Iteracije i složeni logički izrazi

Ciljevi:

- proučiti i shvatiti pojam iteracija
- proučiti tipove iteracija
- shvatiti uporabu iteracija
- proučiti uporabu složenih logičkih izraza

Pregled lekcije

Iteracije s logičkim uvjetom

Prošli smo puta obradili iteraciju koja se temelji na eksplicitnom brojaču. Međutim, u programiranju često ne znamo broj koraka koje iteracija mora izvršiti, već samo uvjet zaustavljanja, tj. uvjet koji mora biti ispunjen da bi se iteracija izvršavala.

Na primjer, pogledajmo sljedeći primjer. Korisnik unosi pozitivne cijele brojeve, sve dok ne unese broj 1. Potrebno je ispisati koliko je od tih brojeva prostih

Primjer 1:
ULAZ: Niz pozitivnih cijelih brojeva
IZLAZ: Broj unešenih prostih brojeva

1. P = 0
2. Upiši N
3. Ako je N prost P++
4. Sve dok je N != 1 radi
5. Unesi N
6. Ako je N prost P++
7. Ispiši P

Ovaj se problem, kao uostalom i svi problemi koji zahtjevaju iteraciju može riješiti pomoću iteratije tipa `for` na sljedeći način.

Primjer 1.1

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

int main () {
    unsigned int P = 0, N;
    bool prost = true;
    cout << "N = ";
    cin >> N;
    for (int I=2; I<=sqrt(N); I++)
```

```
            if (!(N%I)) prost = false;
        if (prost) P++;
        for ( ; N != 1; ) {
            cout << "N = ";
            cin >> N;
            prost = true;
            for (int I=2; I<=sqrt(N); I++)
                if (!(N%I)) prost = false;
            if (prost) P++;
        }
        cout << "Uneseno je " << --P << " prostih brojeva" << endl;
        system("pause");
        return 0;
}
```

Jedino pitanje koje bi moglo biti nejasno jest zašto se P prije ispisa smanjuje za 1. Razlog tome je što u programu nismo posebno napravili kontrolu koja bi prilikom unosa broja 1 preskočila provjeru je li broj prost, tako da će u P biti uvršten i posljednji unos broja 1.

Iako je iteracija tipa `for` u programskom jeziku C++ definirana tako da se pomoću nje mogu riješiti svi problemi, prvenstveno zbog usklađenost s drugim programskim jezicima u programskom jeziku C++ implementirane su i specijalne iteracije koje se koriste kada nema brojača na koji bi se vezalo izvršavanje petlje. Upravo te vrste iteracija ćemo obrađivati u ovoj lekciji.

Prva ovakva iteracija je iteracija tipa **`while`**. Ona se izvršava sve dok je logički uvjet koji je zadan u njenoj glavi istinit. Ova iteracija ima sintaksu

while (logički uvjet)
  blok naredbi

Primjenom iteracije tipa `while`, gornji problem možemo riješiti na sljedeći način.

Primjer 1.2

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

int main () {
    unsigned int P = 0, N;
    bool prost = true;
    cout << "N = ";
    cin >> N;
    for (int I=2; I<=sqrt(N); I++)
        if (!(N%I)) prost = false;
    if (prost) P++;
    while (N != 1) {
        cout << "N = ";
        cin >> N;
        prost = true;
        for (int I=2; I<=sqrt(N); I++)
            if (!(N%I)) prost = false;
```

```
        if (prost) P++;
    }
    cout << "Uneseno je " << --P << " prostih brojeva" << endl;
    system("pause");
    return 0;
}
```

Naše rješenje ima nekoliko problema koje treba riješiti. Prvi, koji i nije problem, već samo nezgrapnost, taj što smo prvi broj u nizu morali posebno unositi i provjeravati njegovu prostost. Ovaj se problem može riješiti primjenom druge iteracije s logičkim uvjetom – **do...while** iteracije. Njena je sintaksa.

do
    blok naredbi
while (logički uvjet); - do while je jedina iteracija iza čijeg kraja ide znak ;

Razlika između ove dvije iteracije je u tome što se kod iteracije tipa **do...while** logički uvjet provjerava tek nakon prvog prolaska kroz tijelo petlje. Dakle, dok se kod iteracija tipa **for** i **while** može dogoditi da se tijelo petlje ne izvrši ni jednom, kod iteracije tipa **do...while** tijelo se petlje mora izvršiti najmanje jedamput.

Primjer 1.3

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

int main () {
    unsigned int P = 0, N;
    bool prost = true;
    do {
        cout << "N = ";
        cin >> N;
        prost = true;
        for (int I=2; I<=sqrt(N); I++)
            if (!(N%I)) prost = false;
        if (prost) P++;
    } while (N != 1);
    cout << "Uneseno je " << --P << " prostih brojeva" << endl;
    system("pause");
    return 0;
}
```

Ovo rješenje naslijedilo je još jedan problem koji smo istakli i u prethodnim lekcijama, a koji je vezan uz provjeru prostosti broja. Naime, ako jednom varijabla `prost` postane **false**, petlja se ne mora dalje izvoditi. U našem rješenju pak se ona izvodi sve do `sqrt(N)` bez obzira jesmo li već prije zaključili da broj nije prost.

Kako bismo rješili ovaj problem, u **for** petlji kojom se ispituje prostost broja trebali bismo dodati još jedan uvjet završetka petlje – kada varijabla `prost` postane `false`. No, petlja može imati samo jedan logički uvjet. Rješenje ovog problema leži u složenim logičkim izrazima, kojima se dva ili više izraza uspoređivanja spajaju u složeniji logički izraz. U našem

slučaju `for` petlja se mora izvršavati sve dok je `I<sqrt(N)` i dok je `prost=` **`true`**. Ovo upućuje na logičku konjunkciju (logički i), koji se u programskom jeziku C++ označava s &&.

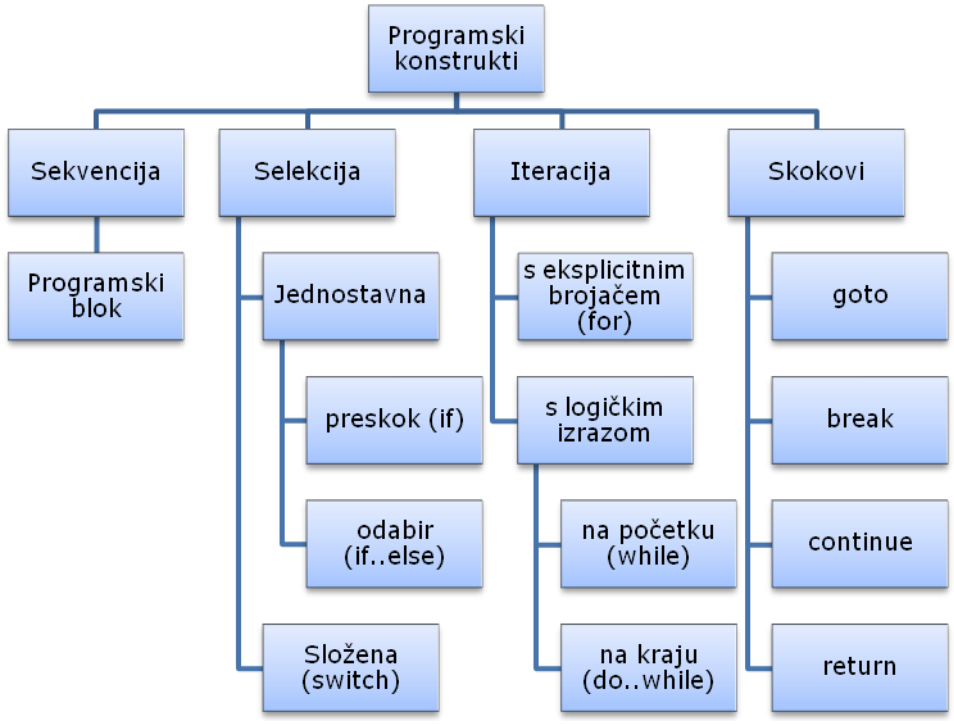
Sada gornji primjer možemo riješiti kao

Primjer 1.4

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

int main () {
    unsigned int P = 0, N;
    bool prost = true;
    do {
        cout << "N = ";
        cin >> N;
        prost = true;
        for (int I=2; I<=sqrt(N) && prost; I++)
            if (!(N%I)) prost = false;
        if (prost) P++;
    } while (N != 1);
    cout << "Uneseno je " << --P << " prostih brojeva" << endl;
    system("pause");
    return 0;
}
```

Time smo obradili sve osnovne programske konstrukte programskog jeszika C++ i dajemo njihov pregled



Slika 1: Programski konstrukti

Logički operatori

Kao što smo mogli vidjeti u prethodnim primjerima izvođenje **`while`** iteracije ovisi o ispunjenosti/neispunjenosti logičkog uvjeta koji se nalazi unutar okruglih zagrada odmah nakon **`while`** izraza. U navedenim primjerima radilo se o jednostavnim logičkim izrazima no oni mogu biti i puno kompleksniji.

Složeni logički izrazi se tvore od jednostavnijih pomoću logičkih operatora. Pod jednostavnijim logičkim izrazima podrazumijevamo izraze uspoređivanja koje smo do sada koristili.

Postoji 3 logička operatora koja koristimo da bismo formirali složene logičke izraze.

Logički operatori su:

- !A - negacija
- A && B - konjunkcija (logičko „i“)
- A || B - disjunkcija (logičko „ili“)

Slijede istinosne tablice svakog pojedinog logičkog operatora.

A	!A
0	1
1	0

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

Kao što možemo vidjeti iz prethodnih tablica vrijedi sljedeće:

- Negacija izvrće logičku vrijednost
- Disjunkcija je istinita ako je istinit jedan ili drugi argument
- Konjunkcija je istinita ako je istinit jedan i drugi argument

Pogledajmo nekoliko primjera.

Uzmimo da je A = 1, B = 0 i C = 5

A>B && !(C==B) će biti istinit zato što je A veće od B a C nije jednako B što će opet dati istinu je izraz C==B ispred sebe ima negaciju. Dakle, izraz je istinit ako je A veće od B a C nije jednako B. Dakle, gledajući cijeli izraz koji se sastoji od dva jednostavna izraza koja su oba istinita je u cjelosti istinit je su jednostavni izrazi povezani operatorom logičkog „i“, što znači da oba jednostavna izraza moraju biti istinita da bi cjelokupan izraz bio istinit što je u ovom slučaju i točno.

A==B || C>B će također biti istinit jer iako je izraz A==B neistinit, izraz C>B je istinit. S obzirom da je između dva jednostvna izraza operator logičko „ili“, dovoljno je da samo jedan od jednostavnih izraza bude istniti da bi cjelokupan izraz bio istinit što je u ovom slučaju i tako.

A==B && C>B je indentičan prethodnom izrazu, osim što se koristi operator logičko „i“ tako da oba jednostavna izraza moraju biti istinita da bi cjelokupan izraz bio istinit. To u ovom slučaju nije tako je izraz A==B nije istinit pa je i cjelokupan izraz neistinit tj. lažan.

Pitanje je kako će se interpretirati izraz **A>B || B>C && C==1**. Jasno je da je prvi izraz uspoređivanja istinit, dok su preostala dva lažna. No istinitost cjelokupnog logičkog izraza ovisit će o tome koji će se od logičkih operatora prvi izvršiti. Izvrši li se prvo disjunkcija, pa onda konjunkcija, cijeli će izraz biti lažan. S druge strane, izvrši li se prvo konjunkcija, a onda disjunkcija, cijeli će izraz biti istinit. U programskom se jeziku C++, kao uostalom i u svim drugim programkim jezicima, konjunkcija ima veći prioritet od disjunkcije. Dakle, prvo će se izvršiti konjunkcija i cijeli će ovaj izraz biti istinit.

Drugi primjer koji ilustrira koliko je potrebno biti pažljiv kod izgradnje logičkih izraza je izraz **!C==1**. Iako bi se na prvi pogled učinilo da će ovaj izraz biti istinit, tome neće biti tako. Naime, operator negacije ima veći prioritet od operatora ==. Tako da će se prvo interpretirati !C, a kako je C=5, dakle **true**, !C će biti 0, a 0==1 je lažno. Želimo li provjeriti je li C različit od 1, treba pisati !(C==1) ili C!=1.

Iz navedenih primjera možemo zaključiti da postoji određeni prioritet operatora u logičkim izrazima. Prioriteti logičkih operatora su sljedeći:

1. !
2. <, <=, >=, >
3. ==, !=
4. &&
5. ||

Slično kao za aritmetičke izraze i za logičke izraze možemo definirati pravila izgradnje:

Neka su A i B jednostavni logički izrazi. Tada je

1. (A) logički izraz
2. A && B logički izraz
3. A ||B logički izraz
4. !A logički izraz

Pogledajmo primjer u kojemu je potrebno napraviti program koji će izračunati broj *e* na točnost 10^{-10}.

Prije no što napišemo pseudokod rješenja ovog problema, pogledajmo matematičku osnovu rješenja. Broj e je baza prirodnog logaritma i izračunava se prema formuli

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots.$$
$$= \sum_{i=0}^{\infty} \frac{1}{i!}$$

Dakle, svakim novim članom rea bit ćemo sve bliže rješenju. No, pitanje je kako ćemo doznati da smo postigli traženu točnost. Naime, ako je neki član reda manji od tražene točnosti ε, to još uvijek ne garantira da smo postigli traženu točnost. Nećemo ovdje razvijati matematičku teoriju provjere točnosti izračuna, no zna se da je suma prvih *N* članova ovog reda aproksimira *e* s točnošću ε ako je

$$\frac{1}{(N+1)!} \leq \frac{\varepsilon}{2}$$

Nadalje, kako ne bismo morali računati faktorijel za svaki pojedinačni član niza, čuvat ćemo faktorijele ćemo računati po formuli

$$N! = (N-1)! \cdot N$$

Tako ćemo u svakom koraku jednostavnim množenjem moći izračunati faktorijel sljedećeg prirodnog broja.

Sada je pseudokod rješenja ovog problema dan kako slijedi:

Primjer 2:

Ulaz: -

Izlaz: Broj *e* s točnošću 10^{-10}

1. epsilon = 1E-10
2. e=1
3. Fakt = 1
4. I = 1
5. Sve dok je 1/(Fakt*I)>epsilon/2
6. Fakt = Fakt * I
7. e = e + 1/(Fakt
8. I++
9. ispiši e

Odnosno u kodu

Primjer 2.1:

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

int main () {
     const double epsilon = 1E-10;
     double e = 1;
     int I = 1, Fakt = 1;
     while (1./(Fakt*I)>epsilon/2) {
           Fakt *= I++;
           e += 1./Fakt;
     }
     cout << setprecision(11) << "e = " << e << endl;
     system("pause");
     return 0;
}
```

U ovom smo primjeru koristili deklaraciju konstante `epsilon`. Konstante se deklariraju isto kao i varijable, osim što se ispred imena tipa podataka stavlja ključna riječ `const`. Nadalje, kod deklaracije konstatne uvijek se mora navesti i inicijalna vrijednost nakon njenog identifikatora. Drugim riječima, konstanta se uvijek prilikom deklaracije mora i inicijalizirati. U programiranju je važno štedjeti memorijski prostor. Konstanta, za razliku od varijable, u izvršnom kodu neće alocirati memorijski prostor, već će se prilikom prevođenja svaka njena pojava jednostavno zamijeniti njenom vrijednošću. Stoga je važno da se vrijednosti koje se u programu ne mijenjaju ne deklariraju kao varijable, već kao konstatne.

Prethodni primjer smo mogli jednostavnije riješiti korištenjem iteracije tipa **`for`** umjesto one tipa **`while`**.

Primjer 2.2:

```
#include <iostream>
#include <iomanip>
#define epsilon 1E-10
using namespace std;

int main () {
     double e = 1;
     int Fakt = 1;
     for (int I = 1; 1./(Fakt*I)>epsilon/2; I++) {
           Fakt *= I;
           e += 1./Fakt;
     }
     cout << setprecision(11) << "e = " << e << endl;
     system("pause");
     return 0;
}
```


Ovaj nam je primjer poslužio da pokažemo još jedan način deklariranja i iniciranja konstanti – pomoću preprocesorske naredbe `#define`. Iako je rezultat jednak, ovakav se način deklariranja i iniciranja konstatni razlikuje od prethodnog u tome kada će se tijekom postupka prevođenja provesti. Naime, konstatna kako je definirana u primjeru 2.1

razjrešavat će se tijekom postupka prevođenja, dok će se konstanta kako je definirana u posljednjem primjeru razrješavati prije prevođenja, u postupku preprocesiranja, odnosno provjere sintaksne konzistentnosti programa i uključivanja biblioteka.

Zadaci za vježbu

1. Napišite program koji traži od korisnika unos jednog broja i taj unos se ponavlja sve dok korisnik ne unese broj 5. Rješenje realizirati koristeći prvo `do while` pa onda `while` petlju.

2. Napišite program koji u `while` petlji ispisuje brojeve od 10 do 20.

3. Odgovorite na pitanje što je to iteracija.

4. Odgovorite na pitanje što su to logički operatori.

Programski primjeri za laboratorijske vježbe

Zadatak 1.

Izradite program u kojem:

1. Korisnik unosi jedan cijeli broj između 8 i 16. Ukoliko korisnik unese broj van zadanog raspona javlja se poruka greške. Ukoliko je uneseni broj u zadanom rasponu, koristeći while petlju, ispisuju na ekran svi brojevi od 5 pa sve do unesene vrijednosti (za 7 se ispisuje 5, 6, 7).

2. Korisnik unosi dva cijela broja (C i D) između 10 i 20 ili između 50 i 70. Ukoliko korisnik ne unese oba broja ispravno traži se ponovni unos oba broja. Ukoliko je korisnik unio ispravne brojeve na ekran se ispisuju rezultati sljedećih operacija:
   - A + B
   - A % B
   - A – B
   - A++
   - Suma rezultata svih prethodnih operacija

3. Unosi prirodni broj N i N velikih slova. Program treba ispisati koliko je suglasnika i samoglasnika upisano.

4. Korisnik unosi prirodne brojeve. Za svaki unos se treba provjeriti je li broj dobro unesen. Računalo treba ispisati najveću zajedničku mjeru unesenih brojeva. Najveća zajednička mjera više brojeva se izračunava tako da se najprije izračuna najveća zajednička mjera prva dva unesena broja, nakon toga najveća zajednička mjera trećeg unesenog broja i najveće zajedničke mjere prvih dvaju itd. S unosom se završava kada korisnik unese broj 0.

5. Korisnik i računalo igraju igru par-nepar. Na početku igre igrač bira hoće li biti par ili nepar. Nakon toga igrač i računalo istovremeno deklariraju proizvoljni broj između 1 i 5. Računalo ne smije svoj broj temeljiti na korisnikovom unesenom broju, već ga mora birati slučajno. Potrebno je provjeriti korektnost korisnikova unosa. Deklarirani brojevi se zbrajaju i ako je korisnik pogodio hoće li broj biti paran ili neparan, dobiva bod, a u suprotom bod dobiva računalo. Pobjednik je onaj tko prvi dobije 3 boda. Na kraju igre treba ispisati tko je dobio igru i s kojim rezultatom.