## 3. Izrazi i kontrolne naredbe

### Nabrojati smjernice za pravilno korištenje *if* naredbe.

- Prvo napisati glavni (nominalni) slučaj, a tek zatim manje česte slučajeve
- Provjeriti jesu li operatori usporedbe (`<`, `>`, `<=`, `>=` itd.) dobro upotrijebljeni
- Napisati glavni (nominalni) slučaj uz `if`, radije nego uz `else`

### Koje su karakteristike `switch-case` naredbe?

- Razlikuje se od programskog jezika do programskog jezika
  - C++, Java, C# – `case` uzima po jednu vrijednost
  - Visual Basic – `case` može primiti i niz vrijednosti
- Mnogi skriptni jezici nemaju `switch-case`

### Ukoliko ima mnogo `case` grana, na koje načine ih je moguće poredati?

- abecedno ili numerički uzlazno
- staviti glavni slučaj prvi
- po frekvenciji korištenja

### Objasniti propadanje kroz `case` grane i zašto je to bolje izbjegavati?

Do propadanja dolazi kada se na kraju `case` grane ne nalazi naredba `break`. U tom se slučaj izvođenje nastavlja sljedećom granom. Propadanje je bolje izbjegavati jer ugnježđuje kontrolne naredbe, otežava modifikacije i kod čini manje čitljivim, pogotovo ako propadanje nije dokumentirano.

### Preurediti sljedeći programski odsječak pazeći na sve naučeno u prva tri tjedna predavanja:

```
if(a == NULL)
  return 0; //action not defined
else {
  if (a == "deleteuser")            du();
  else if (a == "adduser")          au();
  else if(a == "addcategory")       ac();
  else if(a == "deletecategory")    dc();
}
```

```
switch (action) {
  case "adduser":
    addUser();
    break;

  case "deleteuser":
    deleteUser();
    break;

  case "addcategory":
    addCategory();
    break;
```

```
  case "deletecategory":
    deleteCategory();
    break;

  default:
    // action not defined
    return 0;
}
```

## 4. Petlje

Što je to pojava pomiješanih indeksa?

Korištenje jednog te istog indeksa u dvije namjene.

U programskom jeziku C++ napisati program koji neprestano učitava brojeve sve dok se ne unese broj kojem su sve znamenke parne (koristiti `while` petlju s logičkim testom izlaza u sredini).

```
#include <iostream>

int main() {
  while (true) {
    int broj;
    std::cin >> broj;

    bool sveParne = true;
    while (broj != 0) {
      int znamenka = broj % 10;
      if (znamenka % 2 != 0) {
        sveParne = false;
        break;
      }
      broj /= 10;
    }

    if (sveParne) {
      break;
    }
  }
  return 0;
}
```

U programskom jeziku C++ napisati program koji od korisnika traži unos 10 brojeva te ispisuje najvećeg i najmanjeg (paziti na sve do sada naučeno: imena varijabli, magične brojeve i slično).

```
#include <iostream>

int main() {
  const int BROJ_UNOSA = 10;

  int najmanji, najveci;
  for (int i = 0; i < BROJ_UNOSA; ++i) {
    int broj;
    std::cin >> broj;

    if (i == 0 || broj < najmanji) {
      najmanji = broj;
    }
    if (i == 0 || broj > najveci) {
      najveci = broj;
    }
  }
  std::cout << najmanji << ", " << najveci << '\n';
  return 0;
}
```

U programskom jeziku C++ napisati program koji učitava matricu 3x4, i zatim je modificira tako da svakom elementu matrice pribroji umnožak retka i stupca u kojem se on nalazi (paziti na pojavu pomiješanih indeksa).

```
#include <iostream>

int main() {
  const int BROJ_REDAKA = 3;
  const int BROJ_STUPACA = 4;

  int matrica[BROJ_REDAKA][BROJ_STUPACA];
  for (int redak = 0; redak < BROJ_REDAKA; ++redak) {
    for (int stupac = 0; stupac < BROJ_STUPACA; ++stupac) {
      std::cin >> matrica[redak][stupac];
    }
  }


  for (int redak = 0; redak < BROJ_REDAKA; ++redak) {
    for (int stupac = 0; stupac < BROJ_STUPACA; ++stupac) {
      matrica[redak][stupac] += redak * stupac;
    }
  }

  std::cout << '\n';
  for (int redak = 0; redak < BROJ_REDAKA; ++redak) {
    for (int stupac = 0; stupac < BROJ_STUPACA; ++stupac) {
      std::cout << matrica[redak][stupac] << ' ';
    }
    std::cout << '\n';
  }

  return 0;
}
```

## 5. Postupci

Koje vrste kohezije postoje? Uz svaku vrstu kohezije napisati postupak u kojem je ona prisutna.

```
// Funkcionalna kohezija, postupak obavalja jedan
// i samo jedan zadatak
void ispisiPozdrav() {
  printf("Pozdrav!");
}

// Sekvencijalna kohezija, zadaci se s razlogom
// moraju izvršiti u točno određenom redoslijedu
void vecera() {
  kupiSastojke();   // 1. zadatak
  nadjiZenu();      // 2. zadatak
  dobarTek();       // 3. zadatak
}
```

```
// Komunikacijska kohezija, zadaci koriste iste podatke
void ispisi(Dokument dok) {
  ispisiCrnoBijelo(dok);
  ispisiUBoji(dok);
}

// Vremenska kohezija, zadaci se izvršavaju u isto vrijeme
void gasenjeSustava() {
  zatvoriDatoteke();
  oslobodiMemoriju();
  naruciPizzu();
}

// Proceduralna, zadaci se bez razloga moraju izvršiti u
// točno određenom redoslijedu
void wtf() {
  ovoMoraPrvo();
  aOvoDrugo();
  aliNitkoNeZnaZasto();
}

// Logička kohezija, više zadataka se ugura u jedan postupak
// te se parametrom postupka odabire koji se od njih izvršava
void koloSrece(int sta) {
  if (sta == 7) {
    slovoB();
  } else if (sta == -13) {
    slovoZ();
  } else {
    kupiSamoglasnik();
  }
}

// Slučajna kohezija, zadaci nemaju ništa zajedničko
void utorak() {
  rijesiZadacu();
  obrijPazuhe();
}
```

## Nabrojati osnovne smjernice za pisanje postupaka i korištenje parametara postupaka.

- Ime postupka mora opisivati ono što postupak radi
- Imena moraju biti smislena te se postupke ne bi smjelo razlikovati samo po brojevima
- Postupak bi trebao obavljati samo jedan točno određeni zadatak

- Postaviti parametre u sljedećem redoslijedu: 1: ulazni – 2: oni koji se modificiraju – 3: izlazni
- Postupak mora koristiti sve svoje parametre
- Postaviti statusne varijable ili varijable koje signaliziraju pogrešku kao zadnje parametre
- Dokumentirati pretpostavke koje postupak ima o vrijednostima koje se primaju.
- Ograničiti broj parametara postupka na otprilike 7

Odrediti koja se vrsta kohezije javlja u danom programu

```
void DoAction(string action){
  switch(action){
    case "RegisterUser":
    //code to register user here...
    break;
    case "SendWelcomeEmail":
    //code to send email here...
    break;
    case "UpdateUserPermissions":
    //code to update user permissions...
    break;
  }
}
```

Logička kohezija, zadatke treba izdvojiti u zasebne postupke:

```
void RegisterUser() {
  // code to register user here...
}

void SendWelcomeEmail() {
  // code to send email here...
}

void UpdateUserPermissions() {
  // code to update user permissions...
}
```

## 6. Kako pisati kod

### Nabrojati smjernice za pisanje kompleksnih logičkih izraza

- Kompleksni logički izraz potrebno je prelomiti u više programskih linija
- Prelomiti na način da prva linija sama za sebe čitatelju izgleda sintaksno neispravnom

### Navesti načine na koje je navedeni programski odsječak moguće poboljšati

```
action=getAction();
int userId;User user=new User(userId);int lastActivity;
if(user==null)
return -1;
if(action=="registerUser")RegisterUser(user);
else if(action=="editPermissions")EditPermissions();
else if(action=="updateUserProfile")UpdateUserProfile(user);
else if(action=="deleteUser")DeleteUser(user);
else if(action=="logout")Logout();
else if(action=="editPermissions")EditPermissions();
else if(action=="vote")Vote();
else Login();
```

- Staviti svaku naredbu u svoj redak
- Dodati praznine oko zagrada i operatora
- Izbaciti nekorištenu varijablu `lastActivity`
- Sporedni slučaj (`user==null`) pomaknuti na kraj `if-else` lanca

- Dodati provjeru `if(action=="login")` za poziv funkcije Login
- Zadnji else iskoristiti za provjeru neispravne akcije (defenzivno programiranje).
- Ukloniti logičku koheziju (?)

## 7. Uvod u OOP

### Koja je razlika između razreda i objekata (primjer iz stvarnog života)?

Razred sadrži popis atributa od kojih će se sastajati svaki objekt i popis metoda koje se mogu pozivati nad tim objektima. Na primjer nacrt po kojemu se izrađuju stanovi je razred, dok su primjerci već izgrađenih stanova objekt.

### Što su modifikatori vidljivosti? Kako korištenjem modifikatora vidljivosti ostvarujemo učahurivanje?

Modifikatori vidljivosti određuju tko može pristupiti atributima i metodama objekta. Ako je korišten modifikator `private` onda može pristupiti samo objekt, a ako je korišten modifikator `public` mogu pristupiti svi. Učahurivanje se postiže korištenjem privatnih atributa i metoda čime se one skrivaju.

### Objasniti što su to apstrakcija i enkapsulacija.

Apstrakcija je modeliranje objekata na način da se koriste samo bitne komponente stvarnog objekta. Enkapsulacija skriva neke atribute i neka ponašanja od ostalih klasa.

### Objasniti kako se dobrom definicijom javnog sučelja ostvaruje učahurivanje.

Tako da vanjskom svijetu budu vidljive samo one metode koje opisuju ponašanje objekta, a skrivaju njegovu implementaciju.

### Ukratko objasniti nasljeđivanje i polimorfizam.

Nasljeđivanje je odnos između klasa kod kojeg se jedna klasa stvara na temelju druge tako da joj se dodaju specifični atributi i ponašanje. Polimorizam je slično ponašanja različitih klasa kada svaka klasa za sebe precizno definira zajedničko ponašanje.

## 9. Uvod u C#

Definirana je metoda `Ispis`:

```
public void Ispis(object objekt)
{
  Time t;
  int i = (int)objekt;
  if (i == 3) {
    t.DisplayTime();
  } else {
    Console.WriteLine("0 bodova");
  }
}
```

Što se događa prilikom poziva funkcije sa sljedećim argumentima? Objasniti!

Kod se neće prevesti jer varijabla `t` nije inicijalizirana. Pod uvjetom da jest:

- 4 (`int`) – uspješno će se unboxati, ispisati će se „`0 bodova`"
- „3" (`string`) – baciti će se `InvalidCastException` (`int` nije `string`)
- 3.5 (`double`) – baciti će se `InvalidCastException` (`int` nije `double`)
- 3 (`int`) – uspješno će se unboxati i pozvati će se `t.DisplayTime()`. Ako je t `null` referenca baciti će se `NullReferenceException`.

## 10. Osnove OPP-a u C#-u

Ostvariti generički razred `Pair<T, U>` koji pohranjuje dva (uparena) objekta proizvoljnog tipa, primjerice `Pair<string, int>` za `int` i `string` (proučiti C# Generics). Razred treba imati implementirati konstruktor koji prima oba argumenta i osnovne metode za dohvat oba elementa i njihovu zamjenu.

```
class Par<T, U>
{
  private T prvi;
  private U drugi;

  public Par(T prvi, U drugi)
  {
    this.prvi = prvi;
    this.drugi = drugi;
  }

  public T PrviElement
  {
    get
    {
      return prvi;
    }
  }

  public U DrugiElement
  {
    get
    {
      return drugi;
    }
  }

  public Par<U, T> ZamijeniElemente()
  {
    return new Par<U, T>(drugi, prvi);
  }

}
```