Avditorne vaje za Programiranje II

Matej Blagšič

5. april 2018

Pomembno

Te vaje so direktno iz pouka. Koda morda ni identična od te asistena, ampak deluje enako. Prav tako je komentirana in obrazložena skupaj z navodili in opazkami za lažje razumevanje kode. Prav tako se sklicujem in že tu pozivam, da si pogledaš zapiske iz pouka, ki so kot nekakšen učbenik. Notri je snov, teorija in primeri iz pouka. Prijetno branje in učenje želim!

1. vaja

/empty/

2. vaja

Izračunaj $\int_{x_0}^{x_1} 2x^2 - 5x\,dx$. Rezultat preveri analitično.

Če analitično integriramo itegral, dobimo: $\int_{x_0}^{x_1} 2x^2 - 5x\,dx = 2\frac{x_1^3}{3} - 5\frac{x_0^2}{2}$

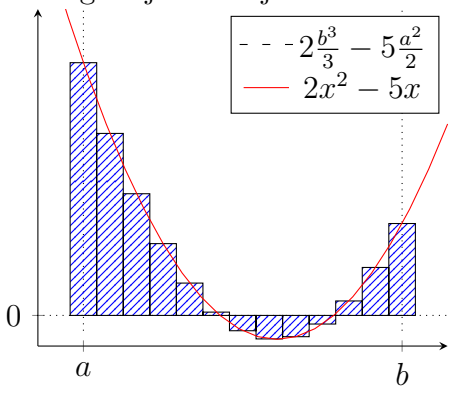
Sedaj spišimo kodo:

```
int main(){
    float x, x0, x1;
    float dx = 0.0000001;
    float integral = 0;
    printf("vnesi spodno mejo");
    scanf("%f",&x0);
    printf("Vnesi zgornjo mejo");
    scanf("%f",&x1);

    for(x=x0; x<x1;x+=dx){
        integral += dx*(2*x*x-5*x);
    }

    printf("Integral znasa: %f\n", integral);
    return 0;
}
```

Pri programu nam spremenljivka dx sporoči, kako širok del območja integrira. Manjša, kot je cifra, bolj natančno izračuna. x0 in x1 sta spodnja in zgornja

meja integracije, x pa je spremenjivka, ki jo premikamo po intervalu za dx razdaljo in seštevamo pravokotnike.

Slika 1: Integracije funkcije na intervalu $[a, b]$



3. vaja

1. naloga

Napiši program, ki izpiše in izračuna faktorielo(fakulteta) nekega števila(1-20)

Pri temu programu spoznamo omejitve velikosti spremenljivk. Pomemno je, da so števila, ki jih hočemo hraniti in predstavljani v polni natančnosti ne presegajo velikosti spremenljivke. Če ne, potem se prične pačenje podatkov. Vidimo, da lahko uporabimo izrad long long in s tem povečamo obseh navadnega long tipa. Podobno lahko naredimo tudi s spremenljivkam s plavajočo vejico, recimo `long double`.

```c
int main(){
    unsigned long long resitev = 1, n;

    for(n=1; n<=20; n++){
        for(int i=1; i<=n; i++){
            resitev *= i;
        }
        printf("Faktoriela od %lld! je %lld\n", n, resitev);
        resitev = 1;
    }
    return 0;
}
```

2. naloga

Napiši program, ki sešteje maso lune in zemlje ter sonca.

Namen vaje je dodatno spoznati omejitve spremenljivk v programskem jeziku C. Mase lune, zemlje in sonca so ogromne, zato se začenja poznati popačenje podatkov. Ugotovili smo, da dobimo kar se da dobre rezultate, če uporabimo `double`, ki ima največji obseh, a se vseeno na določenem mestu pojavi naključna številka, ki ni podana med vhodnimi podatki.

```c
int main(){
    double luna = 7.348e22;
    double zemlja = 5.972e24;
    double sonce = 1.989e30;

    printf("Masa lune je: %40.0lf \n",luna);
    printf("Masa zemlje je: %40.0lf \n",zemlja);
    printf("Masa sonca je: %40.0lf \n",sonce);
    printf("Skupna masa je: %40.0lf\n", zemlja+luna);
    printf("Skupna masa sonca pa lune je: %40.0lf\n", sonce+luna);

    return 0;
}
```

4. vaja

1. naloga

Napiši funkcijo, ki sešteje dve razdalji podani v čevljih in palcjh(feet and inches)

Vemo, da je 1 čevelj 12 palcev. To potrebujemo, da pretvorimo palce v čevlje, kajti če pri vsoti dobimo recimo 13 palcev, pretvorimo to v en čevelj in en palec.

```
#include <stdio.h>

struct razdalja{
    int foot;
    int inch;
};
struct razdalja sestevanje(struct razdalja x, struct razdalja y);
int main(){
    struct razdalja a, b, r;
    printf("Vnesi prvo razdaljo v obliki a b ");
    scanf("\n %d\'%d\"", &a.foot, &a.inch);
    printf("Vnesi drugo razdaljo v obliki a b ");
    scanf("\n %d\'%d\"", &b.foot, &b.inch);
    r = sestevanje(a, b);
    printf("%d %d\n", r.foot, r.inch);
    return 0;
}
struct razdalja sestevanje(struct razdalja x, struct razdalja y){
    struct razdalja z;
    z.inch = x.inch + y.inch;
    z.foot = x.foot + y.foot;
    if(z.inch >=12){
        z.foot++;
        z.inch -=12;
    }
    return z;
}
```

Vidimo, da smo definirali novo funkcijo za seštevanje, katere tip je enak izhodnemu podatku, torej novemu tipu razdalja, ki jo definiramo s struct razdalja. Kako se uporablja struct ukaz in funkcije, si poglej v predavanje zapiskih v 4. in 5. poglavju.

2. naloga

Imamo program, ki nas nauči o vrstah konstant pri primerjavah.

V temu programu spoznamo, da v C-ju so vse konstante tipa `double`. To je zelo pomembno. Poglejmo si priložen program. Hočemo primerjati x z njegovo vrednostjo. Program nam vrne nič, kot da nista iste, čeprav sta. Naš dvojni enačaj je tipa int, a konstanta 0.2, ki se primerja z x, je pa tipa double in ne float, kot smo hoteli definirati x. Zato popravimo spremenljivko x v double.

```c
#include <stdio.h>

int main(){
    double x = 0.2; //prej: float x = 0.2;
    printf("%d\n", x == 0.2);
    return 0;
}
```

5. vaja

1. naloga

Opazi napake v main funkciji danega programa:

```c
#include <stdio.h>

int test1(float a, float b);
void test2(double x);
float test3(int x):
int main(){
    int z;float x;
    z = test1(1.1, 0.4);//ok(opozorilo, pretvorba double v float)
    z = test1(2, 3);//ok(pretvorba int v float)
    test1(x, x);//ok
    z = test2(42);//napaka, test2 ne more dobit vrednosti, je void
    x = test3(z, 12);//napaka, prevec parametrov
    z = test3(13);//napaka, funkcija vrne float
    return 0;
}
```

Pomembno je, da se opazi, da v prvi vrstici, se v funkcijjo test1 vnašajo

številke, ki so tipa double. Zato je pomembno, da vemo, da program pretvori iz double v float. Napaka je v 4. vrstici, kjer v funkcijo, ki vrne nič(tipa void), vnašamo podatek. Prav tako je napaka v naslednji vrstici, kjer vnašamo celo število v funkcijo, ki pa vrne float.

2. naloga

Napiši funkcijo, ki vrne 10 naključnih števil

Tu spoznamo funkcijo `rand()`, ki vrne naključno 16-bitno število. Obstaja tudi funkcija `srand(0)` oz. random seed, ki nam vnese neko seme, ki vpliva na naključnost random funkcije.

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

int main(){
    srand(0);
    int i;
    for(i = 0; i<10; i++){
        printf("%d\n", rand());
    }
    return 0;
}
```

Če zaženemo to kodo lahko opazimo, da ne glede na to kokrat zaženemo program ali ga compailamo, nam vrne enaka števila, kar ni ravno naključno. Zato moramo spreminjati vrednost v funckiji `srand(vrednost)`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    time_t t; //poimenujemo spremenljivko za cas
    srand((unsigned) time(&t));//prej je bilo tole --> srand(0);
    for(int i = 0; i<10; i++){
        printf("%d\n", rand());
    }
    return 0;
}
```

3. naloga

Napiši program, ki naključno sortira zbirko učencev

v temu programu potrebujemo metodo urejanja z mehurčki(bubble sort). Ta deluje tako, da primerjamo po dve števili med seboj. Če je prvo večje od drugega, potem ju zamenja. Tako gre program skozi celotni seznam. Ponovo ta postopek, dokler niso razvrščeni vsi elementi.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 500

struct ucenec {
    int visina;
    int starost;
    float teza;
    char spol;
    char ime[20];
    char priimek[20];
};

struct ucenec ekipa[DIM];

int main (){

    int i, j, urejeno, stPrimerjav=0;
    struct ucenec tmp;
    srand(26);

    for (i = 0; i < DIM; i++){ekipa[i].visina = 110 + rand()%21;}
    for (i = 0; i < DIM; i++){
        stPrimerjav=0;
        for (j = 0; j < DIM-1; j++){
            if (ekipa[j+1].visina < ekipa[j].visina){
                tmp = ekipa[j+1];
                ekipa[j+1] = ekipa[j];
                ekipa[j] = tmp;
                stPrimerjav=1;
            }
        }
        if (stPrimerjav == 0){break;}
    }

    return 0;
}
```

Da razložimo kodo. Na začetku se opazi nekaj novega za nas. Imamo funkcijo `#define DIM 500`. Ta funkcija nam izrazu DIM"priredi vrednost 500. Torej vsakič, ko napišemo DIM nam vpiše 500. To ni spremenljivka!

Nato definiramo novo strukturo z imenom učenec. Ta ima vse podatke posameznega učenca. Mi se bomo osredotočili na višino. Tako definiramo zbirko z imenom ekipa tipa ucenec. Ta ima velikost DIM, torej ustvarimo zbirko 500 ucencev, katere zbirka se imenuje ekipa.

Preidemo v main funkcjo in si nastavimo spremenljivke za for zanki ter spremenljivki urejeno in stPrimerjav. Uporabo teh bom razložil kasneje. Definiramo novo spremenljivko tipa ucenec z imenom tmp. Ta predstavlja začasnega učenca". Ta se bo uporabljal, ko bomo premikali ucence po vrstnem redu in bomo morali enega shraniti v nek začasni prostor, da se bodo lahko ucenci v zbirki ekipa prestavili. Več o tem kasneje.

Prva for zanka nam kreira naključne višine za vsakega od učencev v ekipi. Sprehodimo se od prvega do zadnjega in vsakemu priredimo vrednost višine od 110 do 130 naključno. To pa naredimo tako, da deljimo naključno število z 21 in pogledamo njegov ostanek. To nam naredi operator %. Ta ostanek je vedno od 0 do 20, tako da ravno prav in zato je ostanek pri deljenju z 21 za števila od 0 do 20 in ne deljenje z 20!

Ko ima vsak učenec višino, gremo v dvojni for stavek. Prvi for stavek bo največ DIM-krat izvedel primerjavo in zamenjavo mest učencev v zbirki. Drugi for stavek pa primerja in prestavlja učence po zbirki.
Deluje tako, da se pojavi if stavek, ki primerja trenutno višino učenca[i] ter višino od naslednjika ucenec[i+1]. Če je drugi manjši, ju mora zamenjati. To pa naredi tako, da vzame najprej vrednosti od ucenec[i+1] in jih skopira v začasnega učenca tmp, na mesto ucenec[i+1] skopira vrednosti od ucenec[i] in nato na mesto ucenec[i] skopira vrednosti iz tmp oz. prejšni ucenec[i+1].

Tako potuje ta val"po celotni zbirki in premeša po dva in dva elementa, kjer je to potrebno. Prav tako, vsakič ko premeša dva člena, postavi spremenljivko stPrimerjav na 1. Ta spremenljivka nam pove, ali je bila skozi en prehod skozi zbirko narejena primerjava in zamenjava dveh členov. To potrebujemo zato, da na koncu prve for zanke preverimo, če je bila narejena kakšna primerjava v temu i-tem preletu zbirke. Vidimo tudi, da se ta spremenljivka na začetku te for zanke tudi resetira na 0. Torej ostane 0, če se if stavek v drugi for zanki sploh ne izvede. V tem primeru koda izstopi iz for zanke. Na koncu lahko postavimo kako for zanko, ki nam izpiše od 0 do DIM višine vseh učencev in lahko vidimo, da so po vrsti.

6. vaja

1. naloga

Prva naloga je zadnja naloga pri prejšni vaji, tako da si lahko pogledaš tu.

2. naloga

Druga naloga je bolj nadgradnja prve naloge. Sicer problem je, da več kot ima oseba podatkov in več kot je teh oseb, dlje traja sortiranje. Zato bomo rešili zadevo s kazalci. Poglej si snov v zapiskih o kazalcih. Več o tem naslednjič.

> Slečo kodo sem spisal sam doma. Izgleda da deluje, a ni preverjeno s strani asistenta, tako da vzamite kodo in razlago s kančkom soli!

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

#define DIM 20

struct ucenec {//nova struktura
    int id;
    int visina;
    int starost;
    float teza;
    char spol;
    char ime[20];
    char priimek[20];
};

int main(){

    int stevec;//spremenljivke
    srand(time(NULL));
    struct ucenec ekipa[DIM]
    struct ucenec *t[DIM], *tmp;

    for (int i = 0; i < DIM; i++){//izdelava zbirk
        ekipa[i].visina = 110 + rand()%21;
        ekipa[i].id = i;
        t[i] = &ekipa[i];//naslavljamo i-ti kazalec na i-ti struktni clen
    }
```

```
    for (int i = 0; i < DIM; ++i){//premetavanje zbirke t
        stevec = 0;
        for (int j = 0; j < DIM-1; j++){
            if((*t[j+1]).visina < (*t[j]).visina){//ali je drugi mansi
                tmp = t[j+1];
                t[j+1] = t[j];
                t[j] = tmp;
                stevec = 1;
            }
        }
        if(stevec == 0){//ce nismo premetavali, potem je urejeno
            break;
        }
    }

    for (int i = 0; i < DIM; ++i){//izpis elementov
        printf("%d \t %d\n", t[i]->visina, t[i]->id); //izpisi clene
    }

    return 0;
}
```

Koda je malo dolga, a zato bolje pregledna. Na prvi pogled je enaka, kot pri prejšni nalogi. Razlika je le, da tu se vse primerjanje in izpisovanje dela s kazalci. Pa začnimo kar z main funkcijo.

Na začetku je vse enako razen dela: `struct ucenec *t[DIM] *tmp;`.

Namenoma sem dal v svojo vrstico, saj tu definiramo kazalce; glej zvezdico pred spremenljivko! Kazalec `t` je zbirka, ki bo kazala na elemnte v zbirki ekipa. `tmp` je kazalec, ki bo začasno držal naslove, ko se bojo prestavljali v zbirki `t`. Se pravi je ta koda čisto enaka kot prej, le da uporabljamo kazalce, ki imajo še neka svoja pravila, drugače postopek je pa enak.

Pride prva `for` zanka. V tej zgeneriramo naključne vrednosti za višino vsak element v zbirki ekipa. Prav tako vsakemu priredimo zaporeden `id`. To sem dodal, da bomo kasneje videli, da so se naslovi zares premešali. Nato naslovimo kazalec `t` na vsak člen v zbirki ekipa. To naredimo enako, kot smo se naučili, le da tu uporabimo še oglate oklepaje, saj moramo to narediti za i-ti člen v zbirki posebej.

Sedaj pridemo do glavne for zanke, ki nam torej premetava našo zbirko kazalcev `t`.

```
*t[j+1]).visina < (*t[j]).visina
```

```
t[j+1]->visina < t[j]->visina
```

Zgornja dva okvirčka sta ekvivalenta. V kodi se sicer nahaja levi. To je pogoj v `if` stavku, ki sprašuje po tem ali je člen `j+1` manjši od člena `i` ali ne. To je pogoj, da se ta dva člena zamenja.

Zakaj to deluje tako kot deluje? No je malo drugače, ker imamo zbirko, ki je še tipa struct. Ni panike. Najprej povozimo kazalec oz. zahtevamo vrednost, kjer se nahaja podatek v zbirki, na katero kaže kazalec `t` z zvezdico. In iz tega elementa zahtevamo ven podatek visina, kajti vsak podatek v zbirki, na katero kaže kazalec `t`. To je ekvivalentno drugemu okvirčku. Tam nekako s puščico želimo pridobiti podatek iz strukture, na katero kaže kazalec `t`, z imenom visina.

Če je ta pogoj pravilen, moramo člena zamenjati. Ampak ker se hočemo ogniti prestavljanju elementov zbirke ekipa okoli, bomo uporabljali le kazalce.

```
tmp = t[j+1];
t[j+1] = t[j];
t[j] = tmp;
stevec = 1;
```

Levo je koda v `if` stavku. Najprej skopiramo naslov manjšega člena v začasni `tmp`. Nato skopiramo naslov, ki je bil prej za člen j v mesto j+1. Nato nazaj na mesto j skopiramo naslov iz `tmp`.

Na koncu imamo printanje sedaj urejene zbirke kazalcev `t`. Tako, kot smo prej spoznali, lahko z ukazom `t[i]->visina` zahtevamo podatek višina iz mesta, kamor kaže i-ti kazalec `t`.