Selekcije, operatori uspoređivanja, logički i aritmetički operatori

Ciljevi:

- proučiti i shvatiti pojam selekcije
- proučiti i shvatiti pojam operator
- proučiti i shvatiti pojam konstrukti (kontrolne strukture)
- shvatiti uporabu operatora uspoređivanja
- shvatiti uporabu logičkih operatora
- shvatiti uporabu aritmetičkih operatora

Pregled lekcije

U ovoj lekciji ćemo nastaviti izgrađivati sliku o programiranju i svim elementima potrebnim za izgradnju programa.

Jedan od elemenata potrebnih za izgradnju programa su i selekcije. Selekcije se koriste kada se izvršavanje izvjesnog programskog bloka veže uz istinitost ili lažnost nekog logičkog izraza.

Selekcija je jedan od elementa nečega što nazivamo programski konstrukti (kontrolne strukture). Kako samo ime kaže, programski konstrukti su elementi od kojih konstruiramo (gradimo) računalni program.

U programske konstrukte ubrajamo:

- sekvencu (slijed)
- selekciju (odabir)
- iteraciju (ponavljanje, petlja)
- skok (naredbe **`goto`**, **`break`**, **`return`** i **`continue`**)

Sekvenca je jednostavno slijed tj. niz linija programskog koda koje se izvršavaju jedna za drugom.

Selekcija će biti objašnjena u nastavku lekcije.

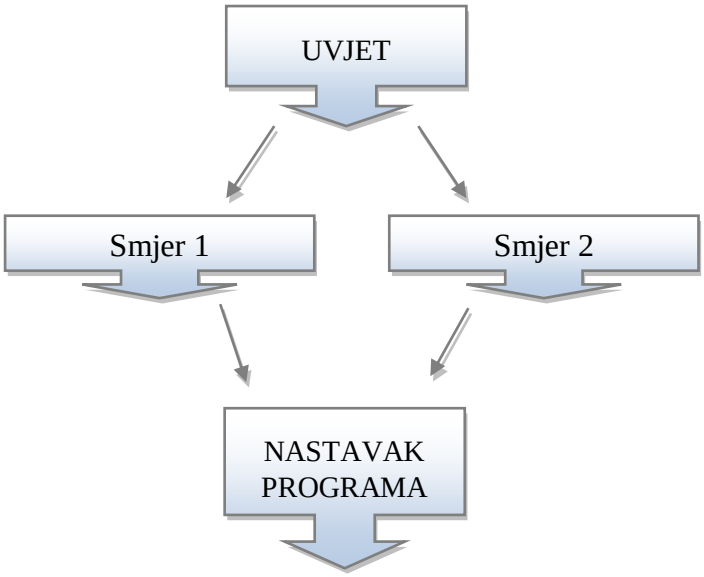
Iteracija je dio programskog koda koji se izvršava više puta uzastopno. O iteraciji će biti više govora u sljedećim lekcijama.

Tipičan primjer skoka je naredba **`goto`** koja predstavlja pomak na bilo koji programski dio sa bilo kojeg drugog programskog dijela. Ona se u pravilu više ne koristi ili se koristi jako malo jer narušava strukturiranost programa. Primjerice uporabom ove naredbe možemo skočiti van iz petlje prije nego što ona završi svojim prirodnim putem. Problem je također i u tome da se je u programskom kodu koji često koristi **`goto`** iznimno teško snaći. U prilog

izbjegavanja korištenja `goto` naredbe treba reći i to što je matematički dokazano da je ova naredba suvišna i da se svaki algoritam može izvesti i bez nje. Osim naredbe **`goto`** u uporabi je još nekoliko naredbi skoka koje se redovito koriste i ne narušavaju toliko strukturiranost programa kao što je to slučaj kod **`goto`** naredbe, no i njih valja izbjegavati jer narušavaju čitljivost programa. Ove naredbe su **`break`**, **`return`** i **`continue`** a više o njima će biti govora u sljedećim lekcijama.

Selekcija

Selekcija programski konstrukt tj. točka u programu u kojoj se odlučuje temeljem uvjeta (logičkog izraza) koji će se od nekoliko mogućih programskih blokova izvesti. U stvarnom životu jedan primjer selekcije je situacija u kojoj se nalazimo na raskrižju i odlučujemo na temelju nečega (nekog uvjeta) kuda ćemo krenuti. Uvjet može biti nešto poput npr. željenog odredišta, vremenskih prilika, itd. Temeljem toga možemo reći, dok stojimo na raskrižju, sljedeće: Ukoliko želim u trgovinu informatičkom opremom idem lijevo, inače idem desno.



Slika 1: Primjer selekcije

Selekcija može biti u svom jednostavnijem obliku ili složenijem obliku. Primjer koji smo upravo naveli je primjer jednostavne selekcije koju nazivamo i selekcija tipa **`if...else`**. U ovom obliku selekcije, ukoliko je uvjet zadovoljen idemo jednim smjerom a ukoliko uvjet nije zadovoljen idemo drugim smjerom.

```
if (uvjet)
   {blok 1}
else
   {blok 2}
```

Navedeni primjer čitamo **`if`** (ako) je uvjet zadovoljen idemo na blok 1 **`else`** (u svakom drugom slučaju) idemo na blok 2. Već smo u prošloj lekciji govorlili, a sada ponavljamo da se u programskom jeziku C++ programski blokovi stavljaju u vitičaste zagrade. Dakle, oba bloka vezana uz selekciju trebaju biti unutar vitičastih zagrada.

Navedeni primjer ne predstavlja najjednostavniju uporabu selekciju. Najjednostavnija varijanta selekcije je selekcija tipa **if**. Kod ove selekcije se ispituje uvjet i ukoliko je on istinit izvršava se određeni dio koda (smjer 1) a ukoliko nije taj dio koda se preskače.

```
if (uvjet)
   {blok 1}
```

Dakle, navedeni primjer čitamo **if** (ako) je uvjet istinit idemo na smjer 1 a ako nije idemo na liniju koda koja se nalazi nakon cjelokupne selekcije.

Selekcije se mogu javljati i jedna unutar druge i tako tvoriti složenije programske konstrukcije. Takve se selekcije nazivaju ugnježdenim selekcijama. Sljedeći primjer prikazuje dvije ugnježdene selekcije tipa `if...else`.

```
if (uvjet 1)
   {blok 1}
else if (uvjet 2)
   {blok 2}
else
   {blok 3}
```

Navedeni primjer čitamo, **if** (ako) je uvjet 1 istinit idemo na smjer 1, ukoliko je uvjet 2 istinit idemo na smjer 2 a u svakom drugom slučaju idemo na smjer 3.

Kada već govorimo o ugnježdenim selekcijama pogledajmo sljedeći primjer

```
if (uvjet 1)
   {blok 1}
else if (uvjet 2)
   {blok 2}
else
   {blok 3}
```

Prije nego promotrimo nekoliko programskih primjera korištenja selekcije moramo se osvrnuti na formiranje jednostavnih logičkih izraza unutar selekcije, odnosno izraza uspoređivanja.

Prilikom formiranja uvjeta koriste se dva ili više operanda (nešto nad čime vršimo operaciju) i jedan ili više operatora (definira operaciju koja se provodi nad operandima) uspoređivanja.

Uzmimo da su A i B dva izraza (operanda). A op B gledamo kao operaciju usporedbe (op) između dva izraza.

Postoji nekoliko mogućih operacija uspoređivanja između dva izraza i one su navedene u sljedećoj tablici.

Tabela 1: Popis operatora uspoređivanja

Operator	Primjer	Opis
<	A < B	A manje od B
<=	A <= B	A manje ili jednako B
== (dva znaka jednakosti)	A == B	A jednako B
>	A > B	A veće od B
>=	A >= B	A veće ili jednako B
!=	A != B	A različito od B

Operacije uspoređivanja kao operande (izraze) uzimaju varijable nekog tipa podataka i vraćaju 1 (**`true`**) ako je izraz istinit ili 0 (**`false`**) ako izraz nije istinit. Rezultati logičkih izraza su **`short`** tipovi podataka gdje 0 označava laž a 1 ili bilo koji drugi broj istinu. Logički tip podataka koji može pohraniti istinu (**`true`**) ili laž (**`false`**) je **`bool`** tip podatka.

```
bool A = true;
```

Operatore uspoređivanja nazivamo još i relacijski operatori.

Na ovaj način, koristeći operatore uspoređivanja možemo formirati jednostavan logički izraz.

U nastavku slijedi nekoliko programskih primjera uvjeta i **`if`** selekcije.

```
if (A == 3) {
   cout << "Tri";
}
```

Ovo je najjednostavniji uvjet. Ukoliko je A jednako 3, ispisuje se odgovarajuća poruka a ukoliko to nije istina ne događa se ništa.

```
if (A == 3) {
   cout << "Tri";
}
else {
     cout << "Nije tri";
}
```

Napomenimo ovdje još jedan detalj s kojim ćete se vrlo brzo susresti čitajući programske kodove – ako se programski blok sastoji samo od jedne jedine naredbe, onda ga nije potrebno stavljati u vitičaste zagrade. Dakle, gornji se primjer može zapisati i kao

```
if (A == 3)
   cout << "Tri";
else
    cout << "Nije tri";
```

korištenjem ugnježdenih selekcija može se doći do jedinog slučaja neregularnoti programskog jezika C++, tj. do slučaja čija semantika (značenje) nije jasna iz samog koda.

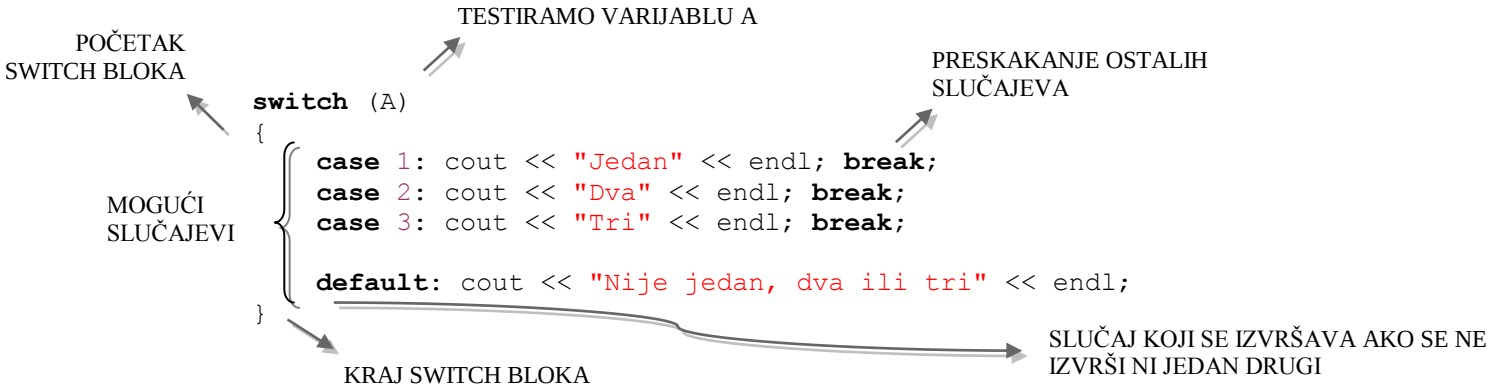
```
if (A == 0)
if (B == 0)
cout << "Da";
else
cout << "Ne";
```

Iz samog ovog koda ne može se shvatiti kada će se ispisati riječ „Ne“. Naime, sam kod ne govori odnosi li se **`else`** klauzula na prvi ili na drugi **`if`**. Drugim riječima, nije jasno hoće li se „Ne“ ispisati ako je A različit od 0 ili pak ako je A jednak 0, a B različit od 0. Programski jezik C++ nije jedini programski jezik u kojem se može izvesti ovakva neregularnost. Iz toga je razloga dogovoreno da se, ako nije jasno iz sintakse, **`else`** klauzula odnosi na posljednji otvoreni **`if`**. To u našem slučaju znači da će se **`else`** klauzula odnositi na drugi **`if`**, tj. da će se „Ne“ ispisivati ako je A jednak 0, a B različit od 0, dok se u slučaju da je A različit od 0 neće ispisivati ništa.

Dajmo još jedan primjer ugnježdenih petlji. U ovom slučaju ukoliko A nije 3, ispisuje se odgovarajuća poruka o tome (''Nije tri'').

```
if (A == 3)
   cout << "Tri";
else if (A == 4)
   cout << "Četiri";
else
   cout << "Nije tri ni četiri";
```

Osim **`if`** selekcije, postoji i selekcija tipa **`switch`**. **`Switch`** je programski konstrukt čija uporaba je slična uporabi **`if`** selekcije sa nekoliko **`if`** uvjeta, samo što je nešto lakša za implementaciju. Selekcija tipa **`switch`** testira vrijednost varijable i za svaku pojedinu vrijednost izvodi pripadajući kod.



Treba napomenuti da se u selekciji tipa `switch`, za razliku od prije objašnjenih selekcija, u zagradi ne nalazi logički već aritmetički, i to cjelobrojni izraz. Ovaj se izraz uspoređuje po jednakosti s vrijenostima navedenim u klauzulama i izvršava se onaj slučaj čija je vrijednost jednaka izrazu u zagradama. Navedeni primjer testira vrijednost varijable A i ukoliko je ona 1, 2 ili 3 ispisuje odgovarajuću poruku. Ukoliko je vrijednost varijable van tih vrijednosti i nema

slučaja (**case**) koji bi pokrio tu vrijednost (npr. vrijednost 4), izvršit će se **default** slučaj koji se izvršava u slučaju da se pojavi vrijednost za koju nije definiran **case** (slučaj). Slučaj default nije obavezno navoditi, no ako se navodi, mora biti jedan jedini i mora se nalaziti iza posljednjeg case slučaja. Naredba break služi za prekidanje rada bilo kojeg složenijeg programskog konstrukta osim selekcije tipa **if** i **if...else**. Iako se naredbe skoka, pa tako i naredba break u ostalim slučajevima trebaju izbjegavati, ova je naredba nužna kod selekcije tipa **switch**. Da bismo to objasnili, moramo objasniti kako u programskom jeziku C++ radi selekcija tipa **switch**. Njena je izvedba u progamskom jeziku C++ drugačija nego kod većine drugih, posebno starijih programskih jezika. Naime, u programskom jeziku C++ selekcija tipa switch se ponaša kao niz **goto** skokova. Aritmetički se izraz u zagradama tretira kao ime oznake na koju treba skočiti, dok se case slučajevi tretiraju kao same oznake. Kada se naiđe na oznaku koja po vrijednosti odgovara, skače se na nju i nastavlja se sa slijednim izvođenjem programa. Zbog toga će se bez prekidanja izvođenja naredbom break izvesti slučaj koji odgovara izrazu, ali i svi slučajevi koji se nalaze iza njega. Kako bismo to izbjegli, koristimo naredbu **break** kojom završava svaki slučaj i kojom se iskače iz **switch** konstrukta. Naravno, u posljednjem slučaju, bio on **case** ili **default**, nije potrebno navoditi **break** naredbu, jer se iza njega ne nalazi ništa, pa se nema što preskočiti. Ovakav način rada je i razlog zog čega blokove naredi vezane uz pojedini slučaj **switch** selekcije nije potrebno

Naravno, postoje i slučajevi kada nam odgovara da se izvođenje **switch** selekcije ne prekida nakon izvođenja koda nekog slučaja, već da se izvrše. Recimo da imamo problem u kojem korisnik zadaje broj između 1 i 10 i program treba ispisati zbroj svih brojeva od 1 do tog broja. Ovaj se problem može lako riješiti iteracijom, koju ćemo učiti u sljedećoj lekciji, ali se isto tako može riješiti pomoću **switch** selekcije. Okrenemo li slučajeve naopako, od 10 do 1 i ako u svakom slučaju na sumu pribrojimo broj koji odovara tom slučaju to će riješiti naš problem.

```
cout << "Upisite N: ";
cin >> N;
short S = 0;
switch (N) {
        case 10: S = S + 10;
        case 9: S = S + 9;
        case 8: S = S + 8;
        case 7: S = S + 7;
        case 6: S = S + 6;
        case 5: S = S + 5;
        case 4: S = S + 4;
        case 3: S = S + 3;
        case 2: S = S + 2;
        case 1: S = S + 1;
}
cout << "S = " << S << endl;
```

Nadalje, ponekad se više slučajeva može riješiti zajedno, istim kodom. Programki jezik C++ ne dozvoljava složene slučajeve. No, kako slučajevi predstavljaju samo oznake na koje se skače, pojedini slučajevi mogu biti i prazni, tj. za njih se ne mora definirati ni jedna naredba, već se može preskočiti na sljedeći slučaj. Na taj se način mogu izvesti slučajevi koji obuhvaćaju više različitih vrijednosti. Ako, na primjer želimo učitati slovo i ispisati radi li se o suglasniku ili samoglasniku, to možemo učiniti kako je prikazano u sljedećem kodu

```
char C;
cout << "Unesite slovo";
switch (C) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': cout << "Samoglasnik" << endl;
        default: cout << "Suglasnik" << endl;
}
```

Aritmetički operatori

Aritmetički operatori su operatori koji vrše aritmetičke operacije nad jednim ili dva operanda. Ukoliko se operacija izvodi nad jednim operandom govorimo o unarnim operatorima a ukoliko se operacija vrši nad dva operanda govorimo o binarnim operatorima.

Aritmetički operatori uključuju sljedeće operacije:

- A+B aritmetički izraz
- A-B aritmetički izraz
- A*B aritmetički izraz
- A/B aritmetički izraz
- A%B aritmetički izraz (modulo)

Operacije zbrajanja, oduzimanja, množenja i dijeljenja ne zahtijevaju dodatna pojašnjenja no reći ćemo nešto dodatno o operaciji modulo. Modulo je operacija koja vraća ostatak cjelobrojnog dijeljenja.

```
4 % 2 = 0
U ovom slučaju 4 / 2 = 2 i ostatak je 0.

5 % 2 = 1
U ovom slučaju 5 / 2 = 2 i ostatak je 1.
```

Ukoliko želimo konkretnu formulu koja će nam iz dijeljenja dati rezultat operacije modulo, možemo koristiti sljedeći postupak.

```
5 / 2 = 2,5
Decimalni dio rezultata: 0,5
0,5 * 2 (djeljitelj) = 1 (ostatak tj. rezultat operacije modulo).
```

Kao što smo već rekli, aritmetički operatori mogu biti binarni ali i unarni.

```
A = 3;
B = 1;
cout << A + B;          + je u ovom slučaju binarni operator koji zbraja 2
                        broja
```

Aritmetički izrazi mogu biti i složeniji nego što u gore opisani. Zapravo, u programskom jeziku C++ mogu se zapisati svi aritmetički izrazi kao u matematici, a i još više od toga.

Još uvijek nismo opisali sve mogućnosti aritmetičkih izraza u progamskom jeziku C++, no dat ćemo djelomičnu definiciju načina njihove gradnje:

- Konstanta, varijabla i funkcija koja vraća numeričku vrijednost je aritmetički izraz
- Ako je A aritmetičk izraz, onda je i
  - -A aritmetički izraz
  - (A) aritmetički izraz
- Ako su A i B aritmetički izrazi onda je i
  - A+B aritmetički izraz
  - A-B aritmetički izraz
  - A*B aritmetički izraz
  - A/B aritmetički izraz
  - A%B aritmetički izraz

Istaknimo posebno definiciju da iz činjnice da je A aritmetički izraz slijedi da je i (A) aitmetički izraz. Ovime u aritmetičke izraze uvodimo zagrade. Dakle, kao i u matematici, u programskom jeziku C++ aritmetički izrazi mogu sadržavati zagrade, ali za razliku od matematike u programskom jeziku C++ sve zagrade moraju biti okrugle.

Nadalje, rečeno je da su funkcije koje vraćaju numeričku vrijednost također aritmetički izrazi. To se posebno odnosi na funkcije koje se vrlo često koriste, a koje se nalaze u biblioteci cmath. Biblioteka **`cmath`** je biblioeka koja sadrži standardne matematičke funkcije. Biblioteka **`cmath`** u sebi sadrži čitav niz funkcija, a mi ovdje navodimo samo najvažnije.

- sin(x) – sinus
- cos(x) – kosinus
- tan(x) – tangens
- asin(x) – akrus sinus
- acos(x) – arkus kosinus
- atan(x) – arkus tangens
- exp(x) – e^x
- log(x) – prirodni logaritam
- log10(x) – dekadski logaritam
- pow(x,y) – x^y
- sqrt(x) – kvadratni korijen

Ono što treba napomenuti da sve funkcije biblioteke `cmath` zahtijevaju da barem jedan njihov argument bude decimalan broj.

Na kraju spomenimo još jedan aritmetički operator, jedini ternarni operator u programskom jeziku C++. To je takozvani uvjetni operator koji omogućuje da se između dvaju aritmetičkih izraza izabere jedan, s obzirom na logičk uvjet. Drugim riječima, ovaj operator predstavlja **if**…**else** selekciju u aritmetičkim izrazima. Njegova je sintaksa


```
logički_izraz ? aritmetički_izraz_1 : aritmetički_izraz_2
```

Ako je logički izraz ispunjen, onda će se kao rješenje ove operacije izračunavati prvi aritmetički izraz, a u suprotnom će se izračunavati drugi.

Navedimo primjer. Želimo li da korisnik upiše dva decimalna broja, a da program ispiše veći od njihova zbroja i umnoška napravit ćemo sljedeći program

```
float   A, B;
cout << "A = ";
cin >> A;
cout << "B = ";
cin >> B;
cout << (A * B > A + B ? A*B : A + B) << endl;
```

Zadaci za vježbu

1. Definirajte nekoliko životnih situacija u kojima je potrebno koristiti selekciju kao sredstvo odlučivanja.

2. Napišite programski kod za nekoliko primjera selekcija tipa **`if`** i **`switch`** po vlastitom izboru.

3. Odgovorite na pitanje što je to selekcija tipa **`if...else if...else`**.

4. Odgovorite na pitanje što su to relacijski operatori.

Programski primjeri za laboratorijske vježbe

Zadatak 1

Napišite programski kod koji radi sljedeće.

1. Traži od korisnika unos broja od 1 do 7 te ispisuje korisniku odgovarajući dan u tjednu (koristiti selekciju tipa switch).
2. Ukoliko korisnik unese broj van zadanog raspona (1-7), potrebno je ispisati da taj dan u tjednu ne postoji (napraviti modifikaciju točke 1).
3. Traži od korisnika unos brojeva 1-5 i ispisuje korisniku uneseni broj slovima (koristiti selekciju tipa if...else if...else).
4. Traži unos cijelog broja između 1 i 10 te nakon toga ispisuje na ekran uneseni broj i slijedno sve brojeve do broja 10 (koristiti selekciju tipa switch). Primjerice za 5 program ispisuje 5, 6, 7, 8, 9, 10. Ukoliko korisnik unese broj van zadanog raspona ispisuje se poruka da taj broj nije podržan.
5. Traži od korisnika unos jednog decimalnog broja. Ispituje da li je broj između 10 i 20 i ukoliko je, ispisuje korisniku da je broj unutar navedenog raspona. Ukoliko uneseni broj nije u navedenom rasponu program ispisuje korisniku poruku o tome, zajedno sa cijelim dijelom unesenog broja u decimalnom, heksadecimalnom i oktalnom obliku.