Razvoj softvera 2

Igor Rodić

KONTROLISANJE PETLJI

U većini jezika koristićete razne petlje:

- Brojačka petlja se izvršava određeni broj puta, recimo jednom za svakog zaposlenog.
- Petlja sa konstantnom proverom uslova ne zna unapred koliko će puta biti izvršena i u svakoj iteraciji testira da li je došla do kraja. Npr. izvršava se dok ima novca, dok korisnik ne izabere da se izađe, ili dok ne naiđe na grešku.

- Večna petlja izvršava se večno od momenta kada je startovana. Ovakva petlja pronalazi se u integrisanim sistemima kao što su pejsmejkeri, mikrotalasne pećnice...
- Iterativna petlja izvršava se po jednom za svaki element u container klasi.

Petlje takođe mogu da se razlikuju po:

- Fleksibilnosti (da li se petlja izvršava određeni broj puta, ili proverava neki uslov pri svakoj iteraciji).
- Primeri fleksibilnih petlji su: for, while, do-while...
- Primeri nefleksibilnih petlji su: For-Next, foreach...

- Lokaciji testa uslova petlje (da li se uslov petlje proverava na početku ili na kraju petlje).
- Petlje koje proveravaju uslov na početku su: for, while, foreach...
- Petlja koja proverava uslov na kraju je do-while.
- A Do-Loop-While petlja može uslov da proverava i na početku i na kraju.

While petlja:

- Koristi se kada niste unapred sigurni koliko će se puta petlja izvršiti.
- Provera uslova petlje vrši se po jednom za svaku iteraciju petlje, i to na njenom početku.
- U slučaju da želite fleksibilnu petlju sa proverom uslova na njenom kraju, treba da odaberete do-while petlju.

Petlja sa izlazom:

- Petlja u kojoj se uslov izlaska iz petlje nalazi na njenoj sredini, umesto na početku ili kraju.
- Postoji u Visual Basic-u (Loop-With-Exit), u C i C++ može da se implementira kombinovanjem while petlje i break naredbe.
- Izbegava kodiranje petlje-i-po (dupliranje pojedinih linija koda)

For petlja:

- For petlja je dobar izbor kada vam treba petlja koja je izvršava tačno određeni broj puta.
- Koristi se kada vam nisu potrebne unutrašnje kontrole u petlji.
- Poenta for petlje je da je namestite na početku petlje i da onda zaboravite na nju, ne treba da radite ništa unutar petlje da bi je kontrolisali.

Foreach petlja:

- Foreach petlja je korisna kada neka operacija treba da bude izvršena nad svim članovima niza, ili neke druge container klase.
- Eliminise šanse da se pojavi greška pri definisanju petlje i njenoj aritmetici.

Kontrola petlji:

- Mnogo stvari može da pođe po zlu kada su u pitanju petlje: neispravna inicijalizacija, izostavljena inicijalizacija brojača petlje ili drugih promenljivih, pogrešan uslov izlaska iz petlje, zaboravljanje inkrementiranja brojača petlje...
- Ovi problemi mogu biti redukovani uz pomoć dve prakse.

- Prvo, minimizovati broj faktora koji utiču na petlju.
- Drugo, tretirati petlju što je više moguće kao funkciju, zadžati većinu kontrole petlje van nje.
- Tretirati petlju kao crnu kutiju, okolni kod zna uslove petlje, ali ne i nje sadržaj.

Ulazak u petlju:

- Pri ulasku u petlju treba se držati sledećih pravila.
- Ulaziti u petlju iz samo jedne lokacije.
- Inicijalizacioni kod treba da se nalazi odmah iznad petlje, čime se višestruko smanjuje mogućnost pojave grešaka pri modifikaciji ili proširivanju petlje.

Neke preporuke:

- Koristiti while(true) za beskonačne petlje.
- Koristiti for petlju kada je to pogodno, tj. kada god je moguće zameniti neku petlju njom.
- Ne koristiti for petlju kada je while petlja pogodnija (kontrola for petlje treba da bude u njenoj definiciji, ne u njenom telu).

Procesiranje tela petlje:

- Koristiti { i } za okruživanje tela petlje.
- Izbegavati prazne petlje.
- Držati naredbe održavanja petlje (naredbe kontrole petlje) na početku ili kraju.
- Jednom petljom obavljati samo jednu funkciju.
- Osigurati se da se petlja završava.

- Uslovi izlaska iz petlje treba da budu očigledni.
- Ne izlaziti iz petlje menjanjem njenog brojača (namestiti da se brojač petlje poklapa sa uslovom izlaska iz nje).
- Izbegavati kod koji se oslanja na vrednost brojača petlje nakon njenog završetka.
- Razmotriti korišćenje sigurnosnih brojača.

Prevremeni izlazak iz petlje:

- Mnogi jezici pružaju načine da se iz petlje izađe pre nego što je ona ispunila svoj uslov (došla do svog prirodnog kraja).
- U C, C++ i Java programiskim jezicima to je break naredba.
- Slična naredba je i naredba continue, ona služi da se preskoči deo koda koji se nalazi posle nje i prelazi se na sledeću iteraciju petlje (kao if-else).

Neke preporuke:

- Bolje je koristiti break naredbu nego boolean promenljive za izlazak iz petlje.
- Biti oprezan sa petljom koja ima mnogo raštrkanih break naredbi u sebi.
- Koristiti naredbu continue za testiranja na početku petlje (preskakanje tela petlje ako je neki uslov ispunjen).
- Pažljivo koristiti break i continue.

Provera završetka petlje:

- Za petlju su važna tri slučaja, početni, srednji i završni.
- Kada kreirate petlju preporuka je da prvo u glavi prođete kroz ova tri slučaja, da ne bi došlo do grešaka tipa “promašen za jedan”.
- Uzmite kalkulator i proverite rešenje.
- Ovakva praksa pravi razliku između efikasnog i neefikasnog programera.

Korišćenje promenljivih u petlji:

- Koristiti redne brojeve ili enumerabilne tipove za granice kako nizova tako i petlji (celi brojevi su dobar izbor).
- Davati intuitivna imena promenljivima radi lakšeg razumevanja koda i izbegavanja preklapanja brojača u ugnježdenim petljama.
- Ograničiti upotrebu brojačke petlje na samu petlju (C++ i Java zbog ovoga dozvoljavaju da se brojači deklarišu unutar petlje).

Koliko dugačka treba da bude jedna petlja?

- Pravite petlje dovoljno kratke da mogu da se vide odjednom (kraće od 50 linija koda).
- Ograničite ugnježdavanje na 3 nivoa.
- Izmestite kod iz ugnježdenih petlji u funkcije.
- Ako imate duže petlje, potrudite se da budu savršeno jasne.

Kreiranje komplikovanih petlji – obrnutim procesom:

- Početi jednim slučajem.
- Kodirati taj slučaj sa literalima.
- Staviti petlju oko tog slučaja i zameniti literale brojačima petlje ili izračunatim vrednostima.
- Staviti još jednu petlju oko nje, ako je to potrebno, i zameniti još literala.
- Ponavljati ovaj proces sve dok za njime ima potrebe.

Sličnosti između petlji i nizova:

- Petlje i nizovi često su povezani.
- Petlja se u mnogim slučajevima koristi da bi izvršila neke operacije nad nizom.
- Brojači petlji su u odnosu 1-1 sa indeksima nizova.
- Neki jeziki npr. APL i Fortran 90 omogućavaju moćne operacije nad nizovima čime se eliminiše potreba za petljama.

Ključne napomene:

- Petlje su komplikovane, jednostavnošću ćete pomoći čitaocima vašeg koda.
- Tehnike za pisanje jednostavnih petlji uključuju izbegavanje egzotičnih petlji, minimizaciju ugnježđavanja, jasnost ulaza i izlaza, i zadržavanje koda za održavanje petlje na jednom mestu.

- Sa brojačkim promenljivim u petlji često se ne postupa na pravi način, dajte im jednostavne nazive i koristite ih samo u jednu svrhu.
- Pažljivo u glavi prođite kroz petlju da bi ste bili sigurni da se izvršava kako vi to očekujete, u svim mogućim slučajevima, i da se završava posle svih mogućih uslova izlaska.