RADNI PAKET (RP) 1

“TESTIRANJE SOFTVERSKIH PROGRAMA”

“RAZVOJ SOFTVERA”

Projekat „Umrežavanje mladih za ekonomsku razmenu u prekograničnom regionu“
Projekat br. CB007.2.22.078

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija

Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

SADRŽAJ

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tematski ciklus 1: "Testiranje softverskih programa"

1. Osnove testiranja: ciljevi, testiranje I otklanjanje grešaka, procena kvaliteta, nedostaci, uzroci i efekti

Testiranje je postupak ispitivanja softvera ili delova softvera u kontrolisanom okruženju i pod kontrolisanim okolnostima kako bi se otkrila odstupanja od zahteva (specifikacije zahteva) i potvrdilo da sistem ispunjava definisane kriterijume prihvatanja.

Testiranje softvera ima za cilj proizvodnju kvalitetnog softvera bez kvarova koji radi kako je očekivano. Testiranje je postupak kojim se procenjuje funkcionalnost sistema kako bi se utvrdilo da li ispunjava unapred definisane zahteve ili ne. Utvrđeni nedostaci se ispravljaju i ispituju sve dok se ne osigura da proizvod nema nedostataka. Može se testirati i celokupni rad sistema i pojedinačne funkcionalnosti.

Tokom testiranja identifikuju se greške, propusti ili nedostajući zahtevi u suprotnosti sa postavljenim zahtevima. Može se ručno ili uz pomoć automatizovanih alata. Neki više vole testiranje softvera, kao što je testiranje bele kutije i crne kutije.

2. Principi testiranja:

1) Testiranje pokazuje prisustvo nedostataka: Testiranje može pokazati prisustvo nedostataka / grešaka, ali ne može biti dokaz da nema nedostataka. Čak i nakon testiranja proizvoda, ne možemo 100% reći da nema nedostataka. Testiranje uvek smanjuje broj neotkrivenih grešaka koje su ostale u softveru, ali čak i ako nema nedostataka, to nije dokaz ispravnosti.

2) Potpuno testiranje nije moguće: Nemoguće je testirati sve, uključujući kombinacije uvoda i preduslova. Umesto da radimo sve ovo, možemo koristiti rizike i prioritete kako bismo se fokusirali na pokušaje testiranja. Na primer, aplikacija ima 15 polja za unos, svako sa po 5 mogućih vrednosti, da bismo testirali svaku moguću kombinaciju trebat će nam 30,517,578,125 testova (5 na 15. stepen). Malo je verovatno da će vam kompanija dati dovoljno vremena za pokretanje ovih testova. Zbog toga su postavljanje rizika i prioriteta neke od najvažnijih stvari u svakom projektu.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

3) Rano testiranje: U životnom ciklusu razvoja softvera (SDLC), testiranje treba započeti u najranijoj mogućoj fazi i treba se usredsrediti na definisane zadatke.

4) Grupisanje kvarova ili nakupljanje grešaka: Mali broj modula sadrži najveći procenat grešaka pronađenih u poslednjem krugu testiranja.

5) Paradoks pesticida: Ako se bilo koji test ponavlja iznova i iznova, na kraju će isti testovi pokazati isto i prestati da pronalaze druge nove greške. Da bismo prevazišli paradoks pesticida, treba često pregledati slučajeve ispitivanja i raditi nove i različite testove kako bi pronašli druge neotkrivene greške.

6) Testiranje zavisi od konteksta: Testiranje generalno zavisi od konteksta. Različite stvari se testiraju na različite greške, a kritični softver (onaj koji je povezan sa ljudskim životima) testira se na različite stvari od uobičajenog softvera i mora da pokrije neke norme

7) Obmana nedostatka nedostataka: Ako sam sistem / samu aplikaciju korisnici ne mogu da koriste za šta je namenjen ili ne ispunjava date uslove i zahteve korisnika, onda pronalaženje i otklanjanje nedostataka ne pomaže .

3. Test operacije i zadaci: planiranje testa, monitoring I kontrola testa, analiza testa, dizajn testa, izvršenje tasta, završetak testa;

Vidljivi deo testiranja je izvršenje testova. Međutim, da bi testovi bili efikasni i efikasni, treba uzeti u obzir potrebu za vremenom za planiranje testova, konstruisanje test slučajeva, pripremu za testove i procenu stanja softvera na osnovu rezultata ispitivanja.

Proces testiranja uključuje sledeće glavne operacije:

- Planiranje testa
- Kontrola
- Analiza i dizajn
- Izvršenje
- Postavljanje kriterijuma za popunjavanje testa i izveštavanje o rezultatima
- Operacije povezane sa završetkom testa

U nekim testovima ove operacije su definisane terminom osnovni procesi testiranja softvera. Iako su logično povezane, ove osnovne operacije mogu se izvoditi istovremeno, a ne nužno i sekvencijalno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Plan testiranja

Plan testiranja materijalizuje strategiju testiranja, opisuje resurse koji će se koristiti i okruženje, okruženje u kojem će se testiranje izvršiti, ograničenja koja će se primeniti i raspored izvođenja testova. Planovi testiranja su različiti za različite nivoe.

4. Struktura testa: projektovanje I prioritizacija test slučajeva i kompleta test slučajeva, identifikacija potrebnih test podataka kao potpora uslova testa i slučajeva testa , projektovanje uslova za test I identifikacija neophodne infrastructure i alata, hvatanje dvosmernog sleđenja između test baza

Test slučaj je niz radnji koje se izvršavaju da bi se utvrdila određena funkcija ili funkcionalnost vaše aplikacije.

Identifikovanje test slučajeva može potrajati dugo, a ponekad je potrebno ponoviti test. Stoga se mora dokumentovati. Sledeći elementi moraju biti zabeleženi za svaki test:

- Test slučaj treba da ima očekivani rezultat.
- Test primer može imati preduslove, kao što su određene vrednosti u bazi podataka, koje moraju biti prisutne unapred.
- Test slučaj takođe može sadržati post-uslove koji se primenjuju nakon završetka test slučaja.
- Tokom test slučaja dokumentirate rezultate koji su uočeni u koloni stvarnih rezultata.

Dizajn test slučajeva

Format testa za prijavu sadrži sledeći format:

- ID test slučaja
- Deo skripte za test
- Izvodi se test
-Test podaci
-Očekivani rezultati
-Stvarni rezultati

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

- rezultat testa (uspešan ili ne)

Najbolje prakse za dobar test

- Test slučajevi bi trebali biti jednostavni i transparentni
- Izvršite test sa krajnjim korisnikom
- Ne uzimajte radnu aplikaciju dok pripremate test slučaj. Pridržavajte se zahteva i projektne dokumentacije.
- Ispitivanje test slučaja mora ispunjavati zahteve
- Test slučaj bi trebalo da generiše iste rezultate svaki put, bez obzira na to ko sprovodi test.

5. Proizvodi za izvođenje testiranja: praćenja osnove testa proizvoda testiranja

Uslovi ispitivanja moraju biti povezani sa njihovim izvorima u bazi za ispitivanje, to je poznato kao sledljivost. Sledljivost može biti horizontalna u čitavoj test dokumentaciji za dati nivo ispitivanja (npr. Ispitivanje sistema, od uslova ispitivanja preko test slučajeva do test skripti) ili može biti vertikalna kroz slojeve razvojne dokumentacije (npr. od zahteva za komponentama).

Sledljivost se odnosi na dokument u testiranju softvera koji namerava da uspostavi vezu između različitih faktora. Pomaže u utvrđivanju potpunosti odnosa upoređivanjem osnovnih dokumenata (poslovnih i tehničkih). Sledljivost se proverava pripremom tabelarnog prikaza podataka (test slučajevi). Praćenje može da se kreće od zahteva za mapiranje do komponenti, uslova za skriptiranje testa ili test slučajeva.

Proizvodi za analizu testa uključuju definisane i prioritetne uslove ispitivanja, od kojih je svaki idealno dvosmeran za praćenje određenih elemenata testa na osnovu testa koji pokriva. Analiza testa takođe može dovesti do otkrivanja i prijavljivanja nedostataka u osnovi za ispitivanje.

Prednosti sledljivosti:

- Dobijamo jasnu sliku da li je softverski projekat razvijen u skladu sa postavljenim zahtevima.
- Da bi bili sigurni da su svi zahtevi uključeni u test slučajeve.
- Da bi se izbeglo dodavanje nepotrebnih ili neprikladnih karakteristika razvijenom projektu.
- Funkcije koje nedostaju mogu se lako identifikovati pomoću različitih vrsta praćenja.
- Ažuriranje test slučajeva može se lako izvršiti ako dođe do promena u zahtevima.
- Potvrđuje 100% pokrivenost testom.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

6. Modeli životnog ciklusa razvoja softvera

Poznavanje procesa razvoja softvera pruža osnovu za razumevanje uzroka softverskih grešaka i kako se one primenjuju u aplikacijama.

6.1. Životni ciklus softverskog sistema obično uključuje sledeće faze:

-Potrebne analize

Životni ciklus softvera započinje sastavljanjem niza zahteva za njega. Analiza zahteva se vrši kako bi se procenila tehnička izvodljivost, utvrdilo kako se deli softverski sistem, identifikovale specifikacije koje treba usaglasiti, utvrdilo za svaki pojedinačni deo na koji se sistem odnosi. Rezultat takve analize je specifikacija zahteva, koja se naziva i razvojni zadatak.

-Dizajn

Faza dizajna uključuje izgradnju pojedinih delova softverskog sistema. Aktivnosti u ovoj fazi uključuju dizajniranje tehničke arhitekture softvera, dizajniranje baze podataka, dizajniranje korisničkog interfejsa, odabir ili kreiranje novih algoritama, odnosno stvaranje tehničkog projekta.

- Programiranje

Aktivnosti u ovoj fazi uključuju stvaranje probnih podataka, stvaranje programskog koda (Source code), objektnog koda, radne dokumentacije, plana integracije i njegove primene.

- Testiranje

Faza testiranja ima za cilj otkrivanje i uklanjanje grešaka u sistemu. Otkriveni kvarovi u ovoj fazi se šalju na korekciju i ponovo testiraju. Ovaj postupak se ponavlja onoliko puta koliko je potrebno da bi se ispunili zahtevi za stepen grešaka koje ostaju neispravljene.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

- **Instalacija i eksperimentalni rad**

Jednom kada se softver testira, on se pruža kupcu. Plan ispitivanja obično sadrži raspored i norme za izvođenje testova u eksperimentalnom režimu rada i metod za prikupljanje, prijavljivanje i uklanjanje grešaka nakon puštanja u rad. Operacije u ovoj fazi su: planiranje instalacije, isporuka softvera, obuka korisnika, instalacija softvera.

- Rad i održavanje Redovnim radom se obavljaju pomoćne aktivnosti. To uključuje pružanje tehničke pomoći, održavanje režima za odgovaranje na zahteve podnosioca zahteva za pomoć

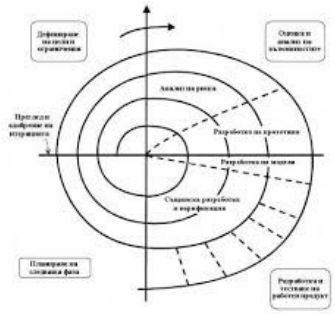
Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

6.1. Modeli životnih ciklusa

Kako su procesi razvoja softvera linearni, postoji nekoliko **teorijskih modela** koji odražavaju različite oblike toka životnog ciklusa softvera. Svaki od ovih modela ima svoje prednosti i nedostatke.

Model	Prednosti	Nedostaci
Waterfall (vodopad)	✓ Svaka faza mora biti završena da bi se prešlo na sledeću ✓ Potrebno je rano planiranje ✓ Testiranje je sastavni deo životnog ciklusa proizvoda ✓ Kvalitetno završavanje svake faze	- Zavisi od definicija i zahteva koji su uspostavljeni od najranijih faza ciklusa. - Zavisi od razdvajanja dizajnerskih zahteva - Povratne informacije su samo iz faze testiranja - Fokusiran je na ceo proizvod, a ne na pojedinačne procese -
Prototyping (prototip) 	✓ Zahtevi se mogu definisati ranije i pouzdanije ✓ Zahtevi se mogu istražiti brzo i po niskoj ceni ✓ Greške u zahtevima i dizajnu otkrivaju se u ranoj fazi	- Potrebni su alati za izradu prototipova i iskustvo sa njima, što je razvojni trošak - Prototip može postati proizvodni sistem

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007.
Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Spirala 	-Ponovna upotreba postojećeg softvera -Eliminiše greške -Uravnotežuje troškove resursa - Ciljevi kvaliteta formulisani su u procesu razvoja - Proizvod se može razvijati - Pruža pouzdan okvir sa integrisanim sistemom za razvoj hardvera i softvera	- To je povezano sa brzim razvojem aplikacija, što je u praksi veoma teško - Procesom je teško upravljati

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007.
Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

7. Nivoi testiranja; testiranje komponenata, testiranje integracije, testiranje sistema, testiranje prihvatljivosti

Najčešće korišćene metode ispitivanja su ispitivanje komponenata, ispitivanje integracije, ispitivanje prihvatljivosti i ispitivanje sistema. Softver prolazi kroz ove testove u određenom redosledu:

7.1. **Testiranje komponenata**

Prvo se izvodi jedan test. Kao što i samo ime govori, ovo je metoda testiranja na nivou objekta. Pojedinačne softverske komponente testiraju se na greške. Ovaj test zahteva tačno poznavanje programa i svakog instaliranog modula. Na ovaj način ovu verifikaciju vrše programeri, a ne testeri. U tu svrhu se kreiraju test kodovi koji proveravaju da li se softver ponaša kako je predviđeno.

7.2. **Integraciono testiranje**

Pojedinačni moduli koji su već testirani na jedinicama integrišu se jedni sa drugima i proveravaju se na kvarove. Ova vrsta testiranja uglavnom otkriva greške u interfejsu. Testiranje integracije može se izvršiti primenom pristupa odozgo nadole, prateći arhitektonski dizajn sistema. Drugi pristup je pristup odozdo prema gore, koji se izvodi sa dna kontrolnog toka.

7.3. **Testiranje sistema**

U ovom testu se čitav sistem proverava na greške. Ovaj test se izvodi povezivanjem hardverskih i softverskih komponenti čitavog sistema, nakon čega se proverava. Ovo testiranje je navedeno pod metodom testiranja „crna kutija“, gde se proveravaju očekivani uslovi korisnika softvera.

7.4. **Testovi prihvatanja**

Ovo je poslednji test koji se izvodi pre predaje softvera kupcu. Sprovodi se kako bi se osiguralo da softver koji je razvijen ispunjava sve zahteve kupaca. Postoje dve vrste testova prihvatljivosti - jedan koji sprovode članovi razvojnog tima, poznat kao interno ispitivanje prihvatljivosti (Alfa testiranje), a drugi koji sprovodi kupac, poznat kao eksterno ispitivanje prihvatanja. Ako se testiranje vrši uz pomoć potencijalnih kupaca, to se naziva testiranje prihvatanja kupaca. Ako testiranje vrši krajnji korisnik softvera, ono je poznato kao testiranje prihvatanja (beta testiranje).

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

8. Vrste testova. Funkcionalno testiranje, nefunkcionalno testiranje, testiranje u beloj kutiji

8.1. **Funkcionalno testiranje**

Funkcionalno testiranje proverava ponašanje sistema unošenjem ulaznih i izlaznih podataka. Koristi se metoda ispitivanja „Crna kutija“. Testovi se kreiraju na osnovu funkcionalnih zahteva. Funkcionalni zahtevi određuju ponašanje sistema. Oni definišu njegova ograničenja. Funkcionalni zahtevi moraju biti dokumentovani u: - sistemu upravljanja zahtevima; - specifikacije softverskih zahteva.(Software Requirements Specifications).Zahtevi se koriste kao osnova za ispitivanje. Za svaki zahtev se kreira najmanje jedan test. Za ispunjavanje zahteva potrebno je više testova. Ispitivanje zasnovano na zahtevima uglavnom se koristi u ispitivanju sistema i ispitivanju prihvatanja.

8.2. **Nefunkcionalni testovi**

Sigurnost aplikacija je jedan od glavnih zadataka programera. Test bezbednosti proverava softver na poverljivost, integritet, potvrdu identiteta, dostupnost i trajanje. Pojedinačni testovi se izvode kako bi se sprečio neovlašćeni pristup programskom kodu.

Stres test je metoda u kojoj je softver izložen uslovima koji prelaze uobičajene radne uslove softvera. Nakon dostizanja kritične tačke, rezultati se beleže. Ovaj test određuje stabilnost celog sistema. Proverava se kompatibilnost softvera sa spoljnim interfejsima kao što su operativni sistemi, hardverske platforme, veb pregledači itd. Test kompatibilnosti proverava da li je proizvod kompatibilan sa bilo kojom softverskom platformom.

Kao što i samo ime govori, ova tehnika testiranja proverava količinu koda ili resursa koje program koristi za dovršavanje radnje.

Ovim testom se proverava upotrebljivost softvera za korisnike. Lakoća s kojom korisnik može pristupiti uređaju glavna je stvar testiranja. Testiranje upotrebljivosti obuhvata pet aspekata testiranja - osposobljenost, efikasnost, zadovoljstvo, pamćenje i greške.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

8.3. **Testiranje u beloj kutiji**

Testiranje u beloj kutiji, za razliku od crne, uzima u obzir unutrašnje funkcionisanje i logiku koda. Da bi izvršio ovaj test, ispitivač mora da poznaje kod da bi odredio tačan deo koda koji ima greške. Ovaj test je poznat i kao testiranje bele kutije, otvorene kutije ili staklene kutije.

Testiranje u beloj kutiji je detaljno proučavanje logike i strukture koda. Za njegovu primenu potrebno je znanje rada koda. Provjerava se programski kod i provjerava koja jedinica ne radi ispravno. Metoda je pogodna za testiranje aplikacija koje pružaju veb usluge i nije praktična za otklanjanje grešaka u velikim sistemima i mrežama. Smatra se bezbednosnim testom, tj. Postupkom utvrđivanja da li informacioni sistem štiti podatke i održava funkcionalnost. Metoda se može koristiti
da potvrdi primenu koda, tj. da li je u skladu sa dizajnom, da proveri primenjenu sigurnosnu funkcionalnost i otkrije ranjivosti koje se mogu iskoristiti.

9. Rizici i testiranje

Rizik je mogućnost negativnog ili neželjenog rezultata ili događaja. Svaki nastali problem smanjuje percepciju kvaliteta proizvoda ili uspeha projekta. „Rizici su neizvesni događaji koji će se verovatno dogoditi u budućnosti i koji će verovatno rezultirati gubicima.“ Identifikacija i upravljanje rizikom su osnovna briga svakog projekta. Efektivna analiza softverskog rizika pomaže u efikasnom planiranju i dodeljivanju radnih zadataka.

Rizik se uglavnom izražava u dve vrste:

- rizik od proizvoda (kvaliteta) - primarni efekat potencijalnog problema na kvalitet proizvoda;

- Projektni rizik (planiranje) - primarni efekat je na uspeh projekta.

Rizici po važnosti nisu jednaki. Određivanje nivoa rizika zavisi od nekoliko faktora (Tabela 1.1 - Određivanje rizika prema verovatnoći pojave i uticaja na sistem):

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

- verovatnoća pojave - proizlazi iz tehničkih razloga, kao što su programski jezici koji se koriste i Internet veza;

- uticaj problema u slučaju pojave - proizilazi iz poslovnih razloga, kao što su finansijski gubici i broj pogođenih korisnika. Napor se raspoređuje srazmerno nivou rizika, odnosno najpre se sprovode važniji testovi.

Za rizike proizvoda moramo uzeti u obzir:

- koje funkcije i atributi su presudni za uspeh proizvoda;

- koliko je problem vidljiv kupcima ili potrošačima;

- koliko često se funkcija koristi; - Da li je moguće preskočiti ovu funkcionalnost?

Kategorije rizika:

Rizik po rasporedu: Vremenski raspored projekta može se odložiti kada se tačno adresiraju projektni zadaci i rok za isporuku proizvoda. Rizici po rasporedu mogu uticati na projekat, a na kraju uštede koje kompanija napravi mogu dovesti do neuspeha projekta. Rasporedi često ne uspevaju iz sledećih razloga:

• Loša procena vremena.

• Nemogućnost pravilne procene funkcionalnosti i vremena potrebnog za razvoj ovih funkcionalnosti.

• Korišćenje resursa se ne nadgleda pravilno. Svi resursi poput osoblja, sistema, individualnih veština.

• Neočekivana proširenja obima projekta.

Operativni rizici: Rizici uzrokovani nedostatkom pravilne implementacije, sistemskim kvarovima ili nekim spoljnim događajima rizika.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Uzroci operativnih rizika:

• Nemogućnost pravilnog rešavanja sukoba u određivanju prioriteta

• Nepodmirivanje odgovornosti

• Nedovoljno resursa

• Nedovoljna obuka u timu

• Neplaniranje resursa

• Nedostatak komunikacije u timu

Tehnički rizici: Tehnički rizici obično dovode do kvarova u funkcionalnosti i performansama.

Uzroci tehničkih rizika su:

• Stalna promena zahteva.

• Korišćenje zastarelih tehnologija ili tehnologija u ranoj fazi.

• Složenost implementacije projekta

• Kompleksna modularna integracija projekata.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pragmatični rizici: To su spoljni rizici izvan operativnih granica. Sve su to nejasni rizici izvan opsega programa.

Ovi spoljni događaji mogu biti:

• Iscrpljivanje sredstava

• Razvoj tržišta

• Promena strategije proizvoda i prioriteta

• Promena državnih propisa. Budžetski rizici:

• Loša procena budžeta

• Prekomerna potrošnja

• Proširivanje obima projekta

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Thematski ciklus 2: "Razvoj softvera"

I. Osnove programiranja:

Šta znači „programiranje"?

Programiranje znači davanje računara naredbi šta treba da radi, kao što je „reprodukcija zvuka", „ispisati nešto na ekranu" ili „pomnožiti dva broja". Kada postoji nekoliko komandi u nizu, one se nazivaju računarskim programom. Tekst računarskih programa naziva se programski kod (ili izvorni kod ili skraćeni kod).

Kompjuterski programi

Računarski programi su niz naredbi napisanih na unapred odabranom programskom jeziku, kao što su C ++, Java, JavaScript, Pithon, Rubi, PHP, C, C #, Svift, Go ili neki drugi. Da bismo pisali komande, moramo znati sintaksu i semantiku jezika sa kojim ćemo raditi, u našem slučaju C ++. Stoga ćemo se upoznati sa sintaksom i semantikom jezika C ++, kao i sa programiranjem uopšte, u trenutnoj knjizi, učeći korak po korak pisanje koda od jednostavnije ka složenijim programskim strukturama.

Algoritmi

Računarski programi obično izvršavaju neki algoritam. Algoritmi su niz koraka, neophodnih za izvršavanje određenog zadatka i za dobijanje nekog očekivanog rezultata, nešto poput „recepta". Na primer, ako pržimo jaja, sledimo neki recept (algoritam): zagrejemo ulje u tiganju, razbijemo jaja u njemu, sačekamo da se prže i odmaknemo ih od šporeta. zagrejemo ulje u šerpi, slomimo jaja u njemu, sačekamo da se prže i odmaknemo ih od šporeta. Slično tome, u programiranju računarski programi izvršavaju algoritme: niz naredbi, neophodnih za izvršavanje određenog zadatka. Na primer, da bi se niz brojeva rasporedio u rastućem redosledu, potreban je algoritam, na primer da se pronađe najmanji broj i odštampa, da se od ostalih brojeva ponovo pronađe najmanji broj i odštampa, i to se ponavlja, dok se brojevi ne iscrpe.

Za praktičnost prilikom kreiranja programa, za pisanje programskog koda (komande), za izvršavanje programa i druge operacije povezane sa programiranjem, potrebno nam je razvojno okruženje, na primer Visual Studio

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Programski jezici, kompajleri, tumači i razvojno okruženje

Programski jezik je veštački jezik (sintaksa za izražavanje), namenjen za davanje naredbi koje želimo da računar čita, obrađuje i izvršava. Koristeći programske jezike, pišemo sekvence naredbi (programa), koje definišu šta računar treba da radi. Izvršenje računarskih programa može se izvršiti pomoću kompajlera ili pomoću tumača.

Kompajler prevodi kod iz programskog jezika u mašinski kod, jer za svaku od struktura (naredbi) u kodu bira odgovarajući, prethodno pripremljeni fragment mašinskog koda i u međuvremenu proverava tekst programa na greške. Sastavljeni fragmenti zajedno čine program u mašinski kod, kako to očekuje mikroprocesor računara. Jednom kad je program sastavljen, može se izvršiti direktno iz mikroprocesora u saradnji sa operativnim sistemom. Sa programskim jezicima zasnovanim na kompajleru, kompajliranje programa se vrši obavezno pre njegovog izvođenja, a sintaksne greške (pogrešne naredbe) se pronalaze tokom vremena kompajliranja. Jezici poput C ++, C #, Java, Svift i Go rade sa kompajlerom.

Neki programski jezici ne koriste kompajler i tumače ih direktno specijalizovani softver koji se naziva „interpreter“. Tumač je „program za izvršavanje programa“, napisan na nekom programskom jeziku. Naredbe u programu izvršava jednu za drugom, jer razume ne samo jednu komandu i nizove naredbi, već i druge jezičke konstrukcije (procene, iteracije, funkcije itd.). Jezici poput Pithon, PHP i JavaScript rade sa interpretatorom i izvršavaju se bez kompajliranja. Zbog odsustva prethodne kompilacije, u interpretiranim jezicima greške se pronalaze tokom vremena izvršavanja, nakon što program počne da se pokreće, a ne ranije.

Programsko okruženje (Integrisano razvojno okruženje - IDE) kombinacija je tradicionalnih alata za razvoj softverskih aplikacija. U razvojnom okruženju pišemo kod, kompajliramo i izvršavamo programe. Razvojna okruženja
integrišu u njih uređivač teksta za pisanje koda, programski jezik, kompajler ili tumač i runtime okruženje za izvršavanje programa, program za pronalaženje grešaka za praćenje programa i traženje grešaka, alate za dizajn korisničkog interfejsa i druge alate i dodatke.

Programska okruženja su pogodna, jer integrišu sve što je potrebno za razvoj programa, bez potrebe za napuštanjem okruženja. Ako ne koristimo razvojno okruženje, moraćemo da napišemo kod u uređivač teksta, da ga kompajliramo komandom na konzoli, da ga pokrenemo drugom komandom na konzoli i da po potrebi napišemo još dodatnih komandi, što je veoma dugotrajno. Zbog toga većina programera koristi IDE u svom svakodnevnom radu.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Za programiranje na jeziku C ++, najčešće korišteno razvojno okruženje je Visual Studio, koji Microsoft razvija i distribuira slobodno i može se preuzeti sa: https://vvv.visualstudio.com/dovnloads/. Alternative Visual Studija su Rider (https://vvv.jetbrains.com/rider/), MonoDevelop / Ksamarin Studio (http://vvv.monodevelop.com), SharpDevelop (http://vvv.icsharpcode.net/OpenSo urce / SD /) i Eclipse (https://vvv.eclipse.org/dovnloads/packages). Još jedno uobičajeno razvojno okruženje je Code :: Blocks (http://vvv.codeblocks.org/dovnloads)

Promenljive i izrazi

U programiranju promenljiva je mesto za čuvanje neke vrednosti u RAM-u računara. Programer ga označava imenom (identifikatorom), obično na latinici. Budući da vrednost sa informacijama u računaru može biti datog tipa (ceo broj, razlomak, slovo, struktura, objekat itd.), Kaže se da je promenljiva koja je sadrži iz odgovarajućeg tipa: celobrojna promenljiva, simbolična, objekat, itd.

Ime promenljive označava određeno područje u memoriji u kome je sadržana njegova vrednost. Ovo razdvajanje imena i sadržaja omogućava da se ime koristi bez obzira na informacije koje predstavlja. Promenljive se mogu pohraniti direktno u radnu memoriju programa (stek) ili u dinamičku memoriju, u kojoj se čuvaju veći objekti (nizovi znakova i nizovi). Primitivni tipovi podataka (brojevi, char, bool) nazivaju se tipovima vrednosti jer svoju vrednost čuvaju direktno u programskom steku. Referentni tipovi podataka (nizovi, objekti i nizovi) čuvaju adresu u dinamičkoj memoriji gde se čuva njihova stvarna vrednost. Oni se mogu odvojiti i dinamički otpustiti, tj. Njihova veličina nije unapred fiksna, kao kod tipova vrednosti. Identifikator u kodu se može postaviti dok je program pokrenut. Međutim, tokom njenog izvršavanja, ova vrednost se može promeniti. Promenljive se mogu koristiti u različitim procesima: da se vrednost postavi na jedno mesto, pa na drugo, zatim mogu da prihvate novu vrednost i koriste je na istim mestima na kojima je ranije korišćena. Vrednost promenljive može se menjati tokom izvršavanja programa, sve dok njegovo ime, tip i lokacija memorije ostaju nepromenjeni.

Promenljive u programiranju najčešće imaju opisna imena, u zavisnosti od toga šta sadrže i kako se mogu koristiti, za razliku od promenljivih u matematici, koje se obično sastoje od 1-2 znaka. Kada želimo da kompajler dodeli područje u memoriji za neke informacije koje se koriste u našem programu, moramo to imenovati. Dozvoljeni simboli su slova a-z, A-Z, brojevi 0 - 9, kao i simbol '_'. Takođe moramo uzeti u obzir da programski jezici imaju službene reči i da ih ne možemo koristiti. Naziv služi kao identifikator i omogućava nam da se uputimo na potrebno područje memorije. Kada proglasimo promenljivu, izvršavamo sledeće radnje:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

- podesite njegov tip (na primer int);

- podesite njegovo ime (identifikator, na primer starost);

- možemo postaviti početnu vrednost (na primer 25), ali to nije obavezno. Sintaksa za deklarisanje promenljivih u C # i Javi je sledeća:

Evo primera deklarisanja promenljivih:

Dodeljivanje vrednosti promenljivoj je postavljanje vrednosti koja će joj biti zapisana. Ova operacija se izvodi putem operatora dodele '='. Ime promenljive prikazano je na levoj strani operatora, a nova vrednost prikazana je na desnoj strani. Inicijalizacija promenljive znači prvo podešavanje vrednosti.

Veliki deo rada programa je izračunavanje izraza. Izrazi su nizovi operatora, literala i promenljivih koji se izračunavaju na određenu vrednost nekog tipa (broj, niz, objekat ili drugi tip).

Izračunavanje izraza takođe može imati neželjene efekte, jer izraz može sadržati ugrađene operatore dodeljivanja, povećanje ili smanjenje vrednosti (priraštaj, umanjenje) i metode pozivanja.

Uslovni kod

U računarskim naukama uslovne strukture su funkcije programskog jezika, pomoću kojih možemo izvršiti različite radnje u zavisnosti od nekog stanja.

U imperativnim programskim jezicima obično se koristi termin „uslovne strukture“, dok se u funkcionalnom programiranju preferiraju izrazi „uslovni izraz“ ili „uslovna struktura“, jer ovi pojmovi imaju različita značenja.

Bulovi izrazi

Skup vrednosti tipa Bool (bool u C / C ++ / C #, boolean u Javi) sastoji se od dva elementa - vrednosti true i false. Te vrednosti se nazivaju i logičke konstante.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Logičkim izrazom u računarskoj nauci nazivamo bilo koji izraz koji posle svog transformacija, može se svesti na logičku vrednost
konstantan.

x	y	! x	x && y	x || y	x ^ y
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Operatori upoređivanja

U jezicima visokog nivoa razlikujemo nekoliko vrsta operatora upoređivanja koji se koriste za upoređivanje parova celih brojeva, nizova i drugih tipova podataka. Svi izrazi koji sadrže operatore upoređivanja su logički izrazi jer se njihov rezultat svodi na logičku konstantu.

Operator	Postupak
==	jednakost
!=	Različito
>	Veće od
<	Manje od
> =	Veće ili jednako
<=	Manje ili jednako

Logički operatori

Logički operatori prihvataju Bulove vrednosti i vraćaju Bulov rezultat.

Logički operator	Stilizovani zapis	naziv
&&	AND	AND
||	OR	OR
!	NOT	Logical negation
^	XOR	Exclusive OR

Vrste uslovnih struktura

U jezicima porodice C razlikujemo sledeće tipove uslovnih struktura:

if / else / else if / embedded if strukture

Program se može ponašati različito, u zavisnosti od uslova, koristeći uslovne strukture if i if-else. Oni su vrsta uslovne kontrole koja se proverava tokom izvođenja konstrukcije.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

IF (ako)

Format uslovne strukture if je sledeći: klauzula if, logički izraz i telo uslovne strukture;

```
if (logički izraz)
{
telo uslovne strukture;
}
```

Logički izrazi mogu biti varijable tipa Boolean ili logički logički izrazi.
U C # ne mogu biti celi brojevi za razliku od drugih programskih jezika kao što su C i C ++.

Telo strukture je zaključano između velikih kovrdžavih zagrada „{}“ i može se sastojati od jedne ili više operacija (naredbi).

Ako se izraz u zagradama iza ključne reči if izračuna na tačno, izvršava se telo uslovne strukture. Ako je rezultat izračunavanja logičkog izraza netačan, tada se operateri u telu neće izvršiti.

ELSE (inače)

Postoji i uslovna struktura sa klauzulom else (if-else). Njegov format uključuje: rezervisanu reč if, logički izraz, telo uslovne strukture, rezervisanu reč else i telo else-strukture, koje se mogu sastojati od jednog ili više operatora:

```
if (logički izraz)
{
telo uslovne strukture;
}
inače
{
telo druge strukture;
}
```

Ovde ova struktura deluje na sledeći način: u zavisnosti od rezultata izraza u zagradama (Bulov izraz), moguća su dva puta za nastavak toka proračuna. Ako je logički izraz tačan, izvršava se telo uslovne strukture, a ostalo je izostavljeno jer se operateri u njemu ne izvršavaju. Inače se izvršava struktura else, ali glavno telo je izostavljeno i operatori u njemu se ne izvršavaju.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

else if (inače ako)

Pored if i else, koji mogu opisati ukupno 2 slučaja, možemo imati i else if strukture koje proveravaju nekoliko uslova. Obavezno je da se else if struktura koristi nakon if strukture, a nakon else if može postojati jedna else struktura, ali ovo nije obavezno.

```
if (x == 0)
    {
 Console.WriteLine("The number is 0");
}
else if (x == 1)
{
 Console.WriteLine("The number is 1");
}
else if (x == 2)
{
 Console.WriteLine("The number is 2");
}
else
{
 Console.WriteLine("Another number");
}
```

Kao što već znamo, ako proverava neki uslov, ako je taj uslov ispunjen, idemo na izvršenje koda u ovoj strukturi if i zanemarite kod svih ostalih struktura if i opcionalne else. Ako uslov u ako nije ispunjen, proveravamo međusobne uslove po redosledu kojim su napisani. Kada se pronađe podudaranje, izvršava se odgovarajući kod, a ostali se zanemaruju. Ako postoji struktura else i nismo imali podudaranje u if ili inače ako, tada se izvršava kod u strukturi else, koji pokriva sve slučajeve koji nisu obuhvaćeni prethodnim strukturama.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Ugrađene strukture ako (if)

Ugrađene strukture if ili if-else često pronalaze primenu u programskoj logici u programu ili aplikaciji. To se postiže postavljanjem strukture if ili if-else u telo druge strukture if ili else. Ovde se svaka druga klauzula odnosi na najbližu prethodnu if klauzulu. Na ovaj način shvatamo na koju se još klauzulu odnosi klauzula.

Ne smeju se ugrađivati više od tri uslovne strukture, u suprotnom se deo koda mora izvesti zasebnom metodom.

“preklopno kućište”

Prekidač uslovne strukture (višestruko grananje u nekim programskim jezicima) koristi se za biranje sa liste opcija. Struktura upoređuje datu vrednost sa određenim konstantama i na osnovu poređenja sa bilo kojom od njih preduzima radnju.

```
switch (selector)
{
 case value-1: execution code; break;
 case value-2: execution code; break;
 case value-3: execution code; break;
 case value-4: execution code; break;
```

```
// ...
default: execution code; break;
}
```

Struktura prekidača bira između delova programskog koda na osnovu izračunate vrednosti određenog izraza. Ovaj izraz je obično ceo broj, ali može biti i tipa string ili char. Vrednost birača mora se izračunati pre upoređivanja sa vrednostima unutar strukture prekidača. Oznake (slučaj) ne moraju imati istu vrednost. Kada se selektor podudara sa bilo kojom od vrednosti slučaja, struktura preklopnog slučaja izvršava kod nakon odgovarajućeg slučaja. U nedostatku podudaranja, izvršava se podrazumevana struktura, kada takva postoji. Svaka oznaka slučaja, kao i podrazumevana oznaka, moraju se završiti

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

ključnom reči break, koja završava strukturu prekidača nakon pronalaska podudaranja i izvršavanja odgovarajućeg koda.

```
int number = 6;
switch (number)
{
 case 1:
 case 4:
 case 6:
 case 8:
 case 10:
 Console.WriteLine("This is not a
 prime number!"); break; case 2:
 case 3:
 case 5:
 case 7:
 Console.WriteLine("This is a
 prime number!"); break;
 default:
 Console.WriteLine("I don't know what that number is!"); break;
}
```

U C Sharpu imamo mogućnost da koristimo više tagova kada je potrebno da izvrše isti kod. Na ovaj način pisanja, kada pronađemo podudaranje, jer nakon odgovarajuće oznake slučaja nema izvršnog koda i operatora prekida, izvršiće se sledeći naišli kod. Ako takav nedostaje, izvršiće se podrazumevana struktura.

Dobre prakse u upotrebi "prekidača"

• Dobra je praksa uvek koristiti podrazumevanu strukturu koja se postavlja na kraj, nakon ostalih oznaka velikih i malih slova, da bi se obrađivale netipične vrednosti koje birač može prihvatiti ili čak situacije koje se mogu smatrati netačnim.
• Dobro je na prvo mesto staviti one predmete koji se bave najčešćim situacijama, a na kraju napustiti strukture predmeta koji se bave ređim situacijama.
• Ako su vrednosti u oznakama slučaja celi brojevi, preporučuje se da budu raspoređene u rastućem redosledu.
• Ako su vrednosti u oznakama velikih i malih slova tipa znakova, preporučuje se da oznake velikih i malih slova budu raspoređene po abecedi.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Trostruki operater?

Uslovni operator? : je operator na C jeziku i C-sličnim jezicima. Poznat je i kao ternarni operator, jer jedini operater prihvata 3 operanda.

```
operand1 ? operand2 : operand3
```

Prvi operand ili uslov uslovne strukture može biti logička promenljiva ili logički izraz i može prihvatiti i logičke vrednosti tačno i netačno. Ako se nakon izvršavanja potrebnih transformacija operand1 svede na tačan izraz, tada će nakon njegovog izvođenja ternarni operater vratiti vrednost operand2, u suprotnom (false), vraćena vrednost biće vrednost operanda3.

```
int a = 5;
int b = 3;
int larger = (a > b) ? a : b;
```

U gornjem primeru inicijalizujemo 2 celobrojne promenljive a i b i dajemo im vrednosti 5 i 3. Za promenljivu veću rezultat dodeljujemo pomoću ternarnog operatora. U ovom slučaju je 5, pošto je uslov (a> b) ispunjen (tačno), vraćena vrednost biće vrednost operanda pre dve tačke, odnosno promenljiva a, koja je 5. Ovaj primer pokazuje kako možemo lako odrediti koji je od dva broja veći bez upotrebe strukture if-else. Isti primer implementiran sa if-else izgledao bi ovako:

```
int a = 5;
int b = 3;
int larger;
if (a > b)
larger = a;
else
 larger = b;
```

Petlje i ponavljanje

Postupak cikličnog izračunavanja je proces, ukratko nazvan petlja, što je ponovljeno izvršavanje niza operacija sa različitim podacima. Najčešće se menja samo jedna vrednost koja se naziva parametar petlje. Za različite vrste petlji, programski kod je
ponavlja se sve dok ne stupe na snagu unapred utvrđeni uslovi ili fiksni broj puta, koji su pomenuti na početku.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Svaki ciklični proces karakterišu sledeći elementi:

• Inicijalizacija - postavlja početnu vrednost parametra petlje.
• Telo petlje - inicijalizuje kod koji se mora izvršiti određeni broj puta.
• Ažuriranje - vrednost parametra petlje se ažurira.
• Uslov prekida - izraz, u zavisnosti od toga koju vrednost petlja zaustavlja ili nastavlja svoje delovanje.
Izostavljanje ili netačno podešavanje bilo kog elementa petlje može dovesti do greške u njenom izvršavanju ili nemogućnosti izvršenja.

Za petlju

Naziv za potiče od engleske reči za - prevedeno na bugarski - „za“, jer se tokom njenog izvršavanja telo petlje izvršava za promenljivu u samoj petlji. Reč za koristi se kao ključna reč u većini programskih jezika.

U računarstvu su petlje for blokovi programskog koda koji se mogu izvršiti ponavljanjem operacija unetih u njih.

Za razliku od ostalih vrsta petlji, u strukturu for petlje uveden je brojač kroz koji se može kontrolisati broj iteracija. Jer petlje su primenljive kada je unapred određeno koliko iteracija program treba da izvede.

Struktura

Struktura petlji uključuje:

a) blok inicijalizacije

b) stanje petlje

v) naredbe za ažuriranje vodećih promenljivih

g) Telo petlje

Primer predstavljanja strukture for petlje u jeziku C #:

Telo petlje

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Telo petlje sadrži stvarni deo programa - blok sa izvornim kodom (koji se takođe naziva izvorni kod ili programski kod). U njemu su dostupne promenljive deklarisane u bloku inicijalizacije petlje. Ovaj kod se izvršava na svakoj sledećoj iteraciji (ponavljanju) dok se ne ispuni uslov za prekid petlje i ne izađe iz petlje

Primeri
Primeri primene petlje u programskom jeziku C #:

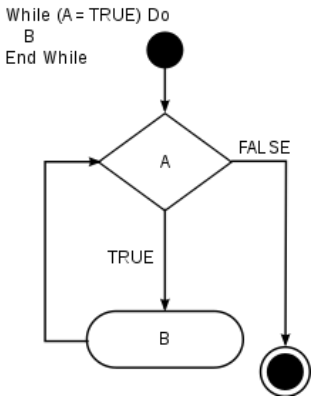
```
for (initialization-„a”; condition-„b”; update on leading variable-”c”)
{
 Body of the loop - “d”
}
```

U ovom primeru se izvodi 10 iteracija u for petlji. Pre svakog od njih proverava se da li je uslov za prekid petlje - vrednost promenljive i, veći od
20. Dok se ne ispuni ovaj uslov - promenljiva i ima vrednost manju ili jednaku 20, štampaju se svi parni brojevi koji su veći od 1 i manji ili jednaki 20.

Petlja while

Petlja while je petlja koja se izvršava sve dok je unapred definisani logički uslov na početku petlje tačan.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.



Struktura while petlje

Petlja radi na sledeći način:

1. Uslov se proverava i ako je tačno nastavlja se na kod u telu petlje. U suprotnom, petlja se završava.
2. Izvodi se kod u telu petlje.
3. Uslov petlje se ponovo proverava itd.

Telo petlje mora biti zatvoreno u vitičaste zagrade ({telo petlje}), a uslov u uglastim zagradama (uslov).

Primer vhile petlje u C #

```
int i = 0;
 while (i<10) //condition
 {
 Console.Write(i); //body of the loop
 i++;
 }
// result 0123456789
```

Postavljena je promenljiva i koja je jednaka 0. Uslov petlje je da radi sve dok je i manje od 10. Sa svakom rotacijom petlje i se povećava za 1. U suprotnom, petlja bi bila beskonačna i na njoj bi se ispisalo 0 svaki put na ekranu. Pri poslednjem okretanju petlje štampa 9 na konzoli i povećava vrednost i za jedan. Sada je i jednako 10. Kada proveravate stanje na početku petlje, već je netačno, jer i nije manje od 10, petlja prekida svoj rad.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Do-while petlja

Do-while struktura slična je while petlji, ali s tom razlikom što je uslov postavljen na kraju, tj. Provera se vrši nakon završetka tela petlje.

Primer radne petlje u C #

```
int i = 0;
 do
            {
             Console.Write(i); //body of the loop
i++;
            }
 while (i < 10) //condition
```

Ugrađene petlje

Ako je nova instrukcija petlje uključena u telo petlje, dobija se složenija struktura koja se naziva „petlja u petlji“ ili ugrađene petlje. U ovom slučaju, petlja koja se nalazi u prvoj naziva se internom, a prva petlja spoljnom. Svaku od dve petlje karakterišu četiri glavna elementa - inicijalizacija, telo, stanje prekida i ažuriranje. Obavezno je da su parametri dve petlje različite promenljive, tj. Da se imenuju drugačije. U svakom izvršenju radnji u spoljnoj petlji vrši se puni broj izvršavanja radnji u unutrašnjoj petlji.

Primer ugrađenih petlji u C#

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
for (int i = 0; i < 3; i++)
{
 for (int j = 0; j < 5; j++)
 {
 Console.Write(j);
 }
 Console.WriteLine();
}
// result
// 01234
// 01234
// 01234
```

Beskonačna petlja

Petlja koja se nikad ne završava naziva se beskonačna petlja. Beskonačna petlja je niz komandi u računarskom programu, koji izvodi beskonačan broj iteracija. Beskonačne petlje mogu se pojaviti u for, while, do-while petljama. Razlozi za njihov nastanak mogu biti:

• Zbog nedostatka uslova za prekid programa.
• Uveden je uslov za prekid petlje, koji nikada ne može biti ispunjen.
• Uveden je uslov za prekid petlje, nakon čijeg izvršenja petlja počinje da radi od svoje početne tačke.

2. Algoritmi, elementi of C/C ++ programskih jezika, osnovni tipovi podataka;

C ++ je programski jezik opšte namene. Stvorio ga je Bjarne Stroustrup, a rad na projektu započeo je 1979. godine, a prva javna verzija jezika objavljena je 1985. Naziv C ++ povezan je sa principima oko kojih je Stroustrup gradio svoj jezik. Kao osnovu uzeo je jezik C i pokušao da mu doda koncept nastave. Tako je rođen projekat „C sa časovima“. Usledila je duga evolucija, da bi na kraju nastao jezik C ++.

Glavna karakteristika koja ga razlikuje od prethodnika je mogućnost objektno orijentisanog programiranja (OOP). OOP je paradigma u kojoj je programski kod podeljen na klase, objekte i metode koji skladište potrebne informacije i međusobno komuniciraju, ako je potrebno. Ovaj koncept programiranja je suprotan konceptu proceduralnog programiranja, u kojem se program izvršava sekvencijalno instrukcijama (računskim koracima). OOP uvodi apstraktniji pristup programskom kodu, što omogućava programerima da se više usredsrede na programsku logiku.

Međutim, na apstraktnijem nivou, C ++ zadržava dovoljno svojstava koja su poznata jezicima nižeg nivoa. Jedan od njih je upravljanje memorijom. Ovo je ključni element u programiranju na C ++ jer, za razliku od jezika višeg nivoa (gde se ova aktivnost obavlja

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

automatski). Ova funkcija (za ručno upravljanje memorijom) čini C ++ jezikom pogodnim za programe koji će se izvoditi u hardverskom okruženju sa ograničenijim resursima

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tipovi podataka

Promenljiva u C ++ mora da navede tip podataka:

Primer

```
int myNum = 5; // Integer (whole number)
float myFloatNum = 5.99; // Floating point number
double myDoubleNum = 9.98; // Floating point number
char myLetter = 'D'; // Character
bool myBoolean = true; // Boolean
string myText = "Hello"; // String
```

Osnovni tipovi podataka

Tip podataka određuje veličinu i vrstu informacija koje će promenljiva čuvati:

Tip podatka	veličina	opis
int	4 bytes	Čuva čitave brojeve, bez decimala
float	4 bytes	Pohranjuje razlomljene brojeve koji sadrže jedan ili više decimala. Dovoljno za čuvanje 7 decimalnih cifara
double	8 bytes	Pohranjuje razlomljene brojeve koji sadrže jedan ili više decimala. Dovoljno za čuvanje 15 decimalnih cifara
boolean	1 byte	Pohranjuje istinite ili netačne vrednosti
char	1 byte	Pohranjuje jedan znak / slovo / broj ili ASCII vrednosti

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Numerički tipovi

Koristite int kada treba da sačuvate ceo broj bez decimalnih mesta, kao što je 35 ili 1000, plutajući ili dvostruki kada vam je potreban broj sa pomičnom tačkom (sa decimalnim mestima), kao što je 9,99 ili 3,14515.

int

```
int myNum = 1000;
cout << myNum;
```

float

```
float myNum = 5.75;
cout << myNum;
```

double

```
double myNum = 19.99;
cout << myNum;
```

float vs. double

Vrednost tačnosti sa pomičnim zarezom označava koliko cifara vrednost može imati posle decimalne tačke. Tačnost float-a iznosi samo šest ili sedam decimalnih mesta, dok dvostruke promenljive imaju tačnost od oko 15 cifara. Zbog toga je sigurnije koristiti dvostruko za većinu proračuna.

Naučni brojevi

Broj sa pokretnom zarezom takođe može biti naučni broj sa „e“ koji označava jačinu 10:

Primer

```
float f1 = 35e3;
double d1 = 12E4;
cout << f1;
cout << d1;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Bulovski tipovi

Logički tip podataka se deklariše sa ključnom reči bool i može da prihvati samo vrednosti true ili false. Kada se vrati vrednost, true = 1 i false = 0.

Primer

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun; // Outputs 1 (true)
cout << isFishTasty; // Outputs 0 (false)
```

Logičke vrednosti se uglavnom koriste za uslovno testiranje.

Vrste simbola

Podaci tipa char koriste se za čuvanje jednog znaka. Znak mora biti okružen pojedinačnim navodnicima, kao što su „A“ ili „c “:

Primer

```
char myGrade = 'B';
cout << myGrade;
```

Kao alternativu, možete koristiti ASCII vrednosti za prikaz određenih simbola:

Example

```
char a = 65, b = 66, c = 67;
cout << a;
cout << b;
cout << c;
```

Tipovi nizova (strings)

Tip niza koristi se za čuvanje niza znakova (teksta). Ovo nije ugrađeni tip, ali se ponaša kao takav u svojoj najosnovnijoj upotrebi. Stalne vrednosti moraju biti zatvorene u dvostruke navodnike:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Primer

```
string greeting = "Hello";
cout << greeting;
```

Da biste koristili nizove, u izvorni kod, biblioteku <string>, morate uključiti dodatnu datoteku

Primer

```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";

// Output string value
cout << greeting;
```

3. Konzistentno i uslovno izvršenje, interaktivna rešenja, funkcije;

Dosledno i uslovno izvršenje

U programiranju imamo posebnu grupu uputstava koja krše konceptualni redosled sekvencijalnog izvođenja u Nojmanovoj arhitekturi. Ova uputstva uključuju uputstva za uslovni prelaz (grana), bezuslovni prelaz (skok),

Aspekti kontrolnih uputstava

• Biće / neće biti tranzicije.
• Kako se ugrađuje (postavlja) uslov u granu (često uslov proverava registar zastava). Registar zastava određen je operacijama koje je izvršio ALU. Zastava se menja sa svakom operacijom.
• Kako izračunati prelaznu adresu.
• Link Return Address - veže povratne adrese (urađeno u prelaznim pozivima i povratima - tj. u potprogramima)

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Kontrolna uputstva

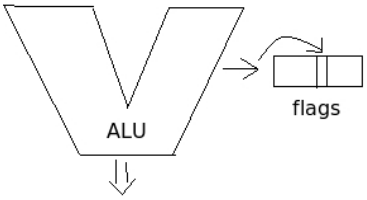
• skok za bezuslovnu tranziciju;
• ogranak za uslovnu tranziciju

Detalji

skok / grana

jump uvek menja sadržaj programskog brojača, dok grana ima uslov, a PC se menja samo ako je taj uslov ispunjen. Kako će biti ugrađeno stanje grane, stvar je primene arhitekture. Uslovni prelazi se mogu izvršiti na takav način da se u njima izračunava vrednost uslova. Najčešće je uslov poređenje - prelazak se vrši ako je rezultat upoređivanja pozitivan, pa najčešće uputstva grane opisuju šta se upoređuje i adresu prelaza. Poređenje se vrši u ALU, ovo je jedna vrsta arhitekture sa uslovnim prelazima sa uputstvima za upoređivanje. U drugoj vrsti arhitektura, uputstva za upoređivanje su odvojena od uputstava za uslovni prelaz i pozivaju se neposredno pre.

Zastave(FLAGS)



Uslov često testira registar zastava. Znakovi rezultata evidentiraju se u registru zastava. Svaka aritmetička operacija stavlja neke zastavice. To su posebna pravila po kojima se prepoznaje uslov prelaza, u zavisnosti od stanja zastava.

Nedostatak kodova uslovnog prelaza je taj što oni pružaju implicitnu vezu između uputstava - na primer, uslovni prelaz se mora izvršiti nakon izvođenja aritmetičke operacije sa ALU za brojeve sa fiksnom tačkom. U cevovodu (transporter) to može poremetiti brzinu - u uputstvima za prelaz ceo transporter je očišćen.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Formiranje prelazne adrese

Prelazna adresa se formira na 4 moguća načina:

1. kao relativna adresa brojaču programa (PC + ofset);
2. Po osnovnom registru i deplasmanu (Rbase + deplasman);
3. Postavljanjem apsolutne adrese (poseban slučaj gore navedenog na Rbase == 0);
4. Kroz vektore.

Spremanje registara

Poslednji aspekt kontrolnih uputstava je čuvanje registara. Da bi brže radio, program radi sa operandima zapisanim u registre. Sa svakim prelaskom na potprogram, ovaj kontekst se gubi i mora se vratiti po povratku. Ovaj program mora imati oblast za čuvanje u kojoj se ovaj kontekst privremeno čuva. Softverska konvencija je pozivna funkcija koja se brine da zadrži svoj kontekst i obnovi ga pri povratku iz potprograma. Lokacija povratka mora imati uputstva za vraćanje konteksta, tj. Svaki put se vrši višestruko učitavanje i više memorija, da bi se sačuvali i učitali svi registri. Postoji registar Rsa u kojem se čuva adresa ovog spremišta. Postoji još jedna konvencija (callee) u kojoj se podprogram brine pre povratka da registre popuni informacijama, kao što je bilo kada je pozvan.

Grupe registara

-u - argumenti prihvaćeni potprogramom
-out - rezultat izvršenja (može se vratiti programu koji poziva)
-local - local variable
-global - global variable

Prilikom ulaska uštedite na i lokalno, a kada napuštate lokalno. Za sistemske pozive vode se svi registri, a za redovne samo osnovni.

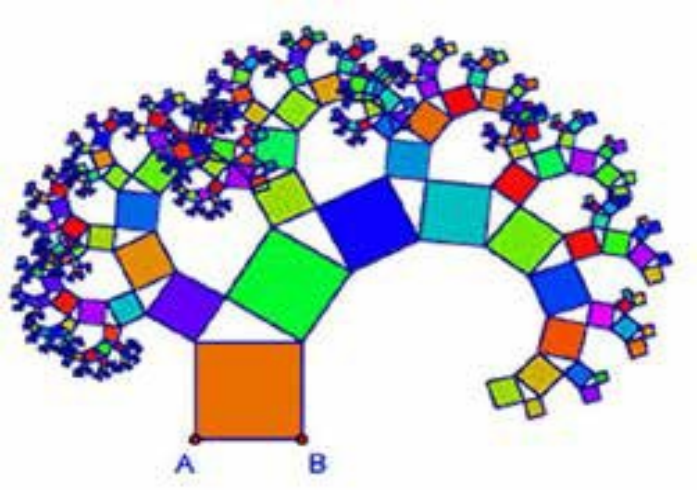
Iterativna rešenja

Uvod u termin „iteracija“

Iteracija (na latinskom iterare „ponavljanje“) je termin koji se obično koristi za označavanje „ponavljanja procesa“. Iteracija je korak u procesu koji se najčešće koristi za rešavanje kvantitativnih problema i koji se ponavlja nekoliko puta. Svako ponavljanje naziva se iteracijom.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

U programiranju, najčešće ponavljanje znači „petlja“ (povratak u stanje dok ne postane istina i program se nastavi).



Fraktalno pitagorejsko drvo

Fraktalno slikanje je usko povezano sa programiranjem. Za ljude koji imaju čak i nejasnu predstavu o programiranju, ovi koncepti su jasni. Za ostale:

Iteracija u programiranju je organizacija obrade podataka u kojoj se radnja ponavlja nekoliko puta ne dovodeći do situacije u kojoj se funkcija poziva (ovo je već rekurzija).

Generalno, iteracija se može prevesti kao „ponavljanje“.

Svaki fraktal ima beskrajno ponavljajući oblik. Prilikom stvaranja takvog fraktala, najjednostavniji način je ponoviti nekoliko radnji koje stvaraju ovaj oblik. Umesto reči „ponavljanje“ koristimo „ponavljanje“. Praktično bilo koji fraktal može da se stvori pomoću ponavljanja nekog pravila. Da biste stvorili pravi fraktal, morate beskrajan broj puta da ponovite. Ali mogli bismo to da učinimo konačan broj puta, ali dovoljno dugo da dobijemo ideju o „pravom“ fraktalu. Prirodno uz pomoć računara. Povećavanje broja ponavljanja čini fraktale tačnijim

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Postoje tri glavne vrste ponavljanja:

1. Zamenljiva iteracija - stvara fraktale, zamenjujući neke geometrijske oblike drugima.

Počinjemo sa figurom koja se naziva baza. Zatim svaki deo osnove zamenimo drugom figurom koja se naziva motiv. Ovu zamenu izvodimo beskonačno mnogo puta dok ne završimo fraktal.

L-sistemi

Zamenljiva iteracija je vrlo jednostavna. Ali za računar nije dovoljno imati sliku osnove i motiva. Potreban je jasan i tačan način čuvanja fraktalnih podataka kako bi se omogućilo izvlačenje jednostavnih algoritama za crtanje fraktala.

L-sistemi su dobar i jednostavan način. Razvio ih je A. Lindenmeier („L“ u reči „L-sistem“). Sastoje se od ugla, aksioma i bar jednog pravila. Aksiomom nazivamo početni oblik (bazu) koji će se koristiti u procesu stvaranja fraktala. Pravila određuju koje simbole u aksiomu treba zameniti drugim simbolima

SIMBOLI					
UOBIČAJENI SIMBOLI		KOMPLEKSNI SIMBOLI		SIMBOLI BOJE	
F	Pomera se unapred, sa tragom	**@n**	Pomnožiti dužinu segmenta sa **n**	**Cn**	Određivanje broja boje n
G	Pomera se napred bez traga	**I**	Isped broja znači deljenje umesto množenje	**<n**	Smanjivanje broja boje za n
+	Rotiranje u smeru obrnutom od kazaljke na satu	**Q**	Isped broja daje kvadratni koren	**> n**	Povećavanje broja boje za n
-	Rotiranje u smeru kazaljke na satu	**!**	Menja znak iz + u -		
|	Rotiranje za180º	**[**	Unosi trenutnu poziciju kursora u memoriju		
		]	Uklanja poslednju sačuvanu poziciju kursora iz memorije		

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Većina fraktala sa fraktalnom dimenzijom od 0 do 2 može se izraziti pomoću L-sistema. Kombinacijom nekoliko simbola i pravila mogu se stvoriti vrlo složeni fraktali. Takvi L-sistemi koriste se za stvaranje realnih biljnih modela.

Takođe je moguće postići veći realizam uvođenjem parametra koji dodaje slučajne brojeve. Što je njihov opseg veći, to će oblici izgledati prirodnije i daleko od simetrije.

2. **Iteracija IFS** - stvara fraktale, primenjujući geometrijske transformacije (vrsta rotacije i refleksije).

IFS (iterativni funkcionalni sistemi) predstavljaju još jedan način za stvaranje fraktala. Ova metoda se zasniva na tački ili figuri, koja je zamenjena sa nekoliko manjih figura.

3.Iteracija sa formulama - uključuje nekoliko načina za stvaranje fraktala, ponavljanje neke matematičke formule ili nekoliko formula.

Iteracija formule je najjednostavniji tip iteracije, ali je najvažnija i daje najsloženije rezultate. Algebarski fraktali se konstruišu iteracijom formule, tj. Matematičkim formulama. O njima ćemo razgovarati u temi Algebarski fraktali.

Funkcije

Funkcije su osnovne strukturne jedinice iz kojih se razvijaju programi. Svaka funkcija sastoji se od više operatora (možda 0), koji se izvršavaju kao jedna opšta operacija ili radnja. Jednom kada se kreira funkcija, koja se postiže njenim definisanjem, može se izvršavati više puta. Svako izvršavanje funkcije može imati različite podatke, jer se određeni podaci postavljaju prilikom pozivanja / aktiviranja / funkcije. Kao rezultat njihovog izvršavanja, funkcije mogu vratiti vrednost koja se naziva povratna vrednost.

Svaki program se sastoji od 1 ili više funkcija. Samo jedna od funkcija u programu nosi naziv glavna. Glavna funkcija je prva funkcija koja se izvršava prilikom pokretanja programa. Dobar stil programiranja zahteva razvoj programa iz mnogih malih funkcija, koje:

• čini programe jasnijim, lakšim za testiranje, podešavanje i modifikovanje;
• izbegava se ponavljanje istih fragmenata programa. Ovi fragmenti se mogu jednom definisati kao funkcije, a zatim izvršavati više puta;
• ušteda memorije je postignuta jer se kod funkcije čuva samo na jednom mestu u memoriji, bez obzira na broj njegovih izvršavanja.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Funkcije su dve vrste:

• funkcija koja vraća vrednost nekog tipa / true, real / function / poziva se kao operand u izrazu i ekvivalentna je jednom množitelju /;
• funkcija koja ne vraća vrednost / postupak u Pascalu / - poziva se kao nezavisni operator u drugoj funkciji;

4 Nizovi, matrice i njihove primene

Niz je uređena sekvenca elemenata istog osnovnog tipa. Pojedinačni element niza označen je imenom celog niza, a zatim rednim brojem (indeksom) elementa. U različitim programskim jezicima, nizovi se grade na različite načine.

Karakteristike nizova

Niz može biti jednodimenzionalni, višedimenzionalni ili niz nizova.

Zasnivaju se na indeksiranju nula - to znači da će u nizu sa N elemenata prvi element imati indeks nula, a poslednji indeks N-1.

Elementi niza mogu biti bilo kog tipa, uključujući tip niza.

Podrazumevana vrednost numeričkih elemenata tipa je nula, za referentne tipove je nula, a za logičke tipove je netačna. U nizu nizova elementi su referentnog tipa i podrazumevano su nula.

Redosled elemenata i dužina niza su fiksni.

Multidimenzionalni nizovi

Za dvodimenzionalni niz, elementi sa indeksima i, j imali bi adresu B + ci + dj, gde su koeficijenti c i d inkrementalni koraci reda i kolone.

Uopštenije, u k-dimenzionalnom nizu adresa elementa sa indeksima i1, i2, ..., ik je:

B + c1 · i1 + c2 · i2 +… + ck · ik. Na primer: int a [3] [2];

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

To znači da je niz tipa int i da ima 3 reda i 2 kolone. U njega možemo smestiti 6 elemenata, poređanih linearno, počev od prvog reda. Gornji niz je uskladišten u sledećem redosledu: a11, a12, a13, a21, a22, a23.

Ova formula zahteva samo k množenja i k dodavanja za bilo koji niz koji se može sačuvati u memoriji. Pored toga, ako je koeficijent tačan eksponent 2, množenje može biti zamenjeno bitnim pomacima.

Koeficijenti ck moraju biti takvi da svaki važeći indeks vodi do adrese određenog elementa.

Ako je minimalna dozvoljena vrednost za svaki indeks 0, tada je B adresa elementa na kojem su svi indeksi 0. Kao i kod jednodimenzionalnih nizova, indeksi elemenata mogu biti promenjeno promenom osnovne adrese B. Stoga, ako dvodimenzionalni niz ima redove i kolone sa indeksima od 1 do 10, odnosno od 1 do 20, onda će zamena B sa B + c1 - - 3 c1 prebrojati indekse elemenata na 0 do 9, odnosno 4 do 23. Koristeći prednost ovog svojstva, neki programski jezici (poput FORTRAN 77) postavljaju indekse niza da počinju na 1, što je tradicionalno matematički, dok drugi jezici (poput Fortran 90, Pascal i Algol) omogućavaju korisniku da odabere minimalnu vrednost svakog indeksa.

Obrada nizova

sa petlja for - koristi se pri radu sa indeksom elemenata i nije potrebno puzati svaki od prvog do poslednjeg elementa.

sa foreach petljom - koristi se kada upotreba indeksa nije potrebna i kada se elementi indeksiraju jedan po jedan. Ovom operacijom elementi se ne mogu menjati, već samo čitati.

Nizovi kao objekti

U jeziku C #, nizovi su zapravo objekti. Sistem.Arrai je osnovni tip svih vrsta nizova i njegove karakteristike se mogu primeniti. Primer ovoga je pronalaženje dužine niza metodom Sistem.Arrai.Length.

Skalabilni nizovi

To su nizovi koji se mogu dinamički menjati dodavanjem ili uklanjanjem elemenata iz njih. Sintaksa je List <T> - gde je T vrsta podataka koji će biti sadržani. Njihova glavna prednost je što ne moramo

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

unapred da znamo dužinu niza. U početku novostvorena lista ima 0 elemenata. Osnovne metode i svojstva:

: Dodaj (T element) - dodaje element na kraju

: Remove (element) - uklanja element

: Count - vraća trenutnu dužinu liste

Nizovi u C / C ++

U programskim jezicima C i C ++, niz je grupa promenljivih koje su istog tipa podataka, imaju istu veličinu, sekvencijalno su raspoređene u memoriji i postoji pokazivač koji pokazuje na prvi element niza.

Nizovi u PHP-u

U programskom jeziku PHP, niz je promenljiva koja sadrži mnogo elemenata. Nizovi mogu biti 2 vrste - obični i asocijativni. Za obične nizove koristi se indeks (redni broj) elementa koji se odnosi na element. U asocijativnim nizovima, tekstualni niz se koristi za upućivanje na element.

Nizovi u JavaScript-u

U programskom jeziku JavaScript niz je objekat. Ima konstruktor i metode. Indeksi ili tekstualni nizovi mogu se koristiti za upućivanje na elemente objekta.

Nizovi u Pithon-u

U programskom jeziku Pithon, nizovi su 2 tipa - tuple i list. Razlika između njih je u tome što se korpica ne može promeniti - sve dok postoji, njeni elementi ostaju takvi kakvi su bili i kada je stvoren. Na listi, elementi se mogu menjati.

Nizovi u C #

Nizovi na jeziku C # su kolekcija nekoliko promenljivih istog tipa. Oni se nazivaju elementima niza

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Matrice

Dvodimenzionalni niz se takođe naziva matricom koja ima postavljene kolone

Postoje dve vrste matrice:

1. Kvadratna matrica gde su kolone jednake redovima
2.Pravougaona matrica gde kolone NISU jednake redovima

Vrste matrica:

• kvadratna matrica - broj redova je jednak broju kolona
• simetrična matrica - kvadratna matrica čiji su elementi simetrično locirani oko glavne dijagonale jednaki
• trouglasta matrica - kvadratna matrica u kojoj su svi elementi ispod ili iznad glavne dijagonale nule, odnosno gornja ili donja trouglasta matrica;
• dijagonalna matrica - kvadratna matrica čiji su elementi koji nisu nula samo u glavnoj dijagonali;
• skalarna matrica - dijagonalna matrica, svi elementi glavne dijagonale su jednaki
• pojedinačna matrica - skalarna matrica sa elementima glavne dijagonale jednakim jedinici

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

5. Elementi za obradu niza;

Niz u računarstvu je konačan niz simbola (predstavlja konačan broj znakova). Njegovi elementi se mogu menjati, kao i dužina. Nizovi se obično tretiraju kao tip podataka i često se pretvaraju u nizove bajtova (ili reči) koji čuvaju niz elemenata pomoću kodiranja simbola.

U zavisnosti od programskog jezika i upotrebljenog tipa podataka, promenljiva deklarisana kao niz može se memorisati statički (unapred definisana maksimalna dužina) ili dinamički (može sadržati različit broj elemenata).

U formalnim jezicima koji se koriste u matematičkoj logici i teorijskoj računarskoj nauci, niz je konačan niz simbola iz skupa koji se naziva abeceda.

Dužina niza

Iako nizovi mogu biti proizvoljne (ali ograničene) dužine, u većini programskih jezika dužina je ograničena na veštački maksimum. Općenito postoje dvije vrste žica:
• sa fiksnom dužinom - imaju konstantnu maksimalnu dužinu i koriste istu količinu memorije, bez obzira na to da li je taj maksimum dostignut;
• sa promenljivom dužinom.

Formalna teorija

Označimo sa S abecedu, koja je prazan konačan skup. Elementi S nazivaju se slovima. Niz (ili reč) od S (S *) je bilo koji konačan niz slova od S. Na primer, ako je S = {0, 1}, onda je 0101 niz abecede S.

Skup svih nizova preko S dužine n zapisuje se Sn. Primer: ako je S = {0, 1}, onda je S2 = {00, 01, 10, 11}. Imajte na umu da je S0 = {e} za svaku abecedu S.

Skup nizova nad S (na primer, bilo koji podskup S *) naziva se formalnim jezikom nad S. Primer: ako je S = {0, 1}, skup nizova sa jednakim brojem nula i jedinica ({e, 1, 00, 11, 001, 010, 100, 111, 0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111, ...}) je formalno
jezik nad S.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Kodiranje simbola

U .NET Framevork-u svaki simbol ima redni broj iz Unicode tabele. Standard Unicode stvoren je krajem 80-ih i početkom 90-ih za skladištenje različitih vrsta tekstualnih podataka. Njegov prethodnik ASCII omogućava snimanje samo 128 ili 256 simbola (odnosno ASCII standard sa 7-bitnom ili 8-bitnom tabelom). Nažalost, ovo često ne zadovoljava potrebe korisnika - jer 128 simbola može stati samo na brojeve, mala i velika latinična slova i neke posebne simbole. Kada je potrebno raditi sa tekstom na ćirilici ili nekom drugom određenom jeziku (na primer, azijski ili afrički), 128 ili 256 simbola je krajnje nedovoljno. Zbog toga .NET koristi 16-bitnu tabelu kodova simbola. Uz pomoć našeg znanja o brojevnim sistemima i predstavljanja informacija u računarima, možemo pretpostaviti da tabela kodova sadrži 2 ^ 16 = 65536 simbola. Neki od simbola su kodirani na specifičan način, tako da je moguće koristiti dva simbola iz Unicode tabele za stvaranje novog simbola - rezultujući znakovi premašuju 100.000. Tipovi podataka niza su u prošlosti koristili po jedan bajt po simbolu. Nekim jezicima kao što su kineski, japanski i drugi (poznati i kao CJK) potrebno je mnogo više od 256 simbola. Jedno rešenje ovog problema je očuvanje jednobajtnog ASCII predstavljanja i upotreba dvobajtnog za jezike CJK. Ovo
dovodi do problema sa podudaranjem nizova, gubicima dužine i još mnogo toga. Sa Unicode-om, stvari su po tom pitanju znatno pojednostavljene i većina programskih jezika to podržava.

Glavni elementi obrade nizova su:

• Operacija dodele
• Pristup pojedinačnim simbolima
• Poređenje nizova simbola
• Izdvoji deo niza
• Funkcija za spajanje nizova simbola
• Funkcija za određivanje dužine (broja simbola) niza simbola
• Postupak za pretvaranje niza simbola u numeričku vrednost
• Postupak za pretvaranje broja u niz simbola

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

6. Strukture

Ovo je poseban tip koji kreiramo tako što preciziramo koje informacije (koje tipove podataka i koliko promenljivih) sadrži.

Na primer, često u geometrijskim zadacima moramo zadržati tačke (recimo u trodimenzionalnom prostoru). Da bi zadržali N takvih tačaka, mnogi takmičari bi koristili tri niza dupla k [N], i [N], z [N], i u svakom nizu bi zadržali jednu od koordinata tačaka. Mnogo je situacija u kojima bi ovaj jednostavan način doveo do nezgodnijeg koda. Na primer, da bismo funkciji prosledili tačku, moramo svaku od njenih koordinata da prosledimo zasebno. Druga opcija je da pošaljete indeks i tada nizovi moraju biti globalni. Još veći problem je ako želimo da preuredimo tačke.

Obično koristimo strukturu kada samo želimo da inkapsuliramo složenije informacije, a klasu kada želimo da imamo dodatne metode koje vrše neku radnju na tim podacima.

Umesto toga, međutim, možemo stvoriti strukturu koja za nas predstavlja tačku. To bi bila struct Point {double k, i, z;};, i trebao bi nam samo jedan niz (umesto tri): Point points
[N];. Prenošenje tačke u funkciju vrši se prenošenjem jedne promenljive tipa Point. Preuređivanje tačaka (na primer pri sortiranju) postaje mnogo lakše, ali o tome ćemo kasnije u ovoj temi.

Strukture pružaju mnogo više slobode. Na primer, nema problema u skladištenju različitih vrsta promenljivih u njemu. Ako želimo da imamo momka koji je student, možda bismo želeli da znamo njegovo ime, prezime, PIN, prosek ocena i broj predmeta. Elementi složenog tipa (struktura ili klasa) nazivaju se „objekti“. Otuda i naziv „Objektno orijentisano programiranje“.

Struktura omogućava grupisanje podataka različitih vrsta u jednu logički povezanu celinu. U tome se struktura razlikuje od niza, koji je skup elemenata istog tipa.

Upotreba struktura podataka

Najčešće se strukture podataka koriste kao deo algoritma - u zavisnosti od izbora strukture podataka, to je različito brzo. Neki trkački zadaci stvoreni su posebno za zanimljivu strukturu podataka koja zahtevaju samo njena svojstva.

Osnovni tipovi konstrukcija

Linearni

Linearne strukture podataka su liste, hrpe i redovi.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Linearna lista je niz elemenata istog tipa. Glavne operacije koje se mogu izvoditi sa elementima su dodavanje i uklanjanje.

Vrste linearnih struktura

• linearna jednostrano povezana lista - svaka stavka, osim poslednje, povezana je sa sledećom jednom vezom. Lista se popisuje od početka do kraja.
• linearna dvostruko povezana lista - svaki element, osim poslednjeg, povezan je sa sledećim pomoću dve veze. Ovo pojednostavljuje rad. Na primer, stavci liste je lako dostupan u zavisnosti od toga da li je bliža početku ili kraju liste.
• ciklična lista - dvostruko vezana ili jednopovezana lista, u kojoj je poslednji element ujedno i prethodnik prve. Ova implementacija uklanja uslovni redosled stavki na listi i pojednostavljuje operacije sa njima.
• paralelna lista - Skup od nekoliko lista. Moguć je paralelni pristup njihovim elementima.
• stek - elementi u jednom steku se dodaju i uklanjaju samo s jednog kraja, poštujući pravilo LIFO (poslednji u prvom izlazu - iz engleskog „poslednji koji ulazi prvi odlazi“), odnosno najnoviji dodati element je prvi za pristup steku. Dakle, operacije na elementima su ograničene.
• red - pristup stavkama reda je takođe ograničen kao i kod steka. Međutim, ovde važi FIFO pravilo (prvi ulaz, prvi izlaz - od engleskog „prvi koji uđe prvi odlazi“), prema kojem je element koji je najduže u redu (dodaje se najranije) prvo obrađena. Dodavanje stavki vrši se samo sa kraja reda, a uklanjanje - s početka.

Strukture nalik drvetu

Strukture podataka o drvetu uključuju različite vrste stabala. [3]

Drveće su mrežne strukture podataka, jedan od najvažnijih koncepata u teoriji grafova. Slede tri ekvivalentne definicije pojma „neorijentisano stablo“:

• povezano drvo koje sadrži n vrhova i n-1 ivica;
• povezano drvo koje ne sadrži petlju;
• Drvo u kojem je svaki par vrhova povezan jednostavnim lancem:
Ako je {\ displaistile G = (Ks, A)} {\ displaistile G = (Ks, A)} neorijentisano stablo sa {\ displaistile n} n vrhova, tada se bilo koje drvo formirano njegovim lukovima naziva „drvo pokrivača“, ako uključuje sve vrhove drveta. Očigledno je da drvo pokrivača ima {\ displaistile n-1} {\ displaistile n-1} ivice.

Orijentisano stablo: orijentisano stablo bez petlji, u kojem je stepen unosa svakog vrha (osim jednog) jednak 1, a označenog izuzetka vrha (zvanog koren) 0.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Drveće su mrežne strukture podataka, skup skupa Ks čiji se elementi nazivaju vrhovi, a skup A uređenih parova vrhova koji se nazivaju lukovi (ivice). Oznaka je (Ks, A).

Niz

Niz je kolekcija elemenata (vrednosti ili promenljivih) kojima se može direktno pristupiti putem indeksa.

Poređenje osnovnih struktura podataka

Nakon što se upoznamo sa konceptom složenosti algoritma, spremni smo da uporedimo osnovne strukture podataka koje smo do sada razmatrali i da procenimo složenost svake od njih izvršavajući osnovne operacije kao što su dodavanje, pretraživanje, brisanje i druge. To će nam omogućiti da u skladu sa operacijama koje su nam potrebne lako razmotrimo koja će struktura podataka biti najprikladnija.

Kada koristiti strukturu?

Pogledajmo svaku strukturu podataka navedenu u tabeli odvojeno i objasnimo u kojim situacijama je prikladno koristiti takvu strukturu i kako se dobijaju složenosti date u tabeli.

Niz (T [])

Nizovi su uređeni skupovi fiksnog broja elemenata datog tipa (kao što su brojevi) kojima se pristupa indeksom. Nizovi su područje memorije određene, unapred definisane veličine. Dodavanje novog elementa u niz vrlo je spora operacija, jer zapravo trebate dodeliti novi niz dimenzije veće od 1 trenutne i prenijeti stare elemente u novi niz. Pretraživanje niza zahteva poređenje svakog elementa sa vrednošću pretrage. U prosečnom slučaju su potrebna N / 2 poređenja. Brisanje iz niza je vrlo spora operacija, jer uključuje dodeljivanje niza 1 veličine manje od trenutnog i premeštanje svih elemenata bez brisanja u novi niz. Pristup indeksu je direktan i samim tim vrlo brz rad.

Nizovi se trebaju koristiti samo kada je potrebno da obradimo fiksni broj elemenata kojima je potreban pristup indeksu. Na primer, ako sortiramo brojeve, brojeve možemo zapisati u niz i primeniti jedan od dobro poznatih algoritama za sortiranje. Kada treba da promenimo broj elemenata sa kojima radimo tokom rada, niz nije odgovarajuća struktura podataka.

Koristite nizove kada trebate obraditi fiksni broj elemenata kojima trebate pristupiti indeksom.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Povezana / dvostruko povezana lista (LinkedList <T>)

Povezana lista i njena varijanta dvostruko povezane liste čuvaju uređeni skup elemenata. Dodavanje je brza operacija, ali je malo sporije od dodavanja na List <T>, jer svako dodavanje zauzima memoriju. Dodela memorije radi brzinom koju je teško predvideti. Pretraživanje na povezanoj listi je spora operacija jer uključuje popisivanje svih njenih elemenata. Pristup elementu indeksom je spora operacija, jer na povezanoj listi nema indeksiranja i listu morate indeksirati, počevši od prvog elementa i krećući se napred element po element. Brisanje elementa indeksom je spora operacija, jer je postizanje elementa sa navedenim indeksom spora operacija. Brisanje elementa po vrednosti je takođe sporo jer uključuje pretragu.

Povezana lista može brzo (uz konstantnu složenost) dodavati i brisati elemente na oba kraja, čineći je pogodnom za primenu stekova, redova i drugih sličnih struktura.

Povezana lista se retko koristi u praksi jer dinamički proširivi niz (List <T>) izvodi gotovo sve operacije koje se mogu izvoditi sa LinkedList-om, ali za većinu njih radi brže i pogodnije.

Koristite List <T> kada vam je potrebna povezana lista - ona ne radi sporije, ali vam pruža veću brzinu i udobnost. Koristite LinkedList ako trebate dodavati i brisati elemente na oba kraja strukture.

Koristite povezanu listu (LinkedList <T>) kada treba da dodate i izbrišete elemente na oba kraja liste. U suprotnom, koristite List <T>.

Dinamički niz (Lista <T>)

Dinamički niz (List <T>) jedna je od najčešće korišćenih struktura podataka u praksi. Nema fiksnu veličinu, poput nizova, i omogućava direktan pristup indeksom, za razliku od povezane liste (LinkedList <T>). Dinamički niz je takođe poznat kao imena „lista nizova“ i „dinamički proširivi niz“.

List <T> interno čuva svoje elemente u nizu koji je veći od broja uskladištenih elemenata. Pri dodavanju elementa obično ima slobodnog prostora u unutrašnjem nizu, tako da ova operacija traje konstantno. Ponekad se niz preplavi i treba ga proširiti. Ovo zahteva linearno vreme, ali je vrlo retko. Konačno, za veliki broj dodataka, prosečna složenost dodavanja elementa na List <T> je konstantna - O (1). Ova prosečna složenost naziva se amortizovana složenost. Amortizovana linearna složenost znači da ako sekvencijalno dodamo 10.000 elemenata, izvešćemo ukupan broj koraka reda od 10.000 i većina njih će se izvršiti u konstantnom vremenu, a ostatak (vrlo mali deo) izvršiće se linearno vreme.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pretraživanje u Listu <T> je spora operacija jer svi elementi moraju biti popisani. Brisanje indeksom ili vrednošću vrši se u linearnom vremenu. Brisanje je spora operacija, jer uključuje premeštanje svih elemenata koji su nakon izbrisanih za jedan položaj ulevo. Pristup indeksu u Listu <T> je neposredan, stalno, jer se elementi interno čuvaju u nizu.

U praksi, List <T> kombinuje snage nizova i lista, čineći ga preferiranom strukturom podataka u mnogim situacijama. Na primer, ako treba da obradimo tekstualnu datoteku i iz nje izdvojimo sve reči koje odgovaraju regularnom izrazu, najpogodnija struktura u kojoj ih možemo akumulirati je List <T>, jer nam je potreban spisak čija dužina nije ranije poznat i da dinamički raste.

Dinamički niz (List <T>) pogodan je kada moramo često da dodajemo elemente i želimo da zadržimo redosled njihovog dodavanja i da im često pristupamo indeksom. Ako često tražimo ili brišemo neki element, List <T> nije odgovarajuća struktura.

Koristite List <T> kada trebate brzo dodati elemente i pristupiti im indeksom.

Gomila(Stack)

Gomila je struktura podataka u kojoj su definisane 3 operacije: dodavanje elementa na vrh steka, brisanje elementa sa vrha steka i preuzimanje elementa sa vrha steka bez uklanjanja. Sve ove operacije se izvode brzo, uz konstantnu složenost. Operacije indeksnog pretraživanja i pristupa nisu podržane.

Gomila je struktura sa LIFO ponašanjem (poslednji ulaz, prvi izlaz) - poslednji put unesen, prvi izlaz. Koristi se kada treba da modeliramo takvo ponašanje, na primer, ako treba da zadržimo put do trenutne pozicije u rekurzivnoj pretrazi.

Koristite stack kada treba da primenite ponašanje poslednjeg ulaska (LIFO).

Red čekanja

Red je struktura podataka u kojoj su definisane dve operacije: dodavanje elementa i preuzimanje elementa koji je u redu. Ove dve operacije se izvode brzo, uz konstantnu složenost, jer se red obično implementira preko povezane liste. Imajte na umu da povezana lista može brzo dodavati i brisati elemente na oba kraja.

Ponašanje strukture reda je FIFO (prvi ulaz, prvi izlaz) - prvi put ušao, prvi izašao. Operacije indeksnog pretraživanja i pristupa nisu podržane. Red prirodno modelira listu ljudi koji čekaju, zadatke ili druge objekte koje treba uzastopno obrađivati, redosledom kojim ulaze.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Kao primer korišćenja reda možemo pomenuti implementaciju algoritma „pretraga u širini“, koji započinje datim početnim elementom, a njegovi susedi se dodaju u red, zatim obrađuju po redosledu prijema i tokom njihove obrade susedi se dodaju u red. To se ponavlja sve dok se ne dostigne element koji tražimo.

Koristite red kada treba da primenite ponašanje prvo ulazak, izlazak (FIFO).

Rečnik implementiran sa heš tabelom (Rečnik <K, T>)

Struktura „rečnika“ pretpostavlja skladištenje parova ključ / vrednost i omogućava brzo pretraživanje ključeva. Kada se implementira sa heš-tabelom (klasa Dictionari <K, T>) u .NET Framevork), dodavanje, pretraživanje i brisanje elemenata rade vrlo brzo - uz konstantnu složenost u prosečnom slučaju. Operacija pristupa indeksu nije dostupna jer su elementi u heš tabeli raspoređeni gotovo nasumično i njihov redosled unosa nije sačuvan.

Rečnik <K, T> interno čuva svoje elemente u nizu, postavljajući svaki element u položaj dat heš funkcijom. Na ovaj način, niz je delimično zauzet - u nekim ćelijama
postoji vrednost, dok su drugi prazni. Ako je potrebno postaviti nekoliko vrednosti u istu ćeliju, one će biti raspoređene u povezanu listu (ulančavanje). Ovo je jedan od načina za rešavanje problema sudara. Kada stopa popunjenosti heš tabele pređe 100% (ovo je podrazumevana vrednost parametra faktora opterećenja), njena veličina se udvostručuje i svi elementi zauzimaju nove položaje. Ova operacija radi linearno složeno, ali se izvodi tako retko da amortizovana složenost operacije dodavanja ostaje konstantna.
Tabela heširanja ima jednu osobinu: sa nepovoljno odabranom heš funkcijom koja uzrokuje mnoštvo sudara, osnovne operacije mogu postati prilično neefikasne i dostići linearnu složenost. Međutim, u praksi se to teško dešava. Stoga se tabela heširanja smatra najbržom strukturom podataka koja pruža dodavanje i pretraživanje ključeva.

Tabela heširanja u .NET Framevork pretpostavlja da se svaki ključ pojavljuje u njoj najviše jednom. Ako dva elementa uzastopno napišemo istim ključem, poslednji će zameniti prethodni i na kraju ćemo izgubiti jedan element. Ovo je važna karakteristika koju moramo uzeti u obzir.

Ponekad moramo da pohranimo nekoliko vrednosti u jedan ključ. Ovo nije podrazumevano podržano, ali možemo koristiti List <T> kao vrednost za ovaj ključ i u njemu akumulirati niz elemenata. Na primer, ako nam treba hash tabela Dictionari <int, string> u kojoj ćemo akumulirati parove
{integer, string} sa iteracijama možemo koristiti Dictionari <int, List <string>>.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tabela heš se preporučuje da se koristi kad god nam je potrebna brza pretraga ključeva. Na primer, ako treba da izbrojimo koliko se puta svaka reč pojavljuje u tekstualnoj datoteci među datim brojem reči, možemo da koristimo Dictionari <string, int> koristeći reči za pretragu kao ključ, a za vrednost - koliko puta se javljaju u datoteci.

Koristite heš tabelu kada želite brzo da dodate elemente i pretražite po ključu.

Mnogi programeri (posebno početnici) žive u zabludi da je glavna prednost heš tabele pogodnost traženja vrednosti po ključu. Zapravo, glavna prednost uopšte nije u tome. Pretragu ključeva možemo izvršiti pomoću niza i liste, pa čak i nizom. Nema problema, svako može da ih primeni. Možemo definisati klasu Entri koja čuva ključ i vrednost i raditi sa nizom ili listom elemenata Entri. Možete izvršiti pretragu, ali u bilo kojoj
slučaj će raditi polako. Ovo je veliki problem sa listama i nizovima - oni ne nude brzu pretragu. Nasuprot tome, heš tabela može brzo da pretražuje i brzo dodaje nove stavke.

Glavna prednost heš tabele u odnosu na druge strukture podataka je izuzetno brzo pretraživanje i dodavanje elemenata. Pogodnost rada je sekundarni faktor.

Rečnik implementiran sa drvetom (SortedDictionary <K, T>)

Implementacija strukture podataka „rečnika“ kroz crveno-crno stablo (klasa SortedDictionary <K, T>) je struktura koja omogućava skladištenje parova ključ / vrednost, u kojima su ključevi poređani (sortirani) po veličini. Struktura omogućava brzo izvršavanje osnovnih operacija (dodavanje elementa, pretraga po ključu i brisanje elementa). Složenost sa kojom se izvode ove operacije je logaritamska - O (log (N)). To znači 10 koraka za 1.000 elemenata i 20 koraka za 1.000.000 elemenata.

Za razliku od heš tabela, gde loša heš funkcija može dovesti do linearne složenosti pretraživanja i sabiranja, u strukturi SortedDictionary <K, T> broj koraka za izvođenje osnovnih operacija u proseku i u najgorem slučaju je isti - log2 (N). Uravnoteženo drveće nema heširanje, nema sudara i nema rizika od upotrebe loše heš funkcije.

Ponovo, kao i kod heš tabela, ključ se u strukturi može pojaviti najviše jednom. Ako želimo da stavimo nekoliko vrednosti pod isti ključ, moramo da koristimo listu kao vrednost za elemente, na primer List <T>.
SortedDictionary <K, T> svoje elemente čuva u izbalansiranom stablu crveno-crne boje, poređane po tasteru. To znači da ćemo, ako pretražimo strukturu (kroz njen iterator ili kroz foreach petlju u C #), dobiti elemente sortirane u rastućem redosledu prema njihovom ključu. Ponekad ovo može biti od velike pomoći.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Koristite SortedDictionary <K, T> u slučajevima kada je potrebna struktura u koju možete brzo dodavati, brzo pretraživati i trebati dohvatiti elemente sortirane u rastućem redosledu. Generalno, Dictionari <K, T> radi malo brže od SortedDictionary <K, T> i poželjniji je.

Kao primer korišćenja SortedDictionary <K, T> možemo dati sledeći zadatak: pronaći sve reči u tekstualnoj datoteci koje se javljaju tačno 10 puta i štampati po abecednom redu. Ovo je zadatak koji takođe možemo uspešno rešiti pomoću Dictionary <K, T>, ali moraćemo da izvršimo još jedno sortiranje. U rešavanju ovog problema možemo koristiti SortedDictionary <string, int> i proći kroz sve reči u tekstualnoj datoteci i za svaku od njih sačuvati u sortiranom rečniku koliko se puta u datoteci pojavljuje. Tada možemo proći kroz sve elemente rečnika i odštampati one od njih u kojima je broj podudaranja tačno 10. Oni će biti raspoređeni po abecedi, kao u prirodnom unutrašnjem redosledu sortiranog rečnika.

Koristite SortedDictionary <K, T> kada želite brzo da dodate elemente i pretražite po ključu, a zatim će vam biti potrebni elementi sortirani po ključu.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Skup realizovan heš tabelom (HashSet <T>)

Struktura „skupa podataka“ je skup elemenata, među kojima nema ponavljajućih. Glavne operacije su dodavanje elementa u skup, provera da li element pripada skupu (pretraga) i uklanjanje elementa iz skupa (brisanje). Operacija pretraživanja indeksa nije podržana, tj. Nemamo direktan pristup elementima prema broju naloga, jer u ovoj strukturi nema brojeva naloga.

Skup koji implementira heš tabela (klasa HashSet <T>) je poseban slučaj heš tabele u kojoj imamo samo ključeve, a vrednosti sačuvane ispod svakog ključa su irelevantne. Ova klasa je uključena u .NET Framevork tek od verzije 3.5.
Kao i kod heš tabele, glavne operacije u strukturi podataka HashSet izvode se sa konstantnom složenošću O (1). Kao i kod heš tabele, nepovoljna heš funkcija može dovesti do linearne složenosti osnovnih operacija, ali u praksi se to gotovo ne dešava.

Kao primer korišćenja HashSet <T> možemo odrediti zadatak pronalaženja svih različitih reči u tekstualnoj datoteci.

Koristite HashSet <T> kada trebate brzo dodati elemente u skup i proveriti da li je element iz skupa.

Set implementiran sa drvetom (SortedSet <T>)
Skup implementiran sa crveno-crnim stablom je poseban slučaj SortedDictionari <K, T>, u kojem se ključevi i vrednosti podudaraju.

Kao i kod strukture SortedDictionari <K, T>, osnovne operacije u SortedSet <T> se izvode sa logaritamskom složenošću O (log (N)), a ta složenost je ista u proseku i u najgorem slučaju.

Kao primer korišćenja SortedSet <T> možemo odrediti zadatak pronalaženja svih različitih reči u tekstualnoj datoteci i njihovog štampanja po abecednom redu.
Koristite SortedSet <T> kada trebate brzo dodati elemente u skup i proveriti da li je element u skupu, a biće vam potrebni i elementi sortirani u rastućem redosledu.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

7. Objekat orijentisano programiranje (OOP)

What is OOP?

OOP je skraćenica za Objekt orijentisano programiranje

Objektno orijentisano programiranje (OOP) je paradigma u računarskom programiranju, n koja se softverski sistem modelira kao skup objekata koji međusobno komuniciraju, za razliku od tradicionalnog pogleda, u kojem je program lista uputstava koje računar izvršava. Svaki objekat je u mogućnosti da prima poruke, obrađuje podatke i šalje poruke drugim objektima.

Objektno orijentisano programiranje daje veću fleksibilnost, što olakšava promenu programa. Široko je popularan u softverskom inženjerstvu za velike projekte. OOP je lakše naučiti od programera početnika, za razliku od ranijih pristupa i metodologija, a OOP pristup je jednostavniji za razvoj i održavanje.

Objektno orijentisano programiranje koristi sledeće koncepte:

• Objekti - drže podatke (polja) i funkcionalnost zajedno u odvojenim celinama u jednom računarskom programu; objekti služe kao osnova modularnosti i strukture u objektno orijentisanom računarskom programu. Predmeti su samostalne jedinice i trebalo bi ih lako prepoznati. Modularnost omogućava da delovi programa odgovaraju pojedinačnim aspektima problema.
• Apstrakcija - Sposobnost programa da ignoriše neke aspekte informacija sa kojima radi - sposobnost da se usredsredi na bitno. Svaki objekat u sistemu služi kao model apstraktnog „agenta“ koji može da obavlja posao, menja svoj status i izveštava o njemu i „komunicira“ sa drugim objektima u sistemu ne otkrivajući kako se ta svojstva ostvaruju.
• Kapsulacija - takođe se naziva „skrivanje informacija“: onemogućava korisnicima predmeta da promene svoje unutrašnje stanje na neočekivan način; samo unutrašnje metode objekta imaju pristup njegovom stanju.

• Polimorfizam - Različite stvari ili objekti mogu imati isti interfejs ili odgovarati na istu (po imenu) poruku i odgovarati na odgovarajući način, u zavisnosti od prirode ili vrste predmeta. Ovo omogućava da mnoge različite stvari budu zamenljive. Dakle, promenljiva u tekstu programa može da sadrži različite objekte tokom izvršavanja programa i da poziva različite metode u različito vreme izvršenja.
• Nasleđivanje - organizuje i podržava polimorfizam i inkapsulaciju, omogućavajući da se definišu i kreiraju objekti koji su specijalizovane varijante postojećih objekata. Novi objekti mogu da koriste (i proširuju) već definisano ponašanje, a da to ponašanje ne moraju ponovo da primene.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Ovo je zapravo niz ideja, od kojih većina postoji već duže vreme. Oni su okupljeni, zajedno sa srodnom terminologijom, kako bi stvorili metodologiju programiranja. Kaže se da su ideje koje stoje iza objektno orijentisanog programiranja toliko jake da su stvorile promenu paradigme u programiranju.

Njihove tačne definicije variraju u zavisnosti od tačke gledišta. Na primer, jezici statičkog tipa često imaju nešto drugačiji pogled na objektno orijentisano programiranje od jezika dinamičkog tipa, uzrokovani fokusiranjem na svojstva programa prilikom kompajliranja, u prvom slučaju, i prilikom izvršavanja u drugom.

Napomene: Apstrakcija je važna, ali nije jedinstvena za objektno orijentisano programiranje. Ponovna upotrebljivost je prednost koja se često pripisuje objektno orijentisanom programiranju.

Objektno orijentisano programiranje često se naziva „paradigmom“, a ne stilom ili vrstom programiranja da bi se naglasilo razumevanje da OOP može promeniti način na koji se softver razvija, promenom načina na koji programeri i softverski inženjeri misle za softver.

OOP paradigma u osnovi nije programska paradigma, već paradigma dizajna. Sistem je dizajniran definisanjem objekata koji će u njemu postojati, a kod koji zapravo obavlja posao nema veze sa objektom ili ljudima koji taj objekat koriste zbog enkapsulacije.

Stoga je izazov u OOP-u dizajn dobro promišljenog sistema predmeta.

Proceduralno programiranje znači pisanje procedura ili funkcija koje izvode operacije nad podacima, dok se objektno orijentisano programiranje odnosi na stvaranje objekata koji sadrže i podatke i funkcije.

Objektno orijentisano programiranje ima nekoliko prednosti u odnosu na proceduralno programiranje:

• OOP je brži i lakši za primenu
• OOP pruža jasnu strukturu programa
• OOP vam omogućava da kreirate kompletne aplikacije za višekratnu upotrebu sa manje koda i kraćim vremenom razvoja

Šta su klase i objekti?

Klase i objekti su dva glavna aspekta objektno orijentisanog programiranja.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pogledajte sledeću ilustraciju da biste videli razliku između klase i objekata:

KLASA	OBJEKTI
voće	jabuka
	banana
	mango

Još jedan primer:

KLASA	OBJEKTI
Automobil	Volvo
	Audi
	Toyota

Dakle, klasa je predložak za objekte, a objekat je instanca klase.

Kada se kreiraju pojedinačni objekti, oni nasleđuju sve promenljive i funkcije klase.

8. Razvoj velikih sistema

Dva su glavna koraka zajednička za razvoj svih računarskih programa, bez obzira na veličinu ili složenost. Prvi korak je analiza, a zatim korak kodiranja. Ova vrsta vrlo jednostavnog koncepta implementacije zapravo je sve što je potrebno ako je napor dovoljno mali i ako konačnim proizvodom moraju upravljati oni koji su ga napravili - kao što se obično radi sa računarskim programima za internu upotrebu. Ali kada su u pitanju veliki sistemi, planiranje primene usmereno samo na ove korake, međutim, osuđeno je na neuspeh.

Specijalizovani softverski sistemi su kompletni krajnji proizvodi koji nude kompletnu uslugu u određenoj oblasti. Razvoj složenog softverskog sistema zahteva u njega ugrađivanje određenog broja specijalizovanih algoritama. Postizanjem određenog nivoa složenosti prelazimo na modularni razvoj sistema - podeljen je na funkcionalne module koji mogu samostalno da rade i razvoj se izvodi modul po modul. Da bi se smanjio rizik od neuspeha, veći projekti zahtevaju više od osnovnih koraka - posebno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

korake za definisanje zahteva pre koraka analize; korak dizajniranja između faze analize i kodiranja; i korak testiranja nakon kodiranja. Takođe, kao što će vam reći neko ko je upoznat sa iterativnim tehnikama razvoja, ako se iterativno razvijamo, sa svakom iterativnom nadogradnjom rezultata svog prethodnika, možemo dalje smanjiti rizik, jer na kraju svake iteracije pomerimo svoju glavnu razvojnu liniju napred, tj. što se više razvijamo, to je manji rizik sa kojim se moramo suočiti.

Potrebno je mnogo dodatnih koraka za razvoj velikih sistema. Faze analize i kodiranja su i dalje na dnevnom redu, ali njima prethode dva nivoa analize zahteva, odvojena fazom dizajna programa i nakon toga fazom testiranja. Raspored koraka zasnovan je na sledećem konceptu: kako svaki korak napreduje i dizajn postaje detaljniji, ponavlja se sa prethodnim i sledećim koracima, ali retko sa udaljenijim koracima u nizu.

Za rešavanje ovog rizika koristi se pet koraka:

Prvo, uvedite korak „preliminarnog dizajna programa“ između generisanja zahteva i analize kako biste odredili ograničenja za skladištenje, vreme i operativna ograničenja, i pročistite dizajn saradnjom sa analitičkim ulazom. Drugim rečima, stvorite zasnovanu arhitekturu kako biste minimalizovali arhitektonski značajne rizike. Ključni faktori za obezbeđivanje uspeha su sledeći:

1. Započnite postupak sa dizajnerima
2. Dizajn, definicija i distribucija načina obrade podataka
3. Napišite dokument za pregled tako da svi razumeju sistem.

Drugo, dokumentujte dizajn. Upravljanje softverom je nemoguće bez vrlo visokog nivoa dokumentacije. Na taj način se pružaju dokazi o potpunosti, kada se dokumentuje, dizajn postaje realan. Nizvodni procesi (razvoj, ispitivanje, operacije itd.) Zahtevaju jaku projektnu dokumentaciju da bi uspeli.

Treće, „Uradi dva puta“ - ovo je razvoj funkcionalnog prototipa koji simulira visoko rizične elemente sistema koji će se razviti; onda, nakon što se uverite da su rešeni elementi visokog rizika, nastavite da razvijate pravu stvar - verziju koja će biti isporučena kupcu.

Četvrto: planiranje, praćenje i kontrolni testovi. Testiranje moraju obaviti nezavisni testeri koji nisu dali svoj doprinos dizajnu na osnovu dokumentacije uspostavljene u ranijim fazama. Vizuelni kod skenira radi grešaka u kodu. Opet, ovo bi trebalo da uradi neko ko nije tako blizu koda kao programer.

Postoje različite metode za organizovanje ovih procesa u razvoju velikih sistema. Jedan je tradicionalni model vodopada, a drugi nudi fleksibilnije modele upravljanja i zove se Agile.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Model Vaterfall je jedna od najranijih metodologija razvijenih za izgradnju softverskih proizvoda. Ovaj model deli softverske procese u različite faze, od kojih svaka sledi određeni redosled u razvoju softvera. Te faze su:

• Specifikacija zahteva
• Dizajn softvera
• Implementacija i integracija
• Testiranje (ili validacija)
• Primena (ili instalacija)

• Podrška

Procesi u modelu vodopada su linearni i sekvencijalni. Svaka od faza u razvojnom procesu započinje tek kada je prethodna faza u potpunosti završena. U strogom skladu sa metodologijom, povratak na prethodnu fazu modifikacije proizvoda zbog promena zahteva nije dozvoljen.

Fleksibilne metodologije (agilne) za razvoj softvera su neformalna zbirka metodologija i tehnika za upravljanje projektima za razvoj softvera. Kao što i samo ime govori, fokus fleksibilnih metodologija je ideja da je razvoj softvera dinamičan proces u kojem dugoročno planiranje ima ograničenu efikasnost. Fleksibilne metodologije se posebno široko koriste u razvoju proizvoda, gde čestim prototipovanjem proizvođači imaju priliku da dobiju povratne informacije od kupaca i razvoj prilagode novim zahtevima.

Spisak najpopularnijih fleksibilnih metodologija:

• Scrum
• Ekstremno programiranje
• Kanban ili Lean
• Kristal
• Metoda razvoja dinamičkih sistema (DSDM)
• Razvoj vođen karakteristikama (FDD)

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

9. Baze podataka za razvoj sistema

Baza podataka je zbirka podataka. Organizacija i raspored podataka mogu slediti određeni obrazac. Prema modelu mogu se opisati dve vrste baza podataka - relacione i nerelacione.

Do sada je najpoznatija i najkorišćenija vrsta baze podataka relaciona (SKL baza podataka). Drugi tip baze podataka koji koristi nerelacioni model podataka naziva se NoSKL (ne SKL / ne samo SKL).

9.1. Relacione baze podataka (SKL)

Relacione baze podataka, takođe poznate i kao „SKL baze podataka“, zbog jezika upita SKL, čuvaju podatke na unapred strukturiran način - u tabelama raspoređenim u redove (zapise) i kolone. Između pojedinačnih podataka i tabela mogu se stvoriti odnosi (relacije). Tabele i odnosi između njih čine strukturu koja se može predstaviti kao šema.

9.1.1. Model skladištenja podataka

Pojedinačne jedinice podataka sadržane su u pojedinačnim poljima raspoređenim u redove u tabeli.

ID	post_content	post_title	post_status	post_type
1	<p>Welcome to WordPress)...	Hello Site!	publish	post
2	This is an example page. It's...	Sample Page	draft	page

Relaciona baza podataka (MiSKL) učitana preko HeidiSKL

Dodatne informacije o podacima beleže se kroz kolone, u dodatnim poljima uz red / zapis.

9.1.1. Korišćenje relacionih baza podataka (SKL)

Široko se koriste SKL baze podataka - za male količine informacija kao što su veb stranice sa dve stranice, za velike veb ili mobilne aplikacije, blogove, on-line prodavnice i još mnogo toga. Najpoznatiji gotovi sistemi za upravljanje sadržajem (CMS) podržavaju i koriste relacione baze podataka - VordPress, Joomla, Drupal, Magento i druge. Međutim, manje je onih koji podržavaju NoSKL baze podataka (kao što je Drupal).

9.1.2. Proširenje (skalabilnost) relacionih baza podataka (SKL)

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Vertikalno. Baza podataka se nalazi na jednom serveru. Da biste se proširili, možete povećati snagu i resurse ovog servera. Moguće je da se baza podataka SKL širi na više servera, ali implementacija je obično složena, zahteva resurse i oduzima puno vremena. A pošto ove baze podataka ne nude takvu funkcionalnost na prirodan način, biće potreban dalji razvoj kako bi različite hardverske tačke mogle imitirati rad jedne baze podataka na jednom serveru. Dodatni razvoj softvera biće potreban za upravljanje logikom i distribucijom zahteva za podacima između tačaka, kao i za preuzimanje i objedinjavanje podataka sa različitih servera.

9.2. Nerelacione baze podataka

NoSKL baze podataka su zajedničko ime za razne tehnologije baza podataka stvorene za savremene aplikacije i ogromnu količinu informacija sa kojima rade. NoSKL baze podataka rešavaju različita SKL ograničenja za:

• laka skalabilnost na klasterima servera (horizontalno skaliranje);
• podrška za različite tipove struktura podataka;
• upotreba u razvoju sa fleksibilnim metodologijama (agilni razvoj).
Neke baze podataka NoSKL možda nisu u potpunosti u skladu sa ACID-ovim modelom transakcije (Atomskost, Doslednost, Izolacija, Trajnost). ACID model je skup svojstava transakcija koje mogu osigurati integritet, potpunost, izolaciju i elastičnost transakcija baze podataka.

Neke NoSKL baze podataka možda takođe ne podržavaju operacije spajanja koje se koriste u relacionim bazama podataka. Umesto toga, koriste se različiti pristupi kao zamena za obradu povezanih podataka: nekoliko zahteva umesto jednog; keširanje, umnožavanje, denormalizacija i gnežđenje podataka.

9.2.1. Model skladištenja podataka

Nerelacione baze podataka (NoSKL) ne koriste šeme i tabele podataka. Prvi zapis podataka u primeru za VordPress bazu podataka i tabelu postova izgledao bi slično u NoSKL bazi podataka:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
{ ID: 1,
 title: "Hello Site!",
 content: "<p>Welcome to
 WordPress...", status:
 "publish",
 type: "post"
}
```

Na primer, informacije u MongoDB NoSKL bazi podataka čuvaju se u „dokumentima“ sličnim JSON-u. Podaci (nazvani dokumenti) u bazi podataka dokumenata mogu se organizovati u zbirku, koja je nešto poput SKL tabele. Model podataka je dinamičan i moguće je dodati nova polja podataka bez potrebe za redizajnom šeme / strukture baze podataka.

9.2.1. Korišćenje NoSKL baza podataka

NoSKL baze podataka uglavnom se koriste za aplikacije koje rade sa ogromnim količinama informacija i potrebom za velikom brzinom i stalnim i automatskim proširivanjem. Takve aplikacije mogu biti masivne veb aplikacije i mobilne aplikacije koje opslužuju milione korisnika. Pojava potrebe za ovom vrstom baze podataka pripisuje se kontinuirano rastućem obimu podataka u digitalnom svetu i razvoju Veb 2.0. Postoje NoSKL plaćeni programi, kao i oni otvorenog koda kao što su: MongoDB, CouchDB, Druid, Neo4j Communiti Edition, Redis, Memcache itd.

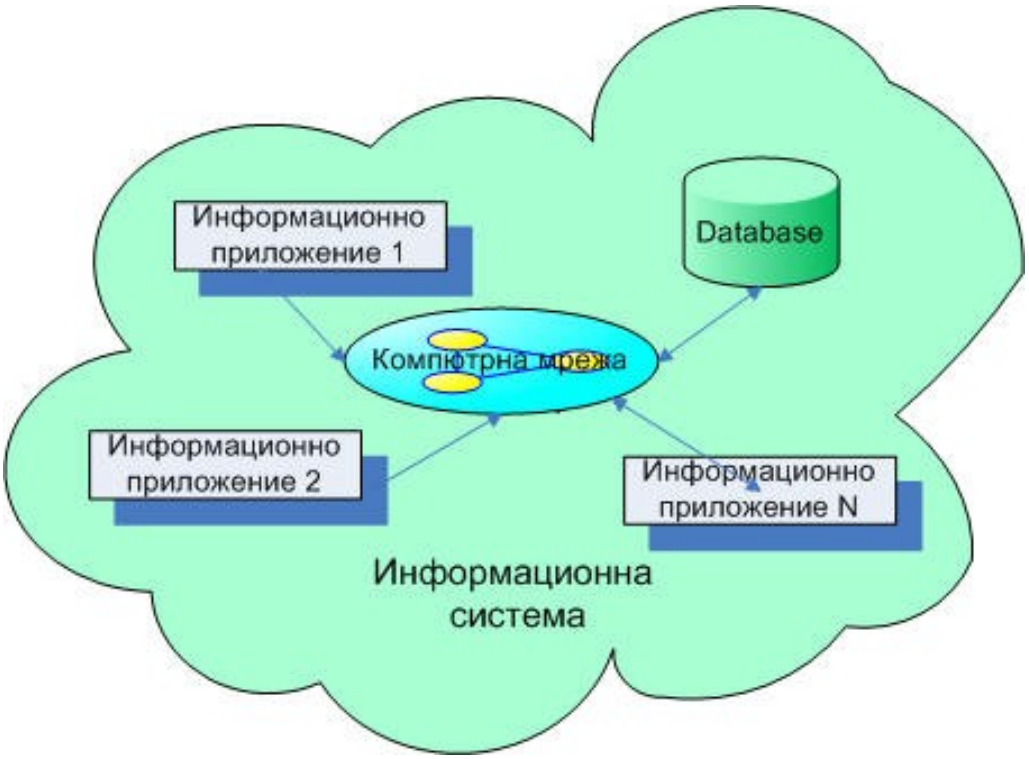
9.2.1. Proširenje (skalabilnost)

Horizontalno. NoSKL baze podataka imaju ugrađenu sposobnost arhitekture za automatsku distribuciju baze podataka na više servera, instance u oblaku i još mnogo toga. Distribuirana NoSKL baza podataka može se nositi sa opterećujućim zahtevima za obradu i ažuriranje velike količine informacija u realnom vremenu, uz visoku operativnu efikasnost.

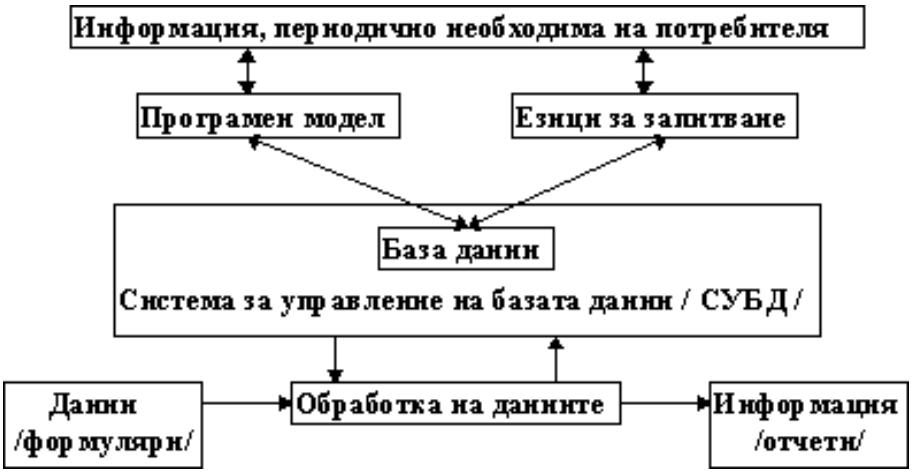
Upravljanje bazom podataka

Podaci su važan resurs organizacije koja zahteva pažljivo planiranje i upravljanje. Sa sve većom snagom računara, više nije problem za svakog korisnika da kreira sopstvenu bazu podataka. Danas su retki slučajevi primene informacija koje ne koriste baze podataka.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.



Model baze podataka



Sistem za upravljanje bazama podataka

Složeni softverski paketi koji podržavaju bazu podataka i pružaju interfejs sa korisnicima i korisničkim programima.

Glavne karakteristike DBMS-a:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

• Softver koji korisnicima i aplikacijama omogućava pristup bazi podataka.

• Korisnicima predstavlja logičke podatke. Detalji o tome kako i gde se podaci čuvaju skriveni su od korisnika.
• Nakon ažuriranja podataka, brine se o njihovoj usklađenosti.
• Pruža i kontroliše pravo pristupa korisnicima.

Pomagala u bazi podataka

Upitni jezici

Dizajnirani su za obične korisnike koji postavljaju upite u bazu podataka. Lako ih je razumeti i koristiti, a obično se sastoje od rečenica bliskih govornom engleskom.

Rečnici podataka

Skup podataka o samim podacima. Čuvajte informacije o:

• vrsta zapisa;
• nazivi i tipovi polja;
• ostale informacije o strukturi baze podataka.

Pomoćna sredstva za praćenje i procenu

Oni određuju stepen upotrebe podataka od strane pojedinačnih korisnika i distribuiraju troškove među njima.

Generatori izveštaja

Omogućavaju vam podešavanje različitih formata za izlazne podatke. Obično su moćni i jednostavni za upotrebu.

Vrste baza podataka

Hijerarhijski model baze podataka

• Podaci se čuvaju u unapred definisanoj hijerarhiji;
• Hijerarhijsko stablo je okrenuto naopako;

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

• Brzi pristup;
• Rezervne informacije se čuvaju

Model mrežne baze podataka

• Predmeti iz jednog predmetnog područja su objedinjeni u mrežu / skup /;
• Baza podataka sastoji se od nekoliko skupova, koji zauzvrat sadrže zapise;
• Skup zapisa može biti sadržan u jednoj ili više mreža;
• Brz, ali složen pristup podacima, koji zahteva dobro poznavanje logičke strukture podataka.

Model relacione baze podataka

• Podaci se čuvaju u tabelama
• Svaka tabela se sastoji od kolona / polja / i redova / zapisa /
• Pristup bazi podataka vrši se preko veza između pojedinih tabela
• Sporiji pristup, na račun čega je pojednostavljen rad sa DB-om;
• Pruža relativno visok stepen nezavisnosti podataka u poređenju sa drugim modelima.

Objektno orijentisane baze podataka

Oni se razlikuju od tradicionalnog relacionog modela i optimizovani su za skladištenje objekata. Mogu da čuvaju složene tipove nestrukturiranih podataka - glas, video, tekst i slike.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

10. Razvoj ugovornih sistema

Dizajn ugovora (DBC), takođe poznat kao programiranje ugovora, programiranje sa ugovorom i programiranje prilagođenog dizajna, pristup je softverskom dizajniranju.

Upućuje dizajnere softvera da definišu formalne, precizne i proverljive specifikacije interfejsa za softverske komponente koje proširuju uobičajenu definiciju apstraktnih tipova podataka sa preduslovima, postuslovima i invarijantama. Ove specifikacije se nazivaju „ugovori“, u skladu sa konceptualnom metaforom za uslove i obaveze poslovnih ugovora. DbC pristup pretpostavlja da će sve korisničke komponente koje pozivaju rad komponenti servera ispunjavati preduslove navedene u toj operaciji.

Centralna ideja DbC-a je metafora o tome kako elementi softverskog sistema međusobno komuniciraju na osnovu uzajamnih obaveza i koristi. Metafora dolazi iz poslovnog života, gde se „kupac“ i „dobavljač“ dogovaraju o „ugovoru“ koji precizira, na primer, da:

• Dobavljač mora da obezbedi određeni proizvod (obavezu) i ima pravo da očekuje da je kupac platio naknadu (naknadu).
• Kupac mora platiti naknadu (obavezu) i ima pravo da primi proizvod (pogodnost).
• Obe strane moraju ispuniti određene obaveze, poput zakona i propisa koji se primenjuju na sve ugovore.
Slično tome, ako metoda klase u objektno orijentisanom programiranju pruža neke funkcionalnosti, ona može:

• Očekuje se da će mu svaki korisnički modul koji ga zove zagarantovati određeni uslov za prijavu: preduslov metode - obaveza za kupca i korist za dobavljača (sam metod), jer ga oslobađa potrebe da rešava slučajeve izvan preduslova.
• Osigurajte određenu imovinu na izlazu: nakon uslova metode - obaveza za dobavljača i očigledna korist (glavna prednost pozivanja metode) za kupca.
• Održavajte određenu imovinu prihvaćenu na ulazu i zagarantovanu na izlasku.
Mnogi programski jezici imaju mogućnost da podnesu takve tvrdnje. Međutim, DbC smatra da su ovi ugovori toliko važni za ispravnost softvera da bi trebali biti deo procesa dizajniranja.

Dizajn prema ugovoru takođe određuje kriterijume za ispravnost softverskog modula:

• Ako je uslov za invarijantnu klasu i tačan pre nego što kupac pozove dobavljača, tada će invarijanta i postuslov biti tačni nakon završetka usluge.
• Kada pozivate dobavljača, softverski modul ne sme kršiti pretpostavke dobavljača.
Dizajn ugovora takođe može olakšati ponovnu upotrebu koda, jer je ugovor za svaki deo koda u potpunosti dokumentovan. Ugovori o modulima mogu se smatrati oblikom softverske dokumentacije za ponašanje ovog modula. Dizajn ugovora ne zamenjuje redovne strategije testiranja, kao što su jedinstveno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

testiranje, integracija i sistemsko testiranje. Umesto toga, dopunjuje spoljno testiranje unutrašnjim samotestiranjem, koje se mogu aktivirati i za izolovane testove i u proizvodnom kodu tokom faze ispitivanja.

11. Integracija sistema

Sistemska integracija je širok pojam i teško se može dati preciznija definicija od one sadržane u samom imenu - integracija podsistema za saradnju. Dakle, čak i pokretanje operativnog sistema na računaru može se pripisati ovoj aktivnosti, ali u praksi, kao specijalizacija sistemskih integratora kompanija, termin je počeo da se koristi pojavom mreža ličnih računara i strukture klijent-server. Komplikovanjem informacionih tehnologija i širenjem njihove upotrebe od strane različitih preduzeća, vladinih organizacija, sektora kao što su medicina i obrazovanje, istraživačke institucije itd., Integraciji sistema dodaju se sve složenije aktivnosti i zahtevi za profesionalizmom i mogućnostima kompanija koje rade u ovoj oblasti. Čak počinju da se pojavljuju specijalizacije, poput integratora telekomunikacione opreme, integratora bežične mreže, ali trend je više ka dodavanju novih aktivnosti i povećanju kompanija koje već posluju u tom sektoru.

Sa tehnološkog stanovišta, možda bi bilo prikladno reći da sistemski integratori moraju razviti infrastrukturu od informacionih i telekomunikacionih tehnologija do specifičnih aplikacija za određenu organizaciju, odakle započinje aktivnost kompanija u sektoru aplikativnog softvera. Ili sistemski integratori moraju da puste u rad sve hardverske uređaje sa odgovarajućim sistemskim softverom i međuopreme, sve mrežne i komunikacione funkcije, uključujući današnju upotrebu Interneta, e-pošte, mobilnih uređaja, bežičnih mreža itd.

Ovo područje integracije koristi se u svrhe ovog kataloga, a u širem smislu pojam se koristi i za integrisanje aplikacija kao što su ERP, CRM, modeliranje poslovnih procesa, kao i u mnogim drugim industrijama izvan informacione i telekomunikacione tehnologije.

Evo još nekih definicija:

• Sistemska integracija je integracija podsistema-komponenti u jedan sistem i osiguravanje njihovog rada kao jedan sistem.
• Sistemska integracija je definicija "lepka" između pojedinih komponenti.
• Sistemska integracija dodaje vrednost sistemu zbog interakcije između podsistema.

• Sistemska integracija obezbeđuje rad svih delova u celini.
• Sistemska integracija pruža kupcima ono što žele.

Najprikladnije je reći da je sve to. Glavni koraci uglavnom uključuju:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

• primanje informacija od kupca
• upravljanje projektima i projektni zadatak
• formulisanje zahteva
• dokumentovanje zahteva
• razvoj podsistema
• preliminarni testovi
• puna integracija
• zvanični testovi
• zvanična potvrda
• prihvatanje od strane kupca

U današnjem povezanom svetu uloga kompanija za sistemsku integraciju kao i inženjera u ovoj oblasti postaje sve važnija - sve više se dizajnira i razvija sistem koji treba da bude povezan zajedno.

Inženjeri sistemske integracije moraju da imaju širok spektar veština koje se teško mogu nabrojati - moraju znati softverski i hardverski inženjering, protokole interfejsa, veštine za rešavanje uobičajenih problema itd. Ponekad se kaže da sistemski integrator mora da zna malo o širokom spektru proizvoda, ali najvažnije je imati sposobnost brzog ispitivanja dostupnih proizvoda i biti dobar dijagnostičar.

Kada razvijate strategiju integracije, sledite ovih šest osnovnih koraka: KORAK 1: Otkrijte ko je vaše gravitaciono težište
Jedno od prvih pitanja koje treba razmotriti prilikom planiranja hibridne integracije u vašoj organizaciji je mesto na kojem ćete hostovati (primeniti) svoje alate i tehnologiju integracije. Odgovor na ovo pitanje leži u vašem „težištu“.

To znači da prvo morate uzeti u obzir lokaciju sistema koji trenutno imate, uključujući strateške sisteme za beleženje i čuvanje informacija, kao što su ERP i CRM. Zatim razmotrite kako planirate da proširite svoje aplikacije i projekte u narednih tri do pet godina kako biste utvrdili brzinu i složenost prelaska na oblak.

Hostiranje integracija u blizini vaših aplikacija i drugih izvora podataka važno je da biste izbegli usporavanje procesa povezivanja između aplikacija. Što je integracija bliža vašim aplikacijama, to bolje performanse možete očekivati od nje. Možete odabrati da hostete integraciju lokalno, u oblaku ili oboje.

Samo u oblaku: Ako vaša organizacija nema puno starih investicija u ovoj oblasti i zauzela je čvrst stav prema oblaku, možda ćete pronaći da je rešenje u potpunosti zasnovano na oblaku najbolje za vas.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Međutim, ovo nije tipičan scenario za većinu organizacija koje su godinama akumulirale bogatu IT infrastrukturu.

Lokalno: Za mnoge će najpraktičnija lokacija za integraciju i dalje biti lokalna, koja se nalazi zajedno sa sistemima za snimanje i skladištenje. Ovo je naročito tačno ako vaša organizacija trenutno ima samo nekoliko aplikacija zasnovanih na oblaku i ne planira da poveća njihov broj u bliskoj budućnosti. Ovaj izbor može biti najprikladniji ako ne možete lako premestiti aplikacije u oblak zbog postojećih propisa ili iz bezbednosnih razloga.

Hibrid iz oblaka i lokalni hosting: Za kompanije koje imaju više lokalnih aplikacija i stare investicije, ali trenutno primenjuju nove aplikacije i infrastrukturu uglavnom putem oblaka, izbor - podrazumevano - teži ka hibridnom integracionom okruženju sa nekim oblikom lokalne integracije rešenje, zajedno sa uslugom integracije hostovanom u oblaku; na primer, integraciona platforma kao usluga (iPaaS). Za većinu takvih kompanija planiranje integracionog rešenja zasnovanog na oblaku trebalo bi da bude prioritet.

Jednom kada vaša kompanija odluči gde će biti smeštena tehnologija za integraciju, sledeće pitanje koje treba da odlučite je koliko kontrolu želite da imate nad hostovanjem i upravljanjem sistemom.

KORAK 2: Odlučite koliko kontrole i odgovornosti želite da imate

U današnjem IT okruženju, kompanije imaju mnogo veću fleksibilnost nego prethodnih godina. Jedno od područja u kojima je ovo naročito tačno je koliko praktične kontrole i odgovornosti odlučuju da izvrše nad svojim sistemima. Svedoci smo povećanja kombinacije samoupravnih, izvezenih i hostovanih sistema.
Na odgovor na pitanje kontrole i odgovornosti često utiče, pre svega, motivacija vaše kompanije za primenu novih cloud usluga i kreće se od potpune kontrole nad celokupnom instalacijom, razvojem i operacijama, do potpunog prenosa ove odgovornosti na spoljne provajderi. Ispod ćete pronaći tri najčešća pristupa hibridnoj integraciji:

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pristup hibridnoj integraciji	**Opis**	**Razmatranja**
Privatna / javna integracija u oblak	Mnoge organizacije proširuju upotrebu internih privatnih tehnologija u oblaku, kao i javno dostupne platforme infrastrukture u oblaku kao što su Amazon EC2®, GoogleCompute Engine ™ Service i Microsoft AZURE ™, da bi ugostile svoje aplikacije i IT projekte. Donoseći ovu promenu, ove kompanije takođe prebacuju svoju strategiju strateške integracije u ista ista privatna okruženja u oblaku.	Prednost ove opcije je u tome što kompanije mogu brže primeniti svoje projekte i istovremeno dobiti kontrolu nad rastom kapaciteta po potrebi. Ova opcija daje najviše kontrole nad tehnologijama integracije. Međutim, glavni izazov sa ovim pristupom je da kompanije moraju da održavaju ove projekte kao da se nalaze lokalno, što znači da će trebati da vrše održavanje, upravljaju ažuriranjima itd. Ako vaša kompanija ima agresivnu strategiju za premeštanje procenta svoje IT infrastrukture u oblak, onda je ova opcija možda vrlo prikladna.
Hibridna integracija u oblak	Ovaj pristup uključuje upotrebu usluge integracije zasnovane na oblaku, kao što je iPaaS, za integraciju aplikacija zasnovanih na oblaku sa lokalnim tehnologijama integracije, kao što je Enterprise Service Bus (ESB). Ovaj pristup nudi stalnu kontrolu nad vašim integracijama i smanjuje vaše odgovornosti za održavanje i ažuriranja.	Ovaj pristup pruža dodatne pogodnosti nadogradnjom pristupa privatnog pristupa integraciji u oblak, jer je iPaaS dostupan širem krugu korisnika, može se koristiti bilo gde (jer je zasnovan na oblaku) i često pruža pojednostavljene alate. Takođe, ovaj pristup smanjuje troškove redizajniranja postojećih integracija poput usluga, povezivanja i transformacija. Dodatne pogodnosti su: ubrzana integracija i skalabilnost na osnovu broja transakcija koje prolaze kroz sistem.
Spoljno upravljanje integracijom u oblaku	Tehnologija integracije u oblaku kojom upravlja spoljna usluga omogućava prenos operacija kao što su instalacija, hosting i održavanje integracione tehnologije, omogućavajući tako spoljnom dobavljaču usluga da upravlja tehnologijom.	Prednost ovog pristupa je u tome što smanjuje količinu praktičnog upravljanja i operacija potrebnih vašoj organizaciji. Takođe vam omogućava da nastavite da koristite tehnologiju integracije koju trenutno koristite i u koju ste investirali, ali eliminiše potrebu za upravljanjem održavanjem, nadogradnjama i mnogim drugim svakodnevnim operacijama.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Kada utvrdite koliko kontrole želite da zadržite nad svojim integracionim okruženjem, morate razmotriti ko će biti korisnici usluge integracije.

KORAK 3: Upoznajte svoje kupce

Jedan od trendova koje smo videli na polju integracije poslednjih godina je da organizacije menjaju svoj pristup razvoju integracije. Izazovi koji pokreću ove promene dolazili su iz sve većeg broja aplikacija u oblaku koje poslovna odeljenja ili odeljenja često nabavljaju umesto IT-a, kao i iz potrebe za decentralizacijom odgovornosti za integraciju kako bi poslovna odeljenja stekla veću kontrolu nad svojim projektima. Zbog toga treba da odgovorite na pitanje: „Ko će ubuduće raditi na integraciji?“

U većini organizacija odgovor je kombinacija tradicionalnih uključenih integratora, neintegracionih programera i nove klase korisnika, često zvanih „građanski integratori“.

Privlačeni tradicionalni integratori: Bez obzira da li su deo kompanijskog centra za integracione kompetencije (ICC) ili su uključeni samo u integracioni projekat, tradicionalni uključeni integratori će i dalje igrati ključnu ulogu u integracionim projektima organizacija.

To je zato što su neki od ovih projekata presudni za organizaciju i zahtevaju specijalizovana znanja i veštine koje se ne mogu preneti na nestručnjake.

Na primer, neke kompanije neće želeti da rizikuju integritet podataka u svojim sistemima za beleženje informacija prenošenjem projekata na ruke neiskusnim programerima. Kompanijama će i dalje trebati stručnjaci za izradu dizajna i plana integracione arhitekture i osiguravanje da njihovi planovi integracije podataka ne oštete postojeću strukturu podataka kompanije.

Neintegrisani programeri: Nisu svi sistemi kritični. Na primer, primećujemo rast uvođenja sistema zasnovanih na oblaku od strane kupaca odeljenja koji traže sisteme koji se lako mogu primeniti i primeniti. U takvim slučajevima često je potrebno sinhronizovati podatke između sistema za beleženje informacija kako bi se osiguralo da su podaci tačni u svakom od sistema. Ova vrsta integracionih projekata, iako često manje složena, ponekad sadrži poverljive informacije.

U kontekstu sve većeg broja integracionih projekata, ljudski resursi u ovoj oblasti opadaju. Stoga se mnoga odeljenja sama bave integracijom. Na primer, sve je češće da odeljenja imaju svoje specijalizovane IT timove, koji se često nazivaju IT u senci. Mnoge kompanije žele da posao obavljaju nespecijalisti, jer su dobra prilično skupa. Cena koja se plaća za nespecijalizovane programere je mnogo niža od one za specijalizovane programere. Privlačni tradicionalni integratori poznaju pristupe i tehnologije za razvoj softvera, ali obično nemaju veštine tipične za privučene tradicionalne integratore. Veštine u polju integracionih tehnologija moraju se lako naučiti ako se koriste nespecijalizovani programeri.

„Građani integratori“ Konačno, kategorija korisnika koji su sve češći. Primećujemo porast broja poslovnih korisnika koji su zaduženi za održavanje i upravljanje aplikacijama zasnovanim na oblaku, uključujući integraciju podataka sa njima.

Ovi „građanski integratori“ obično nisu programeri IT-a. Umesto toga, oni mogu biti poslovni analitičari ili osoblje odeljenja odgovorno za primenu SaaS aplikacija u njemu. Za ove korisnike alat za integraciju treba da bude jednostavan za upotrebu, nudi jednostavniji interfejs i ne zahteva znanje o složenijim

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

arhitekturama i konceptima integracije.

Prenošenje integracionih projekata na korisnike, izvan osnovnog IT tima za integraciju, jedan je od načina da se više uradi uz manje ulaganja. Ali koje su vaše druge mogućnosti da se nosite sa zahtevima projekta?

KORAK 4: Planirajte kako se nositi sa zahtevima projekta

Kako se povećava broj SaaS aplikacija, tako se povećava i broj integracionih projekata za koje treba da obezbedite resurse. Uključeni timovi za integraciju teško mogu da zadovolje ovu potrebu i ponekad se čine kao usko grlo u primeni aplikacija. Zaostajanje visoko prioritetnih projekata integracije za odeljenja je neprihvatljivo za poslovanje.

S obzirom na ovaj trend, trebate razmotriti koje ćete pristupe udovoljavati rastućoj gladi za integracionim projektima, istovremeno ne ulažući dodatne resurse za razvoj integracionih resursa. Razmotrite sledeće pristupe:

Predložite integraciju „samoposluživanja“:

S obzirom na ograničenja koja privlači IT, druga opcija za razmišljanje o ulozi IT-a u integraciji je pružanje pružaoca usluga koji omogućava odeljenjima da sami obavljaju posao integracije. Ovaj pristup zahteva od IT-a da uspostavi zajedničku arhitekturu, usluge, pristup i alate za projekte integracije i učini ih dostupnim na korišćenje osoblju odeljenja. Ovim pristupom IT osigurava da je kompanija zaštićena od potencijalnih destruktivnih problema koji mogu nastati kao rezultat neiskusnih korisnika.

Povećajte ponovnu upotrebu imovine:

Jedan od najboljih načina za rešavanje rastuće potražnje za integracionim projektima je ograničavanje količine „novog posla“ koji treba obaviti. A najlakši način da se to postigne je efikasna upotreba postojećih sredstava za integraciju, poput usluga, povezivanja, transformacija i distribucije. Jedna opasnost, kako za tradicionalne IT integracione projekte, tako i za projekte koji stvaraju integraciju od internih odeljenja, u primeni višestrukih integracionih rešenja, jeste stvaranje spremišta integracionih sredstava. Kupci koji su u prošlosti usvojili najbolje prakse u ponovnoj upotrebi usluga, kao što su kompanije sa inicijativama orijentisane na arhitekturu (SOA), često su u dobrom položaju da izbegnu ponovnu integraciju. Ali čak i ako nemate SOA praksu, pokušajte da iskoristite sve svoje postojeće resurse, pod uslovom da ih mogu koristiti alati za integraciju zasnovani na oblaku.

Razmišljajući o tome kako se nositi sa projektnom potražnjom, trebali biste razmisliti i o tome kako obezbediti fleksibilnost za različite vrste integracionih projekata.

KORAK 5: Koristite fleksibilni pristup za različite vrste projekata

Da li jednostavna i nekritična integracija košta toliko koliko i složena, kritična integracija? Da li za sve svoje projekte zavisite od svojih stručnjaka za integraciju? Pogledajmo dva trenda koja utiču na to kako organizacije pristupaju projektima integracije.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Predložite više načina integracije

Nisu svi projekti integracije isti. Neki su kritični za organizaciju, a neki su hitni i mogu imati kratak život.

Stoga biste trebali razmotriti mogućnost pružanja alata za integraciju za jednostavnije i brže projekte. Analitičari i lideri u industriji često nazivaju ovaj trend u razvoju dvomodalnih IT aplikacija.

Razvojni mode 1	**Razvojni mode 2**
Tradicionalni, složeni integracioni projekti spadaju u kategoriju Razvojnog načina 1. Ovi tradicionalni projekti zahtevaju stabilnost i dobro planiranje, testiranje, upravljanje i pregled arhitekture. U prošlosti svi u ovom su obrađeni integracioni projekti način.	Alternativno, projekti integracije Mode 2 zahtevaju brz, fleksibilan razvoj i najčešće ne uključuju Razvojni Mode 2 je namenjen brzim i inovativnim projektima kojima se tipično upravlja poslovne potrebe. Ova vrsta projekata je gde „građanski integratori“ ili neintegracija programeri mogu biti uključeni u integraciju projekti.poslovno kritični sistemi.

Prihvatite API-je za internu integraciju

U nekim slučajevima može biti prihvatljivo videti API-je kao alat za integraciju aplikacija, koji nespecijalistima daju alate za integraciju aplikacija, ali eliminišući potrebu za specijalizovanim znanjem i veštinama. Većina IT programera oseća se ugodno uz API za stvaranje novih usluga i povezivanje sistema. Međutim, ovaj pristup takođe može da uključuje određeni stepen rizika u odsustvu pravih smernica, što dovodi do pojave tipa integracionih tabela „špageta“ koje već godinama muče IT odeljenja.

KORAK 6: Razmislite kako ćete osigurati integritet podataka za svoje sisteme za evidentiranje informacija

Prilikom uvođenja novih opcija integracije u svojoj kompaniji, razmislite kako bi ove promene uticale na kvalitet podataka i stabilnost projekta.

Promene u tome ko razvija projekat, kao i u pristupima stvaranju integracija, mogu biti rizične, posebno u pogledu načina na koji su vaši podaci zaštićeni od kompromisa izazvanih programerima koji nemaju potrebno znanje vašeg sistema za snimanje i čuvanje. podaci. Samo ovlašćeni korisnici treba da imaju pristup podacima u vašoj aplikaciji i treba da postoji odgovarajući nivo nadzora. Odgovarajuća pravila i kontrole su od suštinskog značaja za smanjenje rizika i poštovanje svih politika usklađenosti u vašoj organizaciji.

Može imati smisla, na primer, dokumentovati vaše integracione procese i najbolje prakse kako biste izbegli rizike koji se mogu pojaviti u vašem okruženju. Vaša organizacija će želeti da smanji rizik povezan sa uvođenjem mnogih novih integracionih projekata i činjenicom da će tim projektima upravljati profesionalci koji se ne integrišu.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

PRAKTIČNE VEŽBE I TESTOVI

Tematski ciklus 1 "Softversko testiranje"

Test pitanja

Pitanje 1. Koja je od sledećih tvrdnji netačna u kontekstu ispitivanja performansi?

1. Merenje vremena odziva.
2. Merenje procenta transakcija.
3. Test oporavka.
4. Simuliranje mnogih korisnika.

Pitanje 2. Koji od sledećih standarda određuje uslove ispitivanja?
1. ANSI / IEEE 829
2. BS7925-2
3. BS7925-1
4.ISO / IEC 12207
5.ANSI / IEEE 729

Pitanje 3. Koja od sledećih aktivnosti ne spada u sistemsko testiranje?

1. Ispitivanje performansi, opterećenja i naprezanja.
2. Ispitivanje upotrebljivosti
3. Testiranje zasnovano na poslovnom procesu.
4. Integraciono testiranje odozgo nadole.

Pitanje 4. Šta od navedenog NIJE tehnika crne kutije?

1. Razdvajanje ekvivalencije
2. Ispitivanje stanja tranzicije
3. Linearni niz kodova i preskočite
4. Testiranje sintakse
5. Analiza granične vrednosti

Pitanje 5. Koja se faza testiranja pojedinačnih softverskih modula kombinuje zajedno u grupu?

1. Ispitivanje modula
2. Integraciono testiranje
3. Testiranje bele kutije
4. Testiranje softvera

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 6. Šta od navedenog NIJE tehnika statičkog ispitivanja?

1. Pogreška u pogađanju
2. Proba
3. Analiza protoka podataka
4. Inspekcije

Pitanje 7. Verifikacija je:

1. Provera da li razvijamo pravi sistem
2. Provera da li sistem pravilno razvijamo
3. Izvodi ga nezavisni test tim
4. Da biste bili sigurni da je to ono što korisnik zaista želi

Pitanje 8. Beta testiranje je:

1. Izvode ga kupci na svom sajtu
2. Izvode ga kupci na mestu svog softvera
3. Izvodi nezavisni test tim
4. Korisno za testiranje naručenog softvera
5. Izvodi se što je ranije moguće u životnom ciklusu

Pitanje 9. Šta NIJE tačno - tester crne kutije:

1. treba da bude u stanju da razume dokument u vezi sa funkcionalnom specifikacijom ili zahtevima
2. mora biti u stanju da razume izvorni kod
3. je visoko motivisan za otkrivanje grešaka
4. kreativan je za otkrivanje sistemskih slabosti

Pitanje 10. Analiza uticaja pomaže u odluci:

1. Razni alati za izvođenje regresivnog ispitivanja
2. Kriterijumi za izlazak
3. Koliko još test slučajeva treba napisati
4. Koliko regresijskog testiranja treba obaviti

Pitanje 11. Razlika između ponovnog i regresivnog testiranja je:

1. Ponovo testiranje testova; regresijsko testiranje traži neočekivane neželjene efekte
2. Ponovno testiranje traži neočekivane neželjene efekte; regresijsko testiranje ponavlja ove testove
3. Ponovno testiranje se vrši nakon rešavanja problema; regresijsko ispitivanje se vrši ranije
4. Ponovno testiranje koristi različita okruženja, regresijsko testiranje koristi isto okruženje
5. Ponovno testiranje vrše programeri, a regresijsko testiranje rade nezavisni testeri

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 12. Utvrdite tvrdnju koja se primenjuje u slučaju istraživačkog ispitivanja:

1. Izvršenje započinje tek kada je dizajn finaliziran
2. Uključuje simultani dizajn i izvođenje testa
3. Izvršenje započinje tek kada se dizajn obnovi
4. Izvršenje započinje tek kada se dizajn promeni

Pitanje 13. Šta od navedenog nije deo testova performansi?

1. Merenje vremena odziva
2. Merenje procenta transakcija
3. Test oporavka
4. Simuliranje mnogih korisnika
5. Generisanje mnogih transakcija

Pitanje 14. Šta je od navedenog glavni zadatak planiranja testa?

1. Definisanje testnog pristupa
2. Priprema specifikacija testa
3. Procena izlaznih kriterijuma i izveštavanje
4. Merenje i analiza rezultata

Pitanje 15. U kojoj aktivnosti glavnog procesa testiranja se stvara test okruženje?

1. Implementacija i izvršenje testa.
2. Planiranje i kontrola testa
3. Analiza i dizajn testa
4. Procena izlaznih kriterijuma i izveštavanje

Pitanje 16. Koji je skup aktivnosti kojim se osigurava da softver pravilno obavlja određenu funkciju.

1. Verifikacija
2. Testiranje
3. Izvršenje
4. Validacija

Pitanje 17. Verifikacija se zasniva na računaru.

1. Tačno
2. Netačno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 18. se radi u fazi razvoja otklanjanja grešaka.

1. Kodiranje
2. Testiranje
3. Rešavanje problema
4. Izvršenje

Pitanje 19. Pronalaženje ili identifikovanje grešaka poznato je kao_____

1. Dizajn
2. Testiranje
3. Rešavanje problema
4. Kodiranje

Pitanje 20. Šta određuje ulogu softvera?

1. Dizajn sistema
2. Dizajn
3. Sistemski inženjering
4. Implementacija

Pitanje 21. Kako nazivate testiranje pojedinih komponenata?

1. Testiranje sistema
2. Jedinstveno testiranje
3. Provera valjanosti
4. Testiranje u crnoj kutiji

Pitanje 22. Strategija testiranja koja testira aplikaciju u celini je:

1. Zahtev za naplatu
2. Verifikaciono testiranje
3. Provera valjanosti
4. Testiranje sistema

Pitanje 23. je testirano kako bi se osiguralo da informacije pravilno ulaze u sistem i izvan njega.

1. modularni interfejs
2. lokalna struktura podataka
3. granični uslovi
4. staze

Pitanje 24. Svrha faze zahteva je

1. da zamrzne zahteve
2. da razume potrebe korisnika
3. da se utvrdi obim ispitivanja
4. Sve navedeno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 25. Koja od sledećih tvrdnji o ispitivanju komponenata nije tačna?

1. Ispitivanje komponenata mora se izvršiti razvojem
2. Ispitivanje komponenata je takođe poznato kao izolacija ili modularno ispitivanje
3. Testiranje komponenata trebalo bi da ima planirane kriterijume završetka
4. Testiranje komponenata ne uključuje regresijsko ispitivanje

Pitanje 26. Tokom koje probne aktivnosti mogu se naći najviše greške u troškovima?

1. Izvršenje
2. Dizajn
3. Planiranje
4. Verifikacija kriterijuma učinka

Pitanje 27. Koji procenat troškova razvoja softvera se uzima u obzir prilikom testiranja softvera?

1. 10-20%
2. 40-50%
3. 70-80%
4. 5-10%

Pitanje 28. ispitivanjem se osigurava da se posle promene ne pojave novi nedostaci u kodu
1. Ponovno testiranje
2. Regresijsko testiranje
3. I odgovor 1 i odgovor 2
4. Ništa od navedenog

Pitanje 29. Koje su svrhe testiranja

1. Identifikacija ranih nedostataka
2. Sticanje samopouzdanja
3. Prevencija defekata
4. Sve navedeno

Pitanje 30. U kom modelu faza ispitivanja započinje nakon faze razvoja

1. V model
2. Agilan model
3. Model prototipa
4. Model vodopada

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tematski ciklus 2: "Razvoj softvera"

Radne vežbe

Zadatak 1. Umetnite deo donjeg koda koji nedostaje za prikaz "Hello World!".

```
int main() {
 ____ << "Hello World!";
 return 0;
}
```

Zadatak 2. Umetnite novi red posle "Hello World" koristeći specijalni simbol :

```
int main() {
 cout << "Hello World! ____";
 cout << "I am learning C ++";
 return 0;
}
```

Zadatak 3. Komentari na jeziku C ++ napisani su posebnim simbolima. Umetnite delove koji nedostaju:

```
____ This is a single-line comment
____ This is a multi-line comment ____
```

Zadatak 4. Kreiraj varijablu naziva myNum i dodeli joj vrednost 50 .

```
____ ____ = ____;
```

Zadatak 5. Prikazuje zbir 5 + 10 pomoću dve promenljive: x i y

```
____ ____ = ____;
int y = 10;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
cout << x + y;
```

Zadatak 6. Napravite promenljivu zvanu z, dodelite joj x i y i prikažite rezultat.

```
int x = 5;
int y = 10;
      =  x + y;
cout <<      ;
```

Zadatak 7. Popunite delove koji nedostaju da biste kreirali tri promenljive istog tipa pomoću liste odvojene zarezima:

```
     x = 5      y = 6      z = 50;
cout << x + y + z;
```

Zadatak 8. Koristite tačnu ključnu reč da biste uneli unos korisnika u promenljivu x:

```
int x;
cout << "Type a number: ";
     >>      ;
```

Zadatak 9. Popunite delove koji nedostaju da biste odštampali zbir dva broja (koji je postavio korisnik):

```
int x, y;
int sum;
cout << "Type a number: ";
     >>      ;
cout << "Type another number: ";
     >>      ;
sum = x + y;
```

Zadatak 10. Pomnoži 10 sa 5 i ispiši rezultat.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
cout << 10      5;
```

Zadatak 11. Podeliti 110 sa 5 i ispisati rezultat.

```
cout << 10      5;
```

Zadatak 12. Koristite tačan operator da biste vrednost promenljive x povećali za 1.

```
int x = 10;
    x;
```

Zadatak 13. Pomoću operatora dodavanja dodeljivanja dodajte promenljivoj x vrednost 5.

```
int x = 10;
x       5;
```

Zadatak 14. Koristite ispravnu funkciju za ispis najveće vrednosti x i y

```
int x = 5;
int y = 10;
cout <<      (x, y);
```

Zadatak 15. Koristite ispravnu funkciju za štampanje kvadratnog korena x.

```
int main() {
 int x = 64;
 cout <<
 return 0;      (x);
}



#include <iostream>
#include <      >
using namespace std;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 16. Koristite tačnu funkciju da broj 2.6 zaokružite na najbliži celi broj.

```
#include <iostream>
#include <      >
using namespace std;

int main() {
 cout <<       (2.6);
 return 0;
}
```

Zadatak 17. Popunite delove koji nedostaju da biste odštampali vrednosti 1 (za tačno) i 0 (za netačno):

```
      isCodingFun = true;
      isFishTasty = false;

cout <<       ;
cout <<       ;
```

Zadatak 18. Popunite delove koji nedostaju da biste ispisali vrednost true (za true):

```
int x = 10;
int y = 9;
cout << (       >       );
```

Zadatak 19. Dodajte tačan tip podataka za sledeće promenljive:

```
      myNum = 9;
      myDoubleNum = 8.99;
      myLetter = 'A';
      myBool = false;
      myText = "Hello World";
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 20. Napravite dve logičke promenljive zvane da i ne i dodajte im odgovarajuće vrednosti:

```
        =  ;
        =  ;
```

Zadatak 21. Napravite pozdravnu promenljivu i pokažite njenu vrednost :

```
        = "Hello";
cout <<  ;
```

Zadatak 22. Ištampaj"Hello World" ako je x veće od y

```
int x = 50;
int y = 10;
     (x       y) {
 cout << "Hello World";
}
```

Zadatak 23. Print "Hello World" ako je x jednako y

```
int x = 50;
int y = 50;
     (x       y) {
 cout << "Hello World";
}
```

Zadatak 24. Ištampaj"Yes ako je x jednako y, u suprotnom ištampaj "No".

```
int x = 50;
int y = 50;
     (x       y) {
 cout << "Yes";
}     {
 cout << "No";
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 25.Ištampaj "1" ako je x jednako y, ištampaj "2" ako je x veće od y, u suprotnom ištampaj 3".

```
int x = 50;
int y = 50;
      (x         y) {
 cout << "1";
}         (x         y) {
 cout << "2";
}       {
 cout << "3";
}
```

Zadatak 26. Umetnite delove koji nedostaju da biste popunili sledeću „kratku izjavu ako ... inače“ (operater terminala):

```
int ti e = 20;
string result =       time < 18          "Good day."
       "Good evening.";
cout << result;
```

Zadatak 27. Unesite delove koji nedostaju da dopunite switch izjavu.

```
int day = 2;
switch (       ) {
       1:
 cout << "Saturday";
 break;
       2:
 cout << "Sunday";
       ;
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 28. Dopunite switch izjavu i dodajte odgovarajuću ključnu reč na kraju da biste definisali kod u switch izrazu, nema slučajnih pogodaka .

```
int day = 4;
switch (      ) {
 ______ 1:
 cout << "Saturday";
 break;
 ______ 2:
 cout << "Sunday";
 ______;
 ______:
 cout << "Weekend";
}
```

Zadatak 29. Napravite varijabilnu promenljivu pod nazivom Obrok, koja treba da bude referenca promenljive Hrana

```
string food = "Pizza";
string &      =      ;
```

Zadatak 30. Dobiti memorijsku adresu promenljive Hrane:

```
string food = "Pizza";
cout << &      ;
```

Zadatak 31. Napravite promenljivu pokazivača nazvanu pizza, koja mora da upućuje na nizu promenljivih sa imenom hrane :

```
string food = "Pizza";
      ____    = &      ;
```

Zadatak 32. Napravite funkciju pod nazivom myFunction i nazovite je unutar main().

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
void        () {
 cout << "I just got executed!";
}

int main() {
        ;
 return 0;
}
```

Zadatak 33. Umetnite deo koji nedostaje da biste dva puta pozvali myFunction

```
void myFunction () {
 cout << "I just got executed!";
}

int main() {
        ;
        ;
 return 0;
}
```

Zadatak 34. Dodajte fname parametar tipa string u myFunction.

```
void myFunction(        ) {
 cout << fname << "Doe";
}

int main() {
 myFunction ("John");
 return 0;
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 35. Unesite deo koji nedostaje da biste ištampali broj 8 main, као користећи specifičnu ključnu reč unutar myFunction:

```
int myFunction (int x) {
  _____ 5 + x;
}

int main() {
 cout << myFunction (3);
 return 0;
}
```

Zadatak 36. Kreiraj niz tipa string naziva cars.

```
_____ _____[4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Zadatak 37. Ištampaj vrednost drugog elementa u cars nizu.

```
_____ _____[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << _____;
```

Zadatak 38. Promenite vrednost iz "Volvo" u "Opel"u cars nizu.

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
_____ = _____;
cout << cars [0];
```

Zadatak 39. Proverite elemente cars niza.

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
_____ (_____ = 0; _____ < 4; _____ ) {
 cout << _____ << "\n";
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 40. Popunite deo koji nedostaje da biste kreirali pozdravnu promenljivu tipa string i dodelili joj vrednost Zdravo.

```
      =      ;
```

Zadatak 41. Koristite tačan operator za povezivanje dva niza:

```
string firstName = "John";
string lastName = "Doe";
cout << firstName      lastName;
```

Zadatak 42. Koristite ispravnu funkciju za ispis dužine tekstualnog niza.

```
string txt = "Hello";
cout <<      .      ;
```

Zadatak 43. Pristupite prvom znaku (H) myString i odštampajte rezultat:

```
string myString = "Hello";
cout <<      ;
```

Zadatak 44. Promenite prvi karakter (H) u myString „J“:

```
string myString = "Hello";
                  ;
cout << myString;
```

Zadatak 45. Koristite ispravnu funkciju za čitanje reda teksta koji je stavio korisnik

```
stri g fullName;
cout << "Type your full name: ";
     (cin, fullName);
cout << "Your name is: " << fullName;
```

Zadatak 46. Ištampajte I ako je I manje od 6.

```
int i = 1;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
      (i < 6) {
 cout << and << "\ n";
       ;
}
```

Zadatak 47. Koristite do / while outline za štampu sve dok je rezultat manji od 6.

```
int i = 1;
      {
 cout << and << "\ n";
       ;
}
      (i <6);
```

Zadatak 48. Koristite za konturu da biste 5 puta odštampali „Da“ :

```
      (int i = 0; i < 5;       ) {
 cout <<       << "\n";
}
```

Zadatak 49. Zaustavite outline ako je i 5.

```
for (int i = 0; i < 10; i++) {
 if (i == 5) {
       ;
 }
 cout << and << "\ n";
}
```

Zadatak 50. U sledećoj petlji, kada je vrednost „4“, pređite direktno na sledeću vrednost

```
for (int i = 0; i < 10; i++) {
 if (i == 4) {
       ;
 }
 cout << and << "\ n";
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Test pitanja

Pitanje 1. Model koji demonstrira implementaciju sistema je:
1. model vodopada
2. prototip
3. inkrementalni model
4. fleksibilni model

Pitanje 2. Održavanje je poslednja faza u modelu vodopada
1. Tačno
2. Netačno

Pitanje 3. Faza u kojoj su pojedinačne komponente integrisane i osigurano je da bez grešaka zadovolje zahteve kupaca
1. Kodiranje
2. Testiranje
3. Dizajn
4. Izvršenje

Pitanje 4. _____je korak u kojem se dizajn pretvara u mašinsko čitljivu formu
1. Dizajn
2. Transformacija
3. Rešavanje problema
4. Kodiranje

Pitanje 5. Zahtevi kupca podeljeni su u logičke module kako bi se olakšalo

1. Nasleđivanje
2. Dizajn
3. Uređivanje
4. Implementacija

Pitanje 6. Kako nazivate tehničku osobu koja je u stanju da razume osnovne zahteve?
1. Vođa tima
2. Analitičar
3. inženjer
4. Akteri

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

7. Korak u modelu vodopada, koji uključuje sastanak sa kupcem radi razumevanja zahteva.

1. Zahtev za naplatu
2. SRS
3. Izvršenje
4. Pregled kupaca

Pitanje 8. Metodologija u kojoj su procesi upravljanja projektom bili korak po korak.

1. Uzlazno
2. Vodopad
3. Spirala
4. Prototip

Pitanje 9. Pojedinac koji planira i upravlja radom.

1. Interesna grupa
2. Rukovodilac projekta
3. Vođa tima
4. Programer

Pitanje 10. Planirani program ako se završi posao koji zahteva konačno vreme, napor i planiranje

1. Problem
2. Projekat
3. Proces
4. Program

Pitanje 11. Prikupljanje povezanih podataka je:

1. Informacije
2. Dragocene informacije
3. Baza podataka
4. Metapodaci

Pitanje 12. DBMS je softver.

1. Tačno
2. Netačno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 13. DBMS upravlja interakcijom između baze podataka.

1. Korisnici
2. Kupci
3. Krajnji korisnici
4. Akteri

Pitanje 14. Šta od navedenog nije uključeno u DBMS?

1. Krajnji korisnici
2. Podaci
3. Zahtev za prijavu
4. HTML

Pitanje 15. Baza podataka je obično

1. Sistematizovano
2. Orijentisan na korisnika
3. Orijentisan na kompaniju
4. Orijentisani prema podacima

Pitanje 16. Karakteristika jedinice je:

1. Stav
2. Atribut
3. Parametar
4. Ograničenja

Pitanje 17. Šta je IMS?

1. Sistem učenja informacija
2. Sistem upravljanja uputstvom
3. Sistem za manipulaciju uputstvima
4. Sistem upravljanja informacijama

Pitanje 18. Model koji su Hammer i Mc Leod razvili 1981. godine je:

1. SDM
2. OODBM
3. DDM

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

4. RDM

Pitanje 19: Objekat=_____+odnosi

1. podaci
2. atributi
3. subjekt
4. ograničenja

Ptanje 20: DBMS je set _____za pristup podacima

1. Kodovi
2. Programi
3. Informacije
4. Metapodaci

Pitanje 21. DBMS pruža ugodno i efikasno okruženje.

1. Tačno
2. Netačno

Pitanje 22. Šta od navedenog nije nivo apstrakcije?

1. fizički
2. logičan
3. nivo korisnika
4. pogled

Pitanje 23. Nivo koji opisuje kako se zapis čuva.

1. fizički
2. logičan
3. nivo korisnika
4. pogled

Pitanje 24. Nivo pomaže aplikativnim programima da sakriju detalje o tipu
1. fizički
2. logičan
3. nivo korisnika
4. pogled

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 25. Logička struktura baze podataka.
1. Šema
2. Atribut
3. Parametar
4. Poseban slučaj

Pitanje 26. Stvarni sadržaj baze podataka u datom trenutku.

1. Šema
2. Atribut
3. Parametar
4. Poseban slučaj

Pitanje 27. Šta od navedenog nije objektivni logički model?

1. ER
2. Mreža
3. Semantički
4. Funkcionalni prikaz

Pitanje 28. SQL je

1. Relacioni model
2. Mreža
3. IMS
4. Hijerarhijski pogled

Pitanje 29. Nivo koji opisuje podatke uskladištene u bazi podataka i odnose između podataka.
1. fizički
2. logičan
3. nivo korisnika
4. pogled

Pitanje 30. Svaki algoritam je program.

1. Tačno
2. Pogrešno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

ODGOVORI
Tematski ciklus 1 „Testiranje softvera“
Odgovori na test pitanja

Pitanje 1. Koja je od sledećih tvrdnji netačna u kontekstu ispitivanja performansi?

1. Test oporavka.

Pitanje 2. Koji od sledećih standarda određuje uslove ispitivanja?

3. BS7925-1

Pitanje 3. Koja od sledećih aktivnosti ne spada u sistemsko testiranje?

2. Integraciono testiranje od vrha nadole.

Pitanje 4. Šta od navedenog NIJE tehnika crne kutije?

3. Linearni niz kodova i preskočite

Pitanje 5. Koja se faza testiranja pojedinačnih softverskih modula kombinuje zajedno u grupu?

2. Integraciono testiranje

Pitanje 6. Šta od navedenog NIJE tehnika statičkog ispitivanja?

1. Pogreška u pogađanju

Pitanje 7. Verifikacija je:

2. Provera da li sistem pravilno razvijamo

Pitanje 8. Beta testiranje je:

1. Izvode ga kupci na svom sajtu

Pitanje 9. Šta NIJE tačno - tester crne kutije:

2. mora biti u stanju da razume izvorni kod

Pitanje 10. Analiza uticaja pomaže u odluci:

4. Koliko regresijskog testiranja treba obaviti

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 11. Razlika između ponovnog i regresivnog testiranja je:

1. ponovno testiranje testova; regresijsko testiranje traži neočekivane neželjene efekte

Pitanje 12. Utvrdite tvrdnju koja se primenjuje u slučaju istraživačkog ispitivanja:

2. Uključuje simultani dizajn i izvođenje testa

Pitanje 13. Šta od navedenog nije deo testova performansi?

1. Test oporavka

Pitanje 14. Šta je od navedenog glavni zadatak planiranja testa?

1. Definisanje testnog pristupa

Pitanje 15. U kojoj aktivnosti glavnog procesa testiranja se stvara test okruženje?

1. Implementacija i izvršenje testa.

Pitanje 16. Koji je skup aktivnosti kojim se osigurava da softver pravilno obavlja određenu funkciju.

1. Verifikacija

Pitanje 17. Verifikacija se zasniva na računaru.

1. Tačno

Pitanje 18. se radi u fazi razvoja otklanjanja grešaka.

1. Kodiranje

Pitanje 19. Pronalaženje ili identifikacija grešaka poznato je kao_____

1. Testiranje
2.

Pitanje 20. Šta određuje ulogu softvera?

3. Sistemski inženjering

Pitanje 21. Kako nazivate testiranje pojedinih komponenata?

2. Jedinstveno testiranje

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 22. Strategija testiranja koja testira aplikaciju u celini je:

4. Testiranje sistema

Pitanje 23. je testirano kako bi se osiguralo da informacije pravilno ulaze u sistem i izvan njega.

1. modularni interfejs

Pitanje 24. Svrha faze zahteva je

3.Sve navedeno

Pitanje 25. Koja od sledećih tvrdnji o ispitivanju komponenata nije tačna?

Testiranje komponenata ne uključuje regresijsko testiranje

Pitanje 26. Tokom koje probne aktivnosti mogu se naći najviše greške?

3.Planiranje

Pitanje 27. Koji procenat troškova razvoja softvera se uzima u obzir prilikom testiranja softvera?

2. 40-50%

Pitanje 28. ispitivanjem osiguranja da se posle promene ne pojave novi nedostaci u kodu

2. regresivno testiranje

Pitanje 29. Koje su svrhe testiranja

4.Sve navedeno

Pitanje 30. U kom modelu faza ispitivanja započinje nakon faze razvoja

Model vodopada

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tematski ciklus 2: "Razvoj softvera"

Odgovori na radne vežbe

Zadatak 1. Umetnite deo donjeg koda koji nedostaje za prikaz "Hello World!".

```
int main() {
 cout << "Hello World!";
 return 0;
}
```

Zadatak 2. Umetnite novi red posle "Hello World" koristeći specijalni simbol :

```
int main() {
 cout << "Hello World! \n";
 cout << "I am learning C ++";
 return 0;
}
```

Zadatak 3. Komentari na jeziku C ++ napisani su posebnim simbolima. Umetnite delove koji nedostaju:

```
// This is a single-line comment
/* This is a multi-line comment */
```

Zadatak 4. Kreiraj varijablu naziva myNum i dodeli joj vrednost 50 .

```
int myNum = 50;
```

Zadatak 5. Prikazuje zbir 5 + 10 pomoću dve promenljive: x i y

```
int x = 5;
int y = 10;
cout << x + y;
```

Zadatak 6. Napravite promenljivu zvanu z, dodelite joj x i y i prikažite rezultat.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
int x = 5;
int y = 10;
int     z       = x + y;
cout <<   z     ;
```

Zadatak 7. Popunite delove koji nedostaju da biste kreirali tri promenljive istog tipa pomoću liste odvojene zarezima:

```
int     x = 5 ,       y = 6 ,       z = 50;
cout << x + y + z;
```

Zadatak 8. Koristite tačnu ključnu reč da biste uneli unos korisnika u promenljivu x:

```
int x;
cout << "Type a number: ";
cin   >>  x      ;
```

Zadatak 9. Popunite delove koji nedostaju da biste odštampali zbir dva broja (koji je postavio korisnik):

```
int x,  y;
int  sum;
cout << "Type a number: ";
cin   >>  x      ;
cout << "Type another number: ";
cin   >>  y      ;
sum = x + y;
```

Zadatak 10. Pomnoži 10 sa 5 i ispiši rezultat.

```
cout << 10  *     5;
```

Zadatak 11. Podeliti 110 sa 5 i ispisati rezultat.

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
cout << 10 [ / ] 5;
```

Zadatak 12. Koristite tačan operator da biste vrednost promenljive x povećali za 1.

```
int x = 10;
[ ++ ]x;
```

Zadatak 13. Pomoću operatora dodavanja dodeljivanja dodajte promenljivoj x vrednost 5.

```
int x = 10;
x [ += ] 5;
```

Zadatak 14. Koristite ispravnu funkciju za ispis najveće vrednosti x i y

```
int x = 5;
int y = 10;
cout << [ max ](x, y);
```

Zadatak 15. Koristite ispravnu funkciju za štampanje kvadratnog korena x.

```
#include <iostream>
#include <[ cmath ]>
using namespace std;

int main() {
 int x = 64;
 cout << [ sqrt ](x);
 return 0;
}
```

Zadatak 16. Koristite tačnu funkciju da broj 2.6 zaokružite na najbliži celi broj.

```
#include <iostream>
#include <[ cmath ]>
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
using namespace std;

int main() {
 cout << round(2.6);
 return 0;
}
```

Zadatak 17. Popunite delove koji nedostaju da biste odštampali vrednosti 1 (za tačno) i 0 (za netačno):

```
bool isCodingFun = true;
bool isFishTasty = false;

cout << isCodingFu;
cout << isFishTasty;
```

Zadatak 18. Popunite delove koji nedostaju da biste ispisali vrednost true (za true):

```
int x = 10;
int y = 9;
cout << ( x > y );
```

Zadatak 19. Dodajte tačan tip podataka za sledeće promenljive:

```
int myNum = 9;
double myDoubleNum = 8.99;
char myLetter = 'A';
bool myBool = false;
string myText = "Hello World";
```

Zadatak 20. Napravite dve logičke promenljive zvane da i ne i dodajte im odgovarajuće vrednosti:

```
bool yay = true;
bool nay = false;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 21. Napravite pozdravnu promenljivu i pokažite njenu vrednost:

```
string greeting = "Hello";
cout << greeting ;
```

Zadatak 22. Ištampaj"Hello World" ako je x veće od y.

```
int x = 50;
int y = 10;
if (x > y) {
 cout << "Hello World";
}
```

Zadatak 23. Print "Hello World" ako je x jednako y

```
int x = 50;
int y = 50;
if (x == y) {
 cout << "Hello World";
}
```

Zadatak 24. Ištampaj"Yes ako je x jednako y, u suprotnom ištampaj "No".

```
int x = 50;
int y = 50;
if (x == y) {
 cout << "Yes";
} else {
 cout << "No";
}
```

Zadatak 25.Ištampaj "1" ako je x jednako y, ištampaj "2" ako je x veće od y, u suprotnom ištampaj 3".

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
int x = 50;
int y = 50;
if (x == y) {
  cout << "1";
} else if (x > y) {
  cout << "2";
} else {
  cout << "3";
}
```

Zadatak 26. Umetnite delove koji nedostaju da biste popunili sledeću „kratku izjavu ako ... inače“ (operater terminala):

```
int ti e = 20;
string result = ( time < 18 ) ? "Good day."
: "Good evening.";
cout << result;
```

Zadatak 27. Unesite delove koji nedostaju da dopunite switch izjavu.

```
int day = 2;
switch (day) {
  case 1:
  cout << "Saturday";
  break;
  case 2:
  cout << "Sunday";
  break;
}
```

Zadatak 28. Dopunite switch izjavu i dodajte odgovarajuću ključnu reč na kraju da biste definisali kod u switch izrazu, nema slučajnih pogodaka.

```
int day = 4;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
switch ( day ) {
 case 1:
 cout << "Saturday";
 break;
 case 2:
 cout << "Sunday";
 break;
 def ault:
 cout << "Weekend";
}
```

Zadatak 29. Napravite varijabilnu promenljivu pod nazivom Obrok, koja treba da bude referenca promenljive Hrana

.

```
string food = "Pizza";
string & meal = food;
```

Zadatak 30. Dobiti memorijsku adresu promenljive Hrane:

```
string food = "Pizza";
cout << & food;
```

Zadatak 31. Napravite promenljivu pokazivača nazvanu pizza, koja mora da upućuje na nizu promenljivih sa imenom hrane:

```
string food = "Pizza";
string* ptr = & food;
```

Zadatak 32. Napravite funkciju pod nazivom myFunction i nazovite je unutar main().

```
void myFunctio() {
 cout << "I just got executed!";
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
int main() {
  myFunctio;
 return 0;
}
```

Zadatak 33. Umetnite deo koji nedostaje da biste dva puta pozvali myFunction

```
void myFunction () {
 cout << "I just got executed!";
}

int main() {
  myFunctio ;
  myFunctio ;
  return 0;
}
```

Zadatak 34. Dodajte fname parametar tipa string u myFunction.

```
void myFunction( string  fname ) {
 cout << fname << "Doe";
}

int main() {
 myFunction ("John");
 return 0;
}
```

Zadatak 35. Unesite deo koji nedostaje da biste ištampali broj 8 main, като користећи specifičnu ključnu reč unutar myFunction:

```
int myFunction (int x) {
  return  5 + x;
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

```
}

int main() {
 cout << myFunction (3);
 return 0;
}
```

Zadatak 36. Kreiraj niz tipa string naziva cars.

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Zadatak 37. Ištampaj vrednost drugog elementa u cars nizu

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[1] ;
```

Zadatak 38. Promenite vrednost iz "Volvo" u "Opel"u cars nizu.

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel" ;
cout << cars [0];
```

Zadatak 39. Proverite elemente cars niza.

```
string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < 4; i++ ) {
 cout << cars[i] << "\n";
}
```

Zadatak 40. Popunite deo koji nedostaje da biste kreirali pozdravnu promenljivu tipa string i dodelili joj vrednost Zdravo.

```
string greeting = "Hello";
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 41. Koristite tačan operator za povezivanje dva niza:

```
string firstName = "John";
string lastName = "Doe";
cout << firstName + lastName;
```

Zadatak 42. Koristite ispravnu funkciju za ispis dužine tekstualnog niza.

```
string txt = "Hello";
cout << txt.length();
```

Zadatak 43. Pristupite prvom znaku (H) myString i odštampajte rezultat:

```
string myString = "Hello";
cout << myString[0];
```

Zadatak 44. Promenite prvi karakter (H) u myString „J“:

```
string myString = "Hello";
myString = 'J';
cout << myString;
```

Zadatak 45. Koristite ispravnu funkciju za čitanje reda teksta koji je stavio korisnik

```
string fullName;
cout << "Type your full name: ";
getline (cin, fullName);
cout << "Your name is: " << fullName;
```

Zadatak 46. Ištampajte I ako je I manje od 6.

```
int i = 1;
while (i < 6) {
  cout << and << "\ n";
    i++;
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Zadatak 47. Koristite do / while outline za štampu sve dok je rezultat manji od 6.

```
int i = 1;
do {
 cout << and << "\ n";
 i++ ;
}
w hile (i <6);
```

Zadatak 48. Koristite za konturu da biste 5 puta odštampali „Da“:

```
for (i t i = 0; i < 5; i++ ) {
 cout << "Yes" << "\n";
}
```

Zadatak 49. Zaustavite outline ako je i 5.

```
for (int i = 0; i < 10; i++) {
 if (i == 5) {
 break ;
 }
 cout << and << "\ n";
}
```

Zadatak 50. U sledećoj petlji, kada je vrednost „4“, pređite direktno na sledeću vrednost

```
for (int i = 0; i < 10; i++) {
 if (i == 4) {
 continue ;
 }
 cout << and << "\ n";
}
```

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Tematski ciklus 2: “Razvoj softvera”

Odgovori na test pitanja

Pitanje 1. Model koji demonstrira implementaciju sistema je:

2. Prototip

Pitanje 2. Održavanje je poslednja faza u modelu vodopada

1. Tačno

Pitanje 3. Faza u kojoj su pojedinačne komponente integrisane i osigurano je da bez grešaka zadovolje zahteve kupaca

2. Testiranje

Pitanje 4. _____je korak u kojem se dizajn pretvara u mašinsko čitljivu formu

4. Kodiranje

Pitanje 5. Zahtevi kupca podeljeni su u logičke module kako bi se olakšalo

4. Implementacija

Pitanje 6. Kako nazivate tehničku osobu koja je u stanju da razume osnovne zahteve?

2. Analitičar

7. Korak u modelu vodopada, koji uključuje sastanak sa kupcem radi razumevanja zahteva.

1. Zahtev za naplatu

Pitanje 8. Metodologija u kojoj su procesi upravljanja projektom bili korak po korak.

2. Vodopad

Pitanje 9. Pojedinac koji planira i upravlja radom.

2. Rukovodilac projekta

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 10. Planirani program ako se završi posao koji zahteva konačno vreme, napor i planiranje

2. Projekat

Pitanje 11. Prikupljanje povezanih podataka je:

3. Baza podataka

Pitanje 12. DBMS je softver.

1. Tačno

Pitanje 13. DBMS upravlja interakcijom između baze podataka.

3. Krajnji korisnici

Pitanje 14. Šta od navedenog nije uključeno u DBMS?

4. HTML

Pitanje 15. Baza podataka je obično_____

2. Orijentisan na korisnika

Pitanje 16. Karakteristika jedinice je:

2. Atribut

Pitanje 17. Šta je IMS?

4. Sistem upravljanja informacijama

Pitanje 18. Model koji su Hammer i Mc Leod razvili 1981. godine je:

1. SDM

Pitanje 19: Objekat=_____+odnosi

3. subjekt

Ptanje 20: DBMS je set _____za pristup podacima

2. Programi

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.

Pitanje 21. DBMS pruža ugodno i efikasno okruženje.

1. Tačno

Pitanje 22. Šta od navedenog nije nivo apstrakcije?

3. Nivo korisnika

Pitanje 23. Nivo koji opisuje kako se zapis čuva.

1. Fizički

Pitanje 24. Nivo pomaže aplikativnim programima da sakriju detalje o tipu

4. Pogled

Pitanje 25. Logička struktura baze podataka.

1. Šema

Pitanje 26. Stvarni sadržaj baze podataka u datom trenutku.

4. Poseban slučaj

Pitanje 27. Šta od navedenog nije objektivni logički model?

2. Mreža

Pitanje 28. SQL je

1. Relacioni model

Pitanje 29. Nivo koji opisuje podatke uskladištene u bazi podataka i odnose između podataka.

2. Logičan

Pitanje 30. Svaki algoritam je program.

2. Pogrešno

Projekat sufinansira EU kroz Interreg-IPA Program prekogranične saradnje Bugarska – Srbija
Ova publikacija je proizvedena uz pomoć Evropske unije kroz Interreg-IPA CBC Program Bugarska-Srbija, CCI br. 2014TC16I5CB007. Sadržaj ove publikacije isključiva je odgovornost Nacionalne asocijacije Pravna inicijativa za otvorenu vladu i ni na koji način ne može odražavati stavove Evropske unije ili Upravljačkog tela Programa.