* tempZauzimanje = NULL;

	tempZauzimanje = (KOMPLEKSNI*)calloc(duljinaPolja, sizeof(KOMPLEKSNI));

	if (tempZauzimanje == NULL) {

		return NULL;
	}

	return tempZauzimanje;
}

void popunjavanjePolja(KOMPLEKSNI* const poljeKompleksnih,
const int duljinaPolja) {

	int i = 0;

	for (i = 0; i < duljinaPolja; i++)
	{
	(poljeKompleksnih + i)->re = DG + (float)rand() / RAND_MAX * (GG - DG);
	(poljeKompleksnih + i)->im = DG + (float)rand() / RAND_MAX * (GG - DG);
		printf("%d. ", i + 1);
		ispisBroja(poljeKompleksnih + i);
	}

	return;
}

KOMPLEKSNI* pretrazivanjePolja(const KOMPLEKSNI* const poljeKompleksnih,
const int duljinaPolja) {

	int i = 0;
	int maksI = 0;
	float maksModul = 0.0;
	float tempModul = 0.0;

	for (i = 0; i < duljinaPolja; i++)
	{
		if (i == 0) {
			maksModul = tempModul = modulBroja((poljeKompleksnih + i));

		}
		else {
			tempModul = modulBroja((poljeKompleksnih + i));

			if (maksModul < tempModul) {
				maksModul = tempModul;
				maksI = i;
			}
		}
	}

	return (poljeKompleksnih + maksI);
}
```

```
void ispisBroja(const KOMPLEKSNI* const maksModulBroj) {

	printf("broj:\t");

	if (maksModulBroj->im < 0.0f) {
	printf("%+5.2f - %5.2f*i ", maksModulBroj->re, maksModulBroj->im * -1.0f);
	}
	else {
	printf("%+5.2f + %5.2f*i ", maksModulBroj->re, maksModulBroj->im);
	}

	printf("\tmodul broja: %5.2f\n", modulBroja(maksModulBroj));

	return;
}

float modulBroja(const KOMPLEKSNI* const kompleksniBroj) {

	return (float)sqrt(pow(kompleksniBroj->re, 2) + pow(kompleksniBroj->im, 2));
}

KOMPLEKSNI* oslobadjanjePolja(KOMPLEKSNI* poljeKompleksnih) {

	free(poljeKompleksnih);

	return NULL;
}
```

Datoteke zaglavlja i strukture

Kada se program rasporedi u više datoteka, dobra je praksa u datoteku zaglavlja smjestiti deklaraciju strukture. Ovim načinom se može po potrebi u svaku datoteku izvornog kôd-a umetnuti datoteka zaglavlja unutar koje se nalazi definicija strukture, te po potrebi kreirati varijablu te strukture.

Primjer 4: Raspoređivanja pojedinog dijela izvornog kôd-a u za to namijenjenu datoteku.

header.h

```
#ifndef HEADER_H
#define HEADER_H

typedef struct kompleksni {
	float re;
	float im;
}KOMPLEKSNI;

KOMPLEKSNI* zbroj(const KOMPLEKSNI * const, const KOMPLEKSNI * const);
void unos(KOMPLEKSNI * const, KOMPLEKSNI * const);
void ispis(const KOMPLEKSNI * const);

#endif // HEADER_H
```

main.c

```
#include <stdio.h>
#include "header.h"

int main(void) {

	KOMPLEKSNI br1 = { 0 };
	KOMPLEKSNI br2 = { 0 };
	KOMPLEKSNI *rez = NULL;

	unos(&br1, &br2);
	rez = zbroj(&br1, &br2);
	ispis(rez);
	free(rez);
	rez = NULL;
	return 0;
}
```

functions.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "header.h"

KOMPLEKSNI* zbroj(const KOMPLEKSNI * const zbrojBr1,
const KOMPLEKSNI * const zbrojBr2) {

	KOMPLEKSNI *zbrojRez = NULL;

	zbrojRez = (KOMPLEKSNI*)calloc(1, sizeof(KOMPLEKSNI));

	zbrojRez->re = zbrojBr1->re + zbrojBr2->re;
	zbrojRez->im = zbrojBr1->im + zbrojBr2->im;

	return zbrojRez;
}

void unos(KOMPLEKSNI * const zbrojBr1, KOMPLEKSNI * const zbrojBr2) {

	printf("Unesite realnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->re);
	printf("Unesite imaginarnu komponentu prvog broja\n");
	scanf("%f", &zbrojBr1->im);
	printf("Unesite realnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->re);
	printf("Unesite imaginarnu komponentu drugog broja\n");
	scanf("%f", &zbrojBr2->im);
}

void ispis(const KOMPLEKSNI * const rez) {

	if (rez->im < 0) {
		printf("%.2f - %.2f*i\n", rez->re, rez->im * -1);
	}
	else {
		printf("%.2f + %.2f*i\n", rez->re, rez->im);
	}
}
```

Zadaci

1. Napisati C program koji omogućuje unos dva broja *m* i *n* sa standardnog ulaza, omogućiti unos duljine polja pomoću funkcije *unosDuljine()*. Uzeti u obzir interval za *m* (4 < *m* <= 100) i za *n* (3 <= *n* < 16). Na temelju broj *m* i *n* dinamički zauzeti memorijski prostor za *m* struktura točka i *n* struktura trokut (u potpunosti rukovati memorijom), pomoću funkcija *zauzimanjeTocaka()* i *zauzimanjeTrokutova()*. Struktura *tocka* sadrži dva cjelobrojna člana, a struktura *trokut* sadrži tri člana koji su pokazivači na strukturu *tocka*. Popuniti koordinate točaka pseudo-slučajnim odabirom iz intervala [0, m], koristeći funkciju *popunjavanjeKoordinata()*. Pseudo-slučajnim odabirom dodijeliti točke pojedinim vrhovima trokuta, koristeći funkciju *odabirVrha()*. Napisati funkciju duljina*Stranice()* pomoću koje se pronalazi udaljenost između dviju točaka. Napisati funkciju *opsegTrokuta()* pomoću koje se računa opseg trokuta. Pronaći i ispisati trokut koji ima najveći opseg.
   Organizirati program u više datoteka!
   Koristiti isključivo pokazivačku notaciju!

$$d(t1, t2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$O_T = a + b + c$$

2. Napisati C program koji omogućuje unos broja *m* sa standardnog ulaza koji predstavlja duljinu polja, unos broj *m* omogućiti pomoću funkcije *unosDuljine()*, gdje je (0 < m <= 60). Dinamički zauzeti memorijsku prostor za polje artikala pomoću funkcije *kreiranjeArtikala()*, u potpunosti rukovati memorijom. Struktura artikl ima članove ime, cijena i kolicina. Unos artikala napraviti pomoću funkcije *unosArtikla()*. Pomoću funkcije *pretragaArtikla()*, pronaći najskuplji artikl i vratiti njegovu memorijsku adresu. Ispisati ime, cijenu i količinu najskupljeg artikla pomoću funkcije *ispisArtikla()*.
   Organizirati program u više datoteka!
   Koristiti isključivo pokazivačku notaciju!