## 1. Značenje reči paradigma i programska paradigma:

Paradigma označava model, uzorak, ili skup pravila koji se koriste kao osnov za nešto. U kontekstu programiranja, programska paradigma se odnosi na stil ili način organizovanja i strukturiranja programa.

## 2. Uloga programskih paradigmi:

Programske paradigme nude osnovne koncepte, principe i stilove programiranja koji pomažu programerima da razvijaju efikasne, struktuirane i održive softverske sisteme. Svaka paradigma pristupa rešavanju problema na svoj način.

## 3. Definicija programskog jezika:

Programski jezik je formalni sistem simbola i pravila koji omogućavaju programerima da napišu softverski kod koji računar može izvršiti.

## 4. Povezanost paradigmi i jezika:

Programski jezici su često dizajnirani kako bi podržavali određene programske paradigme. Različiti jezici imaju različite naglaske ili podržavaju različite kombinacije paradigmi.

## 5. Razvoj programskih jezika:

Programski jezici su evoluirali tokom vremena, prolazeći kroz različite generacije. Razvoj uključuje dodavanje novih funkcionalnosti, poboljšanja efikasnosti, i prilagođavanje novim potrebama i paradigama.

## 6. U koju grupu jezika spadaju mašinski jezici i asemblerski jezici:

Mašinski jezici i asemblerski jezici spadaju u grupu niskog nivoa ili asembler jezika.

## 7. Da li je kod na nekom mašinskom jeziku prenosiv sa jednog na sve druge računare:

Ne, kod napisan na mašinskom jeziku nije prenosiv između različitih računara jer je specifičan za arhitekturu svakog pojedinačnog računara.

## 8. Da li asembler zavisi od mašine na kojoj se koristi:

Da, asembler jezici su specifični za određenu arhitekturu mašine i zavise od nje.

## 9. Šta karakteriše proceduralnu paradigmu:

Proceduralna paradigma karakteriše koncepcija procedura ili funkcija koje obavljaju određene zadatke. Program se sastoji od poziva procedura koje mogu sadržavati niz naredbi.

## 10. Šta karakteriše deklarativnu paradigmu:

Deklarativna paradigma se fokusira na opisivanje željenog rezultata, a ne na specificiranju koraka koje računar treba preduzeti kako bi došao do tog rezultata.

## 11. Koje su osnovne četiri programske paradigme:

Osnovne četiri programske paradigme su:

1. Proceduralna paradigma

2. Funkcionalna paradigma

3. Deklarativna paradigma

4. Objektno-orijentisana paradigma

## 12. Nabroj bar četiri dodatne programske paradigme:

- Dodatne programske paradigme uključuju:

1. Logička paradigma

2. Reaktivna paradigma

3. Paralelna paradigma

4. Imperativna paradigma

## 13. Koje su osnovne karakteristike imperativne paradigme:

Imperativna paradigma se fokusira na niz naredbi koje menjaju stanje programa, pri čemu se akcenat stavlja na tome kako se željeni rezultat postiže.

## 14. Nabroj tri jezika koji pripadaju imperativnoj paradigmi:

C, Java, Python.

## 15. Koje su osnovne karakteristike funkcionalne paradigme:

Funkcionalna paradigma se fokusira na funkcije kao osnovne gradivne blokove, gde se akcenat stavlja na izračunavanje rezultata funkcija bez promene stanja programa.

## 16. Nabroj tri jezika koji pripadaju funkcionalnoj paradigmi:

Haskell, Lisp, Scala.

## 21. Zašto su nastajale i nastaju nove programske paradigme:

Nove programske paradigme nastaju kako bi se nosile sa promjenama u zahtjevima i tehnologijama, poboljšale produktivnost programera i olakšale rešavanje određenih vrsta problema.

## 22. Koje su osnovne karakteristike komponentne paradigme:

Komponentna paradigma se fokusira na izgradnju softvera pomoću ponovno upotrebljivih i nezavisnih komponenti.

## 23. Nabroj tri jezika koji podržavaju komponentnu paradigmu:

Java, C#, Delphi.

## 24. Koje su osnovne karakteristike reaktivne paradigme:

Reaktivna paradigma se bavi asinhronim tokom podataka i reagovanjem na promene u stanju sistema.

## 25. Nabroj tri jezika koji pripadaju reaktivnoj paradigmi:

RxJava, ReactJS, Akka.

## 26. Kroz koje osnovne faze prolazi razvoj svakog programa:

Osnovne faze razvoja svakog programa uključuju analizu, dizajn, implementaciju, testiranje, održavanje.

## 27. Koje se faze smatraju ključnim u toku razvoja svakog programa:

Ključne faze uključuju analizu i dizajn, jer postavljaju osnovu za implementaciju.

## 28. Šta sve obuhvata definisanje programskog zadatka:

Definisanje programskog zadatka obuhvata identifikaciju problema, specifikaciju zahteva, i postavljanje ciljeva koji se žele postići softverom.

## 29. Koja faza razvoja programa je neophodna pre početka pisanja izvornog koda:

Pre početka pisanja izvornog koda, neophodna je faza analize, kako bi se jasno razumele potrebe i zahtevi sistema koji se razvija.

## 30. Od čega zavisi izbor metode projektovanja programa:

Izbor metode projektovanja programa zavisi od prirode problema, zahteva klijenta, iskustva programera, dostupnih resursa, i specifičnosti projekta.

## 31. Osnovne karakteristike strukturne metode i objektno-orijentisane metode projektovanja programa:

Strukturna metoda se fokusira na podelu sistema na manje, lakše upravljive module. Objektno-orijentisana metoda koristi objekte koji integrišu podatke i funkcije.

## 32. Koji programski alat se uobičajeno koristi za pisanje izvornog koda programa:

Uobičajeno se koristi integrisano razvojno okruženje (IDE) kao što su Visual Studio, Eclipse, ili IntelliJ IDEA.

## 33. Zadaci programskih alata kompajlera, linkera i debagera:

Kompajler prevodi izvorni kod u mašinski jezik, linker povezuje različite delove koda u izvršnu datoteku, a debager pomaže u pronalaženju i ispravljanju grešaka u kodu.

## 34. Osnovne metode testiranja rada programa:

Osnovne metode testiranja uključuju jedinčno testiranje, integraciono testiranje, sistemsko testiranje i prihvatno testiranje.

## 35. Šta sve obuhvata održavanje programa:

Održavanje programa obuhvata sve aktivnosti koje su neophodne da bi se program prilagodio novim zahtevima, ispravile greške i poboljšala performansa.

## 36. Šta predstavljaju algoritmi za programe i kako se oni mogu predstaviti:

Algoritmi predstavljaju set koraka ili pravila za rešavanje određenog problema. Mogu se predstaviti pseudokodom, dijagramima toka, ili matematičkim izrazima.

## 37. Koje su osnovne strukture zastupljene u programima:

Osnovne strukture uključuju sekvencu, selekciju i petlje.

## 38. Gde se u programima koristi struktura selekcije:

Struktura selekcije se koristi za donošenje odluka i izbor između različitih putanja izvršavanja u programu.

## 39. Kakve su strukture petlji zastupljene u programima:

Strukture petlji uključuju for, while i do-while petlje.

## 40. Koje se programske strukture mogu uklapati jedne unutar drugih:

Različite programske strukture mogu se uklapati jedna unutar druge, formirajući složenije logičke strukture.

## 41. Koji se model projektovanja koristi za razvoj programa na jeziku C:

Za razvoj programa na jeziku C često se koristi proceduralni model projektovanja.

## 42. Da li se kod identifikatora jezika C pravi razlika velika/mala slova:

Da, u jeziku C postoji razlika između velikih i malih slova kod identifikatora.

## 44. Koje su vrste komponenata obavezno zastupljene u programima na jeziku C:

U programima na jeziku C, obavezno su zastupljene promenljive i funkcije.

## 45. Koje se oznake koriste za početak i kraj tela (naredbi) svake C funkcije:

Telo funkcije počinje sa "{", a završava sa "}".

## 46. Koji se simbol koristi za početak preprocesorske direktive u C programu:

Za početak preprocesorske direktive u C programu koristi se simbol "#".

## 47. Koje značenje ima preprocesorska direktiva #include:

Preprocesorska direktiva #include se koristi za uključivanje sadržaja drugih fajlova u izvorni kod programa.

## 48. Pod kojim uslovom C program može koristiti funkcije iz jedne standardne biblioteke:

C program može koristiti funkcije iz jedne standardne biblioteke ako je ta biblioteka uključena u program pomoću #include direktive.

## 49. Koja standardna C funkcija radi formatirani izlaz na ekran:

Standardna C funkcija za formatirani izlaz na ekran je printf.

## 50. Za koje vrste podataka se može koristiti C funkcija formatiranog izlaza na ekran:

C funkcija formatiranog izlaza na ekran se može koristiti za različite vrste podataka kao što su brojevi, stringovi, itd.

## 51. Koja je oznaka za kraj naredbe u C programu:

Oznaka za kraj naredbe u C programu je ";".

## 52. Koje su sve vrste naredbi zastupljene u C programima:

Vrste naredbi u C programima uključuju izraze, dodele, kontrolne strukture, pozive funkcija, itd.

## 53. Da li više naredbi može biti napisano u jednoj liniji izvornog koda C programa:

Da, više naredbi može biti napisano u jednoj liniji izvornog koda C programa, odvojenih sa ";".

## 56. Da li su komentari obavezne komponente C programa i koje značenje imaju:

Komentari nisu obavezne komponente C programa, ali su od suštinskog značaja za razumevanje koda. Služe programerima za objašnjenje i dokumentaciju.

## 57. Da li se jedan komentar može napisati u nastavcima, u više linija izvornog koda C programa:

- Da, jedan komentar se može napisati u nastavcima, koristeći /\* za početak i \*/ za kraj komentara.

## 58. Kako se može prikazati reč "pozdrav" na ekranu pomoću C programa:

Prikazivanje reči "pozdrav" na ekranu može se postići korišćenjem funkcije printf. Minimalan broj linija koda zavisi od organizacije programa.

## 59. Šta su promenljive, a šta konstante:

Promenljive su simbolički imenovane lokacije u memoriji koje čuvaju vrednosti podataka i mogu im se pridružiti različite vrednosti tokom izvršavanja programa. Konstante su vrednosti koje ne mogu biti promenjene tokom izvršavanja programa.

## 60. Koje vrste značenja mogu sadržavati C identifikatori:

C identifikatori mogu sadržavati slova, cifre i donje crte (\_), ali moraju počinjati slovom ili donjom crtom.

## 61. Koji su celobrojni C tipovi promenljivih i konstanti najviše u upotrebi:

Celobrojni C tipovi uključuju int (celobrojni), char (karakter) i long (dugi celobrojni). Za konstante se često koristi const.

## 62. Koji su realni C tipovi promenljivih i konstanti postoje:

Realni C tipovi uključuju float (plutajući zarez jednostruke preciznosti) i double (plutajući zarez dvostruke preciznosti).

## 63. Da li deklaracija promenljivih automatski inicijalizuje njihove sadržaje:

Ne, deklaracija promenljivih ne inicijalizuje automatski njihove sadržaje. Promenljive treba eksplicitno inicijalizovati pre upotrebe.

## 64. Koji načini postoje za unošenje konstanti u program:

Konstante se mogu uneti direktno u izraz (literalne konstante), koristeći #define preprocesorsku direktivu ili kroz enumeracije.

## 65. Po čemu se međusobno razlikuju deklaracije promenljivih i konstanti:

Deklaracije promenljivih rezervišu prostor u memoriji za skladištenje vrednosti, dok deklaracije konstanti definišu imena za vrednosti koje se ne mogu menjati.

## 66. Da li definicija simboličkih konstanti rezerviše prostor u memoriji:

Ne, definicija simboličkih konstanti ne rezerviše prostor u memoriji. One se koriste kao zamene za određene vrednosti tokom kompilacije.

## 67. Za koje vrste konstanti se može uvesti tip njihovog nabrajanja (enum):

Tip nabrajanja (enum) se često koristi za definisanje konstanti koje predstavljaju set povezanih vrednosti.

## 68. Koji je minimalni broj argumenata kod funkcije printf(), a koji kod funkcije scanf():

Minimalni broj argumenata za funkciju printf() je 1 (format string), a za funkciju scanf() je 2 (format string i adresa promenljive).

## 69. Koje se oznake formata mogu koristiti kod celobrojnih tipova char i int u format stringu funkcije printf(), a koje kod funkcije scanf():

Za celobrojne tipove char i int, oznake formata mogu biti %c (char) i %d (decimalni) u printf(), dok su %c i %d takođe moguće u scanf().

## 70. Koje se oznake formata mogu koristiti kod realnih tipova float i double u format stringu funkcije printf(), a koje kod funkcije scanf():

Oznake formata za realne tipove float i double u printf() mogu biti %f (float) i %lf (double), dok su u scanf() takođe %f i %lf.

## 71. Da li se kod prikaza sadržaja realnih promenljivih tipova float i double može definisati različit način prikaza od podrazumevanog:

Da, kod prikaza sadržaja realnih promenljivih tipova float i double može se koristiti određeni format za prikazivanje, npr. %.2f za dva decimalna mesta.

## 72. Da li formatirani ulaz/izlaz obavezno podrazumeva konverzije formata podataka koji se učitavaju u memoriju/iz memorije:

Da, formatirani ulaz/izlaz obavezno podrazumeva određene konverzije formata podataka kako bi se pravilno interpretirali i prikazali.

## 73. Da li funkcije formatiranog ulaza/izlaza mogu raditi sa mešovitim tipovima podataka:

Da, funkcije formatiranog ulaza/izlaza mogu raditi sa mešovitim tipovima podataka, omogućavajući efikasno čitanje i pisanje podataka različitih tipova.

## 74. Šta sve definiše sintaksa programskog jezika za svaki svoj operator:

Sintaksa programskog jezika za svaki svoj operator definiše način korišćenja, prioritet, asocijativnost i operandi koje operator podržava.

## 75. Koje vrste operatora postoje u C jeziku, u zavisnosti od broja operanada:

Operatori u C jeziku mogu biti unarni (jedan operand), binarni (dva operanda) ili ternarni (tri operanda).

## 76. Da li zagrade mogu promeniti prioritet operatora u programima:

Da, zagrade mogu promeniti prioritet operatora, određujući redosled izvršavanja operacija u složenim izrazima.

## 77. Koje značenje ima aritmetički operator inkrement, a koje dekrement:

Aritmetički operator inkrement (++) povećava vrednost operandi za 1, dok operator dekrement (--) smanjuje vrednost operandi za 1.

## 78. Koji je razlog primene logičkih operatora, u kombinaciji sa relacijskim:

Logički operatori se koriste za kombinovanje rezultata relacijskih izraza, omogućavajući kreiranje kompleksnih logičkih uslova u kontrolnim strukturama.

## 76. U kojim slučajevima je u programu potrebno upotrebiti naredbu if:

Naredba if se koristi kada želimo da izvršimo određeni blok koda samo ako je određeni uslov ispunjen.

## 77. Primenom koje naredbe može eliminisati uslovni C operator:

Uslovni C operator (ternarni operator) može biti eliminisan primenom naredbe if.

## 78. Primenom kojih operatora na nivou bita omogućava proveru vrednosti pojedinih bitova u podacima:

Operatori na nivou bita poput & (i), | (ili) i ^ (isključivo ili) omogućavaju proveru vrednosti pojedinih bitova u podacima.

## 79. Primenom kojih operatora na nivou bita omogućava setovanje/resetovanje/promenu vrednosti pojedinih bitova u podacima:

Operatori na nivou bita poput |= (setovanje), &= ~(resetovanje) i ^= (promena) omogućavaju setovanje, resetovanje i promenu vrednosti pojedinih bitova.

## 80. Kod kojih tipova podataka se može primeniti operator veličine tipa:

Operator veličine tipa se može primeniti kod bilo kog tipa podataka.

## 81. U kojim slučajevima se može koristiti operator konverzije tipa:

Operator konverzije tipa se koristi kada želimo eksplicitno promeniti tip podataka.

## 82. Kako rade funkcije getchar() i putchar():

Funkcija getchar() čita jedan karakter sa standardnog ulaza, dok funkcija putchar() ispisuje karakter na standardni izlaz.

## 83. Pomoću kojih se standardnih funkcija iz biblioteke ctype.h može proveriti da li je znak: slovo/decimlana cifra/štampajući znak:

Za proveru tipa znaka, koriste se funkcije isalpha(), isdigit() i isprint() iz biblioteke ctype.h.

## 84. Pomoću kojih se standardnih funkcija iz biblioteke ctype.h može konvertovati malo slovo u isto veliko/veliko slovo u isto malo:

Za konverziju slova koriste se funkcije toupper() i tolower() iz biblioteke ctype.h.

## 85. Šta je sve potrebno znati u vezi standardne funkcije pre nego što se predvidi njeno korišćenje:

Pre korišćenja standardne funkcije, potrebno je znati njen povratni tip, potrebne argumente i njenu namenu.

## 86. Šta predstavlja ciklus u programu:

Ciklus u programu predstavlja blok koda koji se izvršava više puta dok je određeni uslov ispunjen.

## 87. Šta je neophodno definisati na početku projektovanja bilo koje naredbe ciklusa u programu:

Na početku projektovanja bilo koje naredbe ciklusa, neophodno je definisati uslov koji određuje trajanje ciklusa.

## 88. Koje naredbe ciklusa ispituju uslov na vrhu, a koje u dnu naredbe:

Naredbe while i do..while ispituju uslov na vrhu, dok naredba for ispituje uslov na vrhu naredbe.

## 89. U kojim slučajevima se koristi naredba for i koja je njena struktura:

Naredba for se koristi kada unapred znamo broj iteracija. Njena struktura je for (inicijalizacija; uslov; ažuriranje).

## 90. Kako se može opisati tok izvršavanja naredbe for:

Tok izvršavanja naredbe for uključuje izvršavanje inicijalizacije, proveru uslova, izvršavanje tela petlje, ažuriranje brojača i ponovno proveravanje uslova.

## 91. Kako se može opisati tok izvršavanja naredbe while:

Tok izvršavanja naredbe while uključuje proveru uslova, izvršavanje tela petlje, ažuriranje brojača i ponovno proveravanje uslova.

## 92. Kako se može opisati tok izvršavanja naredbe do..while:

Tok izvršavanja naredbe do..while uključuje izvršavanje tela petlje, ažuriranje brojača i proveru uslova na kraju petlje.

## 93. Koji od tri izraza naredbe for (početni izraz/uslov/izraz promene) može biti složen (sadržati u sebi više izraza):

Sva tri izraza naredbe for (početni izraz, uslov i izraz promene) mogu biti složeni i sadržavati više izraza.

## 94. Koliko puta se u toku for ciklusa može izvršiti početni izraz:

Početni izraz u toku for ciklusa može se izvršiti samo jednom, na početku ciklusa.

## 95. Da li korak promene brojača u for naredbi može biti samo pozitivan/samo negativan/pozitivan ili negativan:

Korak promene brojača u for naredbi može biti pozitivan, negativan ili nula, u zavisnosti od željenog smera iteracije.

## 96. Koja vrsta izraza može biti napisana kao uslov naredbe while:

Uslov naredbe while može biti bilo koja logička ekspresija, često uključujući relacijske i logičke operatore.

## 97. U kojim slučajevima se koristi naredba while, a u kojim naredba do..while:

Naredba while se koristi kada uslov treba biti ispunjen pre prvog izvršavanja tela petlje, dok se naredba do..while koristi kada želimo garantovati barem jedno izvršavanje tela petlje.

## 98. Koji je minimalni broj iteracija kod while, a koji kod do..while ciklusa:

Minimalni broj iteracija kod while ciklusa je 0, dok je kod do..while ciklusa minimalan broj iteracija 1.

## 99. Unutar koje naredbe se može uklopiti bilo koja od postojećih naredbi ciklusa:

Bilo koja od postojećih naredbi ciklusa može se uklopiti unutar naredbe funkcije ili bilo koje druge naredbe koja dozvoljava blok koda.

## 100. Da li naredba ciklusa može biti napisana unutar naredbe selekcije i da li može biti i obrnuto:

- Da, naredba ciklusa može biti napisana unutar naredbe selekcije, i obrnuto. Mogu se međusobno gnijezditi.

## 101. Šta znači naredba skoka u programu:

- Naredba skoka je naredba koja omogućava prelazak iz jednog dela programa u drugi, preskakajući određeni deo koda.

## 102. Da li su u C programima zastupljeni uslovni ili bezuslovni skokovi:

U C programima su zastupljeni i uslovni i bezuslovni skokovi.

## 103. Na kojim mestima u programu može biti napisana naredba skoka:

Naredba skoka može biti napisana na bilo kom mestu u programu gde je dozvoljen blok koda, ali se obično koristi za upravljanje tokom izvršavanja naredbi.

## 104. U kojim se sve slučajevima može koristiti naredba break:

Naredba break se može koristiti u okviru petlji (for, while, do..while) i switch naredbi.

## 105. Da li je za izlaz iz ciklusa pomoću break naredbe neophodno postojanje if naredbe za proveru uslova:

Ne, nije neophodno, ali se često koristi u kombinaciji sa if naredbom kako bi se postavio uslov za izlazak iz ciklusa.

## 106. Da li naredba break u svakom slučaju znači definitivan izlaz iz ciklusa:

Da, naredba break znači definitivan izlaz iz ciklusa u kojem se nalazi.

## 107. U kojim se sve slučajevima može koristiti naredba continue:

Naredba continue se može koristiti unutar petlji (for, while, do..while) za preskakanje ostatka trenutne iteracije i prelazak na sledeću.

## 108. Da li je za izlaz iz iteracije ciklusa pomoću continue naredbe neophodno postojanje if naredbe za proveru uslova:

Ne, nije neophodno, ali se često koristi u kombinaciji sa if naredbom kako bi se postavio uslov za preskakanje dela iteracije.

## 109. Kako izgleda struktura naredbe switch:

Struktura naredbe switch uključuje izraz koji se evaluira, listu slučajeva (case), i opcionu naredbu default.

## 110. Kako radi naredba switch:

Naredba switch evaluira izraz i traži odgovarajući slučaj (case) koji se podudara sa vrednošću izraza. Izvršava se kod tog slučaja i nastavlja se izvršavanje sve do naredbe break ili kraja switch bloka.

## 111. Koju vrstu izraza može testirati naredba switch:

Naredba switch može testirati samo celobrojne izraze (integers) ili karaktere (char).

## 112. Kako se naredba break primenjuje kod naredbe switch:

Naredba break se koristi za prekid izvršavanja naredbe switch, prelazak na kraj bloka switch.

## 113. Da li broj slučajeva kod naredbe switch može biti proizvoljan i da li je neophodan podrazumevani slučaj:

Broj slučajeva kod naredbe switch može biti proizvoljan, i nije neophodno imati podrazumevani slučaj, ali je preporučljivo za neočekivane vrednosti izraza.

## 114. Kako se ostvaruje "propadanje" iz slučaja u slučaj kod naredbe switch:

"Propadanje" iz slučaja u slučaj kod naredbe switch ostvaruje se kada se naredba break ne koristi, pa se izvršavanje nastavlja u sledećem slučaju.

## 115. Koje naredbe može biti zamenjene pomoću naredbe switch:

Naredba switch može zameniti niz uzastopnih if-else if naredbi kada postoji potreba za testiranjem iste promenljive na više vrednosti.