Skripta iz Osnova programiranja

Teorija za ispit

Nikola Vlahović

Beograd 2017.

Ova skripta sadrži teoriju neophodnu za prvi i drugi kolokvijum. Na ispitu će se pojaviti kombinacija pitanja iz oba kolokvijuma.

01V - Uvodno predavanje

1. Šta je programiranje?

- U suštini pisanje uputstava za računare.

2. Šta je algoritam?

- Zapis neke procedure, korak-po-korak uputstvo kako da se obavi neki zadatak.

* Može biti zapisan kao lista koraka ili kao blok šema.

3. Šta su granjanje i ciklusi?

- Granjanje predstavljaju delovi programa kod kojeg se izvršavanje može nastaviti na više načina.
- Ciklusi su delovi programa koji se mogu više puta uzastopno izvršiti.

4. Šta je izvorni kod?

- Tekst ispisan u prog. jeziku, početni oblik programa koji piše programer koji se potom prevodi/izvršava.

5. Šta je C?

- Jedan od najzastupljenijih i najuticajnijih programskih jezika u istoriji, razvio ga Dennis Ritchie.

6. Šta je kompajliranje?

- Prevođenje izvornog koda u mašinske instrukcije koje procesor može da izvrši.

7. Šta su IDE?

- Integrisana razvojna okruženja, sve-u-jednom rešenja za razvoj softvera (imaju i editor i kompajler).

8. Šta su komentari?

- Delovi koda koji se neće izvršiti, postoje linijski (//) i blok (*/ /*) komentari.

01P - Osnovni elementi jezika

1. Šta su ključne reči? Navedi nekoliko. *// Sintaksa = pravila o pojedinačnim elementima u jeziku.*

- Rezervisane reči koje imaju unapred definisano značenje u samom jeziku, nmg biti identifikatori. If, int...

2. Šta su identifikatori?

- Nazivi koje dajemo elementima u programu. Koriste se za imenovanje promenjivih, funkcija, struktura.

** Pravila: mogu da sadrže sva v/m slova alfabeta, cifre 0-9, donju crtu _. Ne smeju da počnu cifrom.*

3. Šta su literali? Koje vrste literala postoje?

- Eksplicitne vrednosti koje se javljaju u programu. Celobrojni(int), realni/dec(float/double), znakovni(char) i string.

* Celobrojni literali: Decimalni : 1, 18… ; Oktalni (nula na početku, zatim 0-7) : 01, 023, ~~0679~~ ; Heksadec. (0x) : 0xabf
* Na kraju sufiks u/U za unsigned (123u) i l/L za long (456L). Unsigned ne može da piše neg. br., long x2 preciznost.
* Decimalne navodimo sa tačkom (10.0) ili eksp. (e/E) (5e4), dodajemo f/F da bi bio float umesto double (10.0f).

***** Biće najverovatnije prepoznati koji literal je u pitanju (npr. 0x12 - celbr. u hex. sistemu) i da li je broj float ili double.***

***** Takođe je potrebno razlikovati char i string literale (string su proizvoljne dužine (char je samo 1) i idu u navodnike “”).***

4. Šta radi neki od navedenih operatora? ***(biće naveden operator)***

* **=** je dodela, **==** je poređenje; a += 5 je isto što i a = a + 5 (isto tako i -=, /=,*=,%=).
* ! (nije), && (AND), ||(OR). i++ je inkrement (isto što i i=i+1), i-- je dekrement.

02P - Tipovi i formatiranje

1. Navedi osnovne tipove.

- char (znak, 1 bajt), int (ceo broj, 2 bajta), float (realan broj, 4 bajta), double (-ll-, 8 bajta, precizniji).

2. Šta znače short i long? signed i unsigned?

- Ispred tipa int možemo dodati kvalifikator short za manje brojeve (2 bajta) ili long za veće (4 ili 8 bajta).
- Kod int i char: S i U, koji određuju da li će tip imati predznak, ili će i taj bit biti iskorišćen za veću vrednost.

3. Napraviti promenjivu.

- char a; char ovoJeStringOdPetKaraktera[5]; int celiBroj; float realniBroj; double realniBrojPrec; ...

4. Skeniraj i ispiši neku promenjivu.

- int a; scanf("%i", &a); printf("%i", a*);* *=> Deklarišemo, skeniramo, štampamo.*

** <tip prom.> <naziv>; || scanf("%<tip>", &<naziv promenjive>); || printf("%<tip>", <naziv>);*

** %i = int || %**l**i = **long** int || %d = decimal int || %f = float || %lf = long float || %c = char || %s = string*

02V - Granjanja i ciklusi

1. Objasniti ukratko granjanje i cikluse.

- Granjanje nam omogućava da izmenimo tok programa u zavisnosti od zadatih uslova.
- Ciklusi nam omogućavaju da određeni deo programa izvršimo više puta uzastopno.

2. Sintaksa if-a.

1. **if**(uslov){odradi ovo unutra ako se ispuni uslov}
2. **if**(uslov){odradi ovo ako se ispuni uslov (true)}**else**{u suprotnom, odradi ovo}
3. **if**(uslov){odradi ovo ako true} **else if**(uslov){pitaj za ovo ako nije prvo}**else**{u suprotnom ovo}.

3. Razlika između = i ==?

- Jedan znak jednakosti se koristi za dodelu (a=5), dva se koriste kod poređenja (if(a==b)).

4. Navedi operatore za poređenje vrednosti.

- ==, != , >, < , >=, <= : jednako, različito, veće, manje, veće ili jednako, manje ili jednako.

5. Šta su ciklusi i koje vrste petlje postoje? Razlika između njih? Koliko min. puta mora da se izvrši svaka?

- Ciklusi omogućavaju izvršavanje istog bloka koda više puta uzastopno.

* **while**(uslov){ dok je uslov tačan } => Blok naredbi se ponavlja dokle god je zadati uslov tačan. Min. 0.

* **do**{dok je uslov true}**while**(uslov); => Prvo odradi pa pita. Odradiće blok barem jednom.

* **for**(početak; uslov; korak){ telo petlje } => for(i=0;i<10;i++) {printf("%i",i);}

6. Objasni segmente iz for petlje.

- Početak : Stavljamo neki **brojač** (prom. koju koristimo za for petlju) na prvu vrednost koju želimo da ima.
- Uslov : Uslov pod kojim želimo izvršavati petlju, ona će se izvšravati sve dok je ovaj uslov ispunjen.
- Korak : Menjamo vrednost brojača. (i++ ga uvećava za jedan, i+=2 za dva, isto tako imamo i--, i-= itd.)

7. Koju će vrednost imati i nakon for(i=0;i<10;i++){ //kod }? ***//možda pita brojanje unazad (i=10,i>0,i--)***

- 10, jer će taj poslednji put i porasti sa 9 na 10 i naredni put kad pita i<10 neće biti true i neće se izvršiti.

03P - Sortiranje i stabla

1. Šta je sortiranje?

- Proces dovođenja liste u zadati poredak.

2. Vremenska i prostorna složenost?

- Vremenska složenost = složenost u vremenu izvršavanja (br. operacija u odnosu na veličinu ulaznih pdtk).
- Prostorna složenost = zauzeće memorije, koliko je dodatne memorije potrebno da se operacija obavi.

3. Navedi nekoliko naziva algoritama za sortiranje.

- Bubble sort, selection sort, insertion sort.

** Na praktičnom delu biće sortiranje, znati barem 1 algoritam (najlakši za učenje je bubble sort).*

Bubble sort za niz u kome imamo 50 članova: int niz[50], i, ,n, temp;

```
for(i=0;i<50;i++){                  // minus jedan ide jer se porede dva uzastopna člana, a posl. član nema sa kim dalje.
 for(n=0;n<50-1-i;n++){             //Nova promenjiva, skoro isto kao gore samo još minus gornja prom u uslovu.
  if(niz[n]> ili < ili >= ili <= niz[n+1]){  //Znak se menja u zavisnosti kakvo sortiranje nam treba.
    temp = niz[n];          // Prvo u temp ubacimo vrednost od niz[n];
    niz[n] = niz[n+1];      //Potom u niz[n] ubacimo vrednost od niz[n+1];
    niz[n+1] =temp;}        //Onda uzimamo onu vrednost iz temp i dajemo niz[n+1].
        }}                  //Time smo u suštini zamenili mesta vrednostima iz niz[n] i niz[n+1].
```

03V - Nizovi

1. Šta je niz? Deklariši niz dužine 5.

- Imenovana kolekcija promenjivih istog tipa. int duzinaNizaIdeuZagrade[5];

** Indeks niza počinje od nule, najveći član ima indeks jednak dužini-1 :*

** char test[3]="asd" => test[0]="a", test[1]="s", test[2]="d" || Niz ima 3 karaktera, indeks posl. je 2.*

2. Šta su stringovi?

- Nizovi char-ova (karaktera).

3. Razlika između dužine niza i dužine stringa?

- Dužina niza se odnosi na rezervisanu memoriju, dužina stringa se odnosi na dužinu teksta zapisanog u nizu.

4. Šta je null char \0?

- Karakter koji označava kraj stringa u nizu. ** Dužina niza je 10, dužina stringa je 5.*

* Pošto indeksi počinju sa nulom, null char uvek odgovara dužini stringa (Hello ima 5 karaktera, \0 se nalazi na indeksu 5).

* **gets(<ime stringa>);** => Za razliku od scanf-a ne smatra razmak kao kraj stringa.

** **for(i=0; tekst[i]!='\0';i++)** - Petlja koja ide do kraja str a ne niza.

5. Funkcije iz biblioteke <string.h>.

- strlen(string) = vraća dužinu stringa.
- strcpy(string1, string2) = kopira string 2 u string 1.
- strcat(string1, string2) = nadovezuje string 2 u string 1.
- strcmp(string1, string2) = poredi stringove, vraća 0 ako su jednaki, <0 ako je 1 abecedno ispred, >0 drugi.
- strstr(string1, string2) = traži string 2 u string 1 i daje pokazivač na prvo pojavljivanje, inače NULL.

04P - Predprocesor i komandna linija

1. Šta je kompajliranje?

- Prevođenje izvornog koda u mašinske instrukcije koje procesor može da izvrši.

2. Navedi faze prevođenja.

* Predprocesiranje - Izvršavanje predprocesorskih direktiva
* Tokenizacija - Razlaganje teksta programa na elemente jezika (identifikator, literali...)
* Kompajliranje - Generisanje mašinskih instrukcija na osnovu tokena
* Linkovanje - Spajanje jedinica prevođenja i spoljašnjih biblioteka.

3. Predprocesor i predprocesorske direktive.

- Predprocesor priprema kod za dalje prevođenje izvršavajući direktive i tekstualne zamene u kodu.
- Predprocesorske direktive su linije koda koje počinju #, one su uputstva za transformaciju koda.
* Direktiva #define definiše tekstualnu supstituciju koja će se izvršiti u kodu. Podržava parametrizaciju.
* #define PI 3.14 - svuda u kodu gde stoji PI će se zamenti sa 3.14 (kao find and replace u Word-u).
* #define saberi(a,b) (a+b) => saberi(x, 3) će u predprocesoru postati x+3 => Ovo se naziva makro.

4. Uslovno kompajliranje.

- Predprocesor može proveriti uslov i u zavisnosti od njega neke linije koda propustiti ili izbaciti iz koda.

** Drugim rečima, neki delovi se mogu kompajlirati ili ne u zavisnoti od uslova.Ovo sve se radi pre prevođenja.*

- Relevantne direktive : #if, #elif, #else, #ifdef, #ifndef, #endif.

```
printf(“Jedan”);
#ifdef TEST              //Ako TEST nije definisan (#define TEST) neće ispisati dva.
printf(“Dva”);
#endif
printf(“Tri”);
```

5. Oblik main-a ako koristimo argumente sa komandne linije.

```
int main(int argc, char *argv[])
```

argc *– broj argumenata (argument count)*
argv *– niz stringova (vrednosti argumenata)*

```
int main(int argc, char *argv[])
{int i;
printf(“Imamo %d argumenata.\n”, argc);
for(i=0;i<argc;i++){printf(“Argument %d glasi: %s\n”, i, argv[i])};
return 0;}
```

04V - Fajlovi

1. Šta je file handle?

- U suštini dozvola koja nam daje pristup i ident. fajla. Dobijamo je samo ako smo uspeli da otvorimo fajl.

2. Osnovne funkcije za rad sa fajlom?

- **fopen** - prima dva argumenta, prvi je naziv/putanja fajla koji želimo otvoriti, drugi je mod.

* r - read, samo za čitanje; w - write, samo za pisanje; a - append, dodavanje na kraj; t - tekstualni mod; b - binarni mod

- Vraća FILE*, pokazivač na FILE strukturu ako je uspeo, NULL ako nije.
- Za čitanje i pisanje mogu se koristiti **fscanf** i **fprintf** koji su isti samo na početku se stavlja handle.
- Kada je rad sa fajlom završen, zatvara se funkcijom **fclose**, javljamo OS-u da nam više ne treba handle i tada fajl ponovo postaje dostupan za ostale programe.

```
FILE *ulazni, *izlazni;
ulazni = fopen("ime.txt", "rt"); izlazni = fopen("izlaz.txt", "wt");
fscanf(ulazni, ''%d%d", &a, &b);
fprintf(izlazni, "%d", a+b);
fclose(ulazni); fclose(izlazni);
```

3. Kako se fajl zatvara?

- fclose(naziv FILE promenjive);

4. Šta je feof?

- Funkcija koja nam može reći da li smo stigli do kraja fajla. Primena kod fajlova nepoznate veličine.

5. Fgets - koji su mu argumenti?

- **fgets**(string, max dužina, file handle);

05P - Projekti i zaglavlja

1. Objasni razliku između deklaracije i definicije.

- Kod f-ja : deklaracijom objavljujemo da ona postoji (sinonim za prototip), definicija je celo njeno pisanje.
- Kod promenjivih : deklaracija je extern (objavljujemo da postoji u nekom drugom fajlu unutar programa).

2. Objasni razliku između globalnih i lokalnih promenjivih.

- Globalne : vide se u čitavom programu i postoje kroz njegovo celo izvršavanje, napisane izvan svih f-ja.
- Lokalne : definisane unutar funkcija, životni vek im je vezan za njih.

3. Šta su zaglavlja?

- Posebna vrsta fajlova sa ekstenzijom .h, služe za zapis prototipa funkcija.

4. Objasni šta su include guards.

- Predprocesorske direktive koje sprečavaju procesiranje istog zaglavlja više puta tokom kompajliranja.

5. Objasni pojam static.

- Ključna reč koja ima dva značenja, u zavisnosti da li je zapisana uz globalnu ili lokalnu promenjivu.
- Globalna : promenjiva neće biti vidljiva nigde van fajla u kom se nalazi (čak ni uz extern).
- Lokalna : prom. će se ponašati kao globalna, njen životni vek će biti sličan globalnoj, živi kroz ceo prog.

05V - Funkcije

1. Šta su funkcije?

- Imenovani blok koda koji obavljaju jedan zadatak, možemo ih koristiti više puta tokom izvršavanja.

2. Iz čega se sastoji zaglavlje funkcije? *//Možda da ispisanu f-ju pa pita da li je ispravno napisana.*

- [tip] [naziv](paramteri){…kod…} - Ako je deklaracija onda samo do parametra sa ;.
- Parametri : ime, prezime ; Argumenti : Marko, Marković.
- Ako funkcija nije void, mora da ima return. Ako jeste, može a i ne mora.
- U main-u je pozivamo po nazivu i prosleđujemo joj argumente. => a = funkcija(x,y);
- Ako je funkcija int, onda mora i a (promenjiva u koju se smešta rezultat) da bude tipa int.
- Ukoliko je void, ne može da se dodeli. void f(); => a = f() je greška.
- Nizovi se ne kopiraju, ako prosledimo niz funkcija će dobiti original.

Primer zadatka: Zaokruži ispravno napisane funkcije.

* int f(); | float f(int c); | void funkcija (void f); | float saberi (double a,b,c); | f(char tekst[]); | void ispisi(int a[]);

06P - Debugger

1. Šta je debugger?

- Alat za uočavanje i uklanjanje grešaka. Alat koji pomaže u analizi programa.

2. Šta je breakpoint?

- Označena linija na kojoj debugger pauzira izvršavanje programa.

3. Šta je watch?

- Mogućnost posmatranja vrednosti iz promenjive dok program radi.

4. Koja je razlika između debug i release profila?

- Debug je namenjen programerima, sadrži dodatne informacije za debagovanje i veći je fajl.
- Release je režim za kreiranje konačne verzije programa, optimizovanog i bez pomoćnih informacija.

06V - Strukture i pokazivači

1. Šta je struktura?

- Kolekcija promenjivih; grupišemo više promenjivih u jedan tip.
- Objekat ili instanca je jedan član strukture (pr. struct Student a).

2. Kako pristupamo poljima objekta?

- Operatorom tačka (pr. Student.Ime).

3. Objasni pokazivač.

- Poseban tip podataka koji čuvaju neku memorijsku adresu tj. pokazuju na druge promenjive.
- Pravimo ih dodavanjem * između tipa i naziva promenjive.
- & = adresa | * = vrednost. | Uzimamo i dodeljujemo adresu a kao vrednost b, a potom preko *b menjamo a.
- b = &a ; *b = 2; => Vrednost polja a čija je adresa u b će biti 2.

07P - Memorija

1. Navedi vrste alokacije memorije i ukratko ih opiši.

- Statička alokacija = memorija poznata još za vreme kompajliranja (literali, globalne promenjive i sl.).
- Automatska alokacija = sve lokalne promenjive.
- Dinamička alokacija = sve što ručno alociramo.

2. Iz kojih blokova se sastoji memorija procesa?

- Heap = ostatak dostupne memorije u kojoj će se nalaziti dinamički alocirana memorija.
- Stack = memorija koja se koristi za vreme izvršavanja funkcija.
- Statički / globalni blok = svi statičko alocirani podaci (sadrži vrednosti poznate tokom kompajliranja).
- Programski kod = u njemu se nalazi program koji se izvršava tj. prevedene mašinske instrukcije.

3. Objasni funkcije malloc, calloc i free.

- Malloc i calloc su funkcije za dinamičku alokaciju memorije. Tražimo blok iz heap-a.
- Malloc zauzima veličinu u bajtima na heap-u i daje pokazivač na njega.
- Calloc zauzima veličinu * broj bajtova i inicijalizuje se sa svim popunjeno nulama.
- Free oslobađa memoriju tj. dati blok.

4. Koje su greške koje mogu nastati pri radu sa dinamičkom alokacijom?

- Upotreba neinicijalizovane memorije = korišćenje vred. preko pokazivača iz neinic. ili nealloc. bloka.
- Use-after-free = kada nastavimo koristiti memoriju koju smo oslobodili sa free().
- Memory leak = kada uzimamo mem. a ne oslobađamo je, sve veće i veće zauzeće mem. tokom rada programa.

5. Kako možemo saznati veličinu nekog tipa u bajtima?

- Funkcijom sizeof => sizeof(int).