# Programiranje II

PODATKOVNI TOKOVI

Vhodne in izhodne operacije se v jeziku C++ odvijajo v tokovih (*stream*).
Tok je zaporedje zlogov, "*teče*" iz enote (tipkovnica, disk, mreža) v pomnilnik. Pri izhodni operaciji pa zlogi "*tečejo*" iz pomnilnika v enoto.

Za delo z datotekami imamo na voljo naslednje knjižnice:

- <iostream.h> - razredi za delo z zaslonom in tipkovnico
- <iomanip.h> - manipulatorji s parametri
- <fstream.h> - razredi za delo z datotekami
- <strstream.h> - dodatne operacije za delo z nizi

Enote, iz katerih beremo in na katere pišemo podatke (*cout, cerr, clog in cin*) so v knjižnici <iostream.h> definirane kot objekti. Enote *cout, cerr* in *clog* so objekti tipa *ostream*, enota *cin* pa je objekt tipa *istream*. Ti objekti so povezani s standardnimi enotami računalnika: *cout* s standardnim izhodom (ponavadi zaslon), *cin* s standardnim vhodom (ponavadi tipkovnica) *cerr* in *clog* pa s standardno enoto za napake (prav tako ponavadi zaslon).

Prekriti operator >> za branje vrne vrednost, ki jo želimo prebrati. Kadar pa na enoti, iz katere beremo, naleti na konec datoteke, vrne 0 (nepravilno). Konec datoteke je različen od sistema. Na sistemih DOS in MS Windows ga vnesemo s pritiskom na tipko <Cntrl-Z> na sistemih Unix pa s pritiskom <Cntrl-D>.

Primer: izračun vsote števil

```
#include <iostream.h>
void main ()
{
     int stevilo, vsota=0;
     cout <<"Vnesi celo število (ali end -of file za konec):";
     while (cin>>stevilo)
     {   vsota+=stevilo;
          cout <<"Vnesi celo število (ali end -of file za konec):";
     }
     cout <<"Vsota števil je: " << vsota;
}
```

Pri izpisu se tok znakov ne zapiše takoj v enoto, na katero pišemo, temveč se shranjuje v vmesnik (buffer). Podobno se zgodi pri branju iz enote. Praznenje vmesnika (flushing) je operacija, ki povzroči da se vsi znaki, ki so trenutno v vmesniku zapišejo v enoto.

## Metode za vstavljane in jemanje iz toka podatkov:

Z metodo *put()* lahko v podatkovni tok pošljemo en znak. Njen prototip je.

```
ostream& ostream ::put(char ch);
```

primer: `cout.put('a');`

Z metodo *write()* pošljemo v podatkovni tok niz znakov.

Metodi get in getline sta komplementarni metodi put in omogočata branje podatkov iz podatkvnega toka. Primeri prototipov so:

```
int istream::get ();
istream& istream::get(char&rch);
istream& istream::get(char* pch, int count, char delim = '\n')
istream& istream::getline(char* pch, int count, char delim = '\n')
```

Prva metoda *get* brez argumentov preprosto prebere en znak iz vhodnega toka in ga vrne kot rezultat metode. Druga metoda naredi isto, le da vrne argument s prenosom po referenci. Naslednji metodi *get* in *getline* imata tri parametre. Obe bereta znake iz vhodnega toka do podanega delimiterja (privzeti je '\n'). Če metodi ne najdeta tega delimiterja, pa je število prebranih znakov omejeno z največjim številom znakov, ki jih naj prebereta (drugi parameter). Metoda *getline* iz vhodnega toka prebere delimiter, metoda *get* pa delimiter pusti v vhodnem toku.

Prototipi metod *peek, ignore, putback*:

```
int istream::peek();
istream& istream::ignore(int count = 1, int delim = EOF);
istream& istream::putback(char ch);
```

Metoda *peek* vrne naslednji znak iz vhodnega toka, ne da bi ga odstranila iz toka. Uporabljamo jo, kadar moramo znak pustiti v podatkovnem toku, a vseeno želimo vedeti njegovo vrednost.

Metoda *ignore* preskoči count znakov v vhodnem toku (privzeta vrednost je en znak). Kot drugi parameter podamo še delimiter, če hočemo da se preskočijo vsi znaki do tega delimiterja.

Metoda *putback* vstavi zadnji znak, prebran z metodo *get* nazaj v vhodni tok.
Metode, ki vračajo status toka so naslednje:

- Metoda eof vrne vrednost pravilno, kadar smo pri branju iz vhodnega toka naleteli na konec datoteke.
- Metoda good vrne vrednost pravilno, kadar pri branju ni bilo napake
- Metodi bad in fail pa vrneta pravilno, kadar je pri branju prišlo do napake. Razlika med metodama fail in bad je v tem, da prva označuje, da iz vhodnega toka ni bil izgubljen noben znak in lahko branje nadaljujemo, druga pa označuje, da so se znaki izgubili iz vhodnega toka, tako, da je branje vprašljivo.

# Vhodni-izhodni manipulatorji:

## Manipulatorji brez argumentov:

| razred | manipulator | pomen |
|---|---|---|
| ios | dec | izpis števil v desetiškem številskem sistemu |
| | hex | izpis števil v šestnasjstiškem številskem sistemu |
| | oct | izpis števil v osmiškem številskem sistemu |
| ostream | endl | v izhodni tok pošlje znak '\n' |
| | ends | v izhodni tok pošlje znak '\0' |
| | flush | izprazni vmesnik izhodnega toka |
| istream | ws | odstrani ločilne znake iz začetka vhodnega toka |

## Manipultorji z argumenti:

| manipulator | pomen | primer |
|---|---|---|
| setw | nastavitev širine izpisa | cout << setw(8) |
| setfill | nastavitev znaka za zapolnjevanje | cout << setfill('*') |
| setprecision | nastavitev natančnosti izpisa števil | cout << setprecision(10) |
| setioflags | postavljanje zastavic | |
| resetioflags | brisanje zastavic | |

## Zastavice:

| | |
|---|---|
| ios::skipws | izločanje ločilnih znakov pri vnosu |
| ios::left | poravnava levo |
| ios::right | poravnava desno |
| ios::showpoint | izpis decimalne pike in decimalnih mest tudi pri celih realnih številih |
| ios::showpos | izpis predznaka tudi pred pozitivnimi števili |
| ios::scientific | izpis realnih števil v eksponentni obliki |
| ios::fixed | izpis realnih števil v fiksni obliki |

# PODATKOVNE ZBIRKE (DATOTEKE)

Če želimo podatke ohraniti tudi po koncu zagona programa za vnos, moramo podatke shraniti v podatkovno zbirko, ki je na disku. **Vsaka podatkovna zbirka ima svoje *logično ime* (npr. osebe) in svoje *fizično ime*** (npr. osebe.dat).

Za delo s podatkovnimi zbirkami moramo z direktivo #include vključiti datoteko fstream.ha, ta pa že vključije datoteko iostream.h. V datoteki so definiranirazredi, ki jih uporabljamo za ustvarjanje objektov za delo z datotekami.

| razred | vrsta datotek |
|---|---|
| ifstream | vhodne datoteke |
| ofstream | izhodne datoteke |
| fstream | vhodno/izhodne datoteke |

## Tekstovne in binarne datoteke

Če zaporedje zlogov v datotekisestavlja zloge, ki predstavljajo ASCII kod, govorimo o tekstovnih datotekah. Kadar želimo, da datoteko uporabljamo samo za shranjevanje podatkov raje uporabljamo binarne datoteke. Pri binarnih datotekah se podatki ne pretvarjajo v znake, temveč se zapišejo zlogi tako, kot so predstavljeni v pomnilniku. Pri delu z binarnimi datotekami uporabljamo metodi read() in write(). Prednost binarnih datotek je manjše število bitov za posamezne zapise.

## Sekvenčne datoteke

V sekvenčnih datotekah lahko beremo samo zlog po zlog. Velikost in obliko zloga definira programer sam.

Preden datoteko želimo uporabiti, jo moramo odpreti. To storimo s klicem metode open, ali s pomočjo uporabe konstruktorja.

Primer:

```
ofstream izhodna_datoteka;
izhodna_datoteka.open("izhodna.dat");
```

ali

```
ofstream  izhodna_datoteka("izhodna.dat");
```

Ko dostopa do datoteke ne potrebujemo več jo zapremo z ukazom:

```
izhodna_datoteka.close();
```

Privzeti argumenti konstruktorja in metode open() nastavijo drugi in tretji argument na privzete vrednosti. Če nam te ne ustrezajo, jih lahko spremenimo na naslednje vrednosti:

| | |
|---|---|
| ios::in | odpremo datoteko za branje |
| ios::out | odpremo datoteko za pisanje |
| ios::ate | odpremo datoteko za dodajanje (podatki se prvič dodajo na začetek datoteke, v naslednjih zapisovanjih pa na mesto, kamor kaže kazalec |
| ios::app | odpremo datoteko za dodajanje (podatki se zmeraj dodajajo na konec |
| ios::trunc | pri odpiranju se najprej izbriše vsebina datoteke |
| ios::nocreate | odpiranje uspe samo, če datoteka že obstaja |
| ios::noreplace | odpiranje uspe samo, če datoteka še ne obstaja |

Za pozicioniranje kazalca uporabimo metode *seekg* ali *seekp*. Obe metodi imata isti pomen, le da je prva definirana za vhodne podatkovne tokove, druga pa za izhodne podatkovne tokove. Metoda ima dva parametra: prvi parameter pove število zlogov, za koliko se naj kazalec v datoteki premakne, drugi parameter pa pove , od kod se naj začne šteti pomik. Na voljo imamo naslednje možnosti:

- ios::beg - od začetka datoteke
- ios:: cur - od trenutne pozicije kazalca
- ios::end - od konca datoteke

Primer:

```
datoteka.seekg(0);
datoteka seekg(20,ios::cur);
```

Primer iz predavanj:

```
#include<fstream.h>
#include<istream.h>
#include<ctype.h>
#include <stdlib.h>
void ZamenjajMaleInVelike(char *bes,int dolzina);
void IzpisiBesedo(char *bes,int dolzina);
void main()
{
      char ime_dat[30];
      cout << "\tVpisi ime datoteke: ";
      cin.getline(ime_dat,30,'\n');

      ifstream dat(ime_dat);

      char znak, prejsnji_znak;
      char beseda[30];
```

```
      long st_znakov_v_besedi, st_znakov = 0, st_besed = 0, st_vrstic=0;
      long st_znakov_v_besedah=0;

while (!dat.eof())
      {
             znak = dat.get();
             if (znak=='\n') st_vrstic++;
              st_znakov_v_besedi=0;
             while (isalpha(znak) && isalnum(znak))
             {
                    beseda[st_znakov_v_besedi] = znak;
                    st_znakov_v_besedi++;
                    st_znakov++;
                    prejsnji_znak=znak;
                    if (!dat.eof())
                      znak = dat.get();
              }

        if (isspace(znak) || (!isalnum(znak)))
             {
                    if ( isalnum(prejsnji_znak))
                    {
                        st_besed++;
                        if (st_besed % 2 == 1)  //ce je beseda liha

ZamenjajMaleInVelike(beseda,st_znakov_v_besedi);
                        IzpisiBesedo(beseda,st_znakov_v_besedi);
                        st_znakov_v_besedah+=st_znakov_v_besedi;
                    }
                    cout << znak;
                    prejsnji_znak=znak;
                    st_znakov++;
             }

      } // konec While


      cout << endl;
      cout << endl << "\tStevilo besed: " << st_besed;
      cout << endl << "\tStevilo vseh znakov: " << st_znakov;
      cout << endl << "\tStevilo vrstic: " << st_vrstic;
      cout << endl << "\tPovprecno stevilo besed na vrstico je: "
              << (float)(st_besed) / st_vrstic;
      cout << endl << "\tPovprecno stevilo crk v besedi je: "
              << (float)(st_znakov_v_besedah) / st_besed;
      cout << endl;
}
```

## Datoteke z naključnim dostopom

Datoteke z naključnim dostopom so takšne, kjer lahko do posameznih zapisov datoteke dostopamo neposredno. Pri datotekah z naključnim dostopom uporabljamo datotečne kazalce. Za določanje datotečnega kazalca uporabimo metodi *seekp* in *seekg*. Trenutni položaj kazalca pa dobimo z metodama *tellg()* in *tellp()*.

```
#include <iostream.h>
#include <fstream.h>
#include <process.h>

int main()
{
ifstream infile;
ofstream outfile;
ofstream printer;
char filename[20];

   cout << "Vnesi ime datoteke ki jo želiš kopirat ----> ";

   cin >> filename;

   infile.open(filename, ios::nocreate);
   if (!infile)
   {
      cout << "Vhodne datoteke ni možno odpret.\n";
      exit(1);
   }

   outfile.open("copy");
   if (!outfile)
   {
      cout << "Izhodne datoteke ni možno odpret.\n";
      exit(1);
   }

   printer.open("PRN");
   if (!printer)
   {
      cout << "Nekaj je narobe s tiskalnikom.\n";
      exit(1);
    }

   cout << "Vse tri datoteke so odprte.\n";

char one_char;

   printer << "Tukaj je začetek izpisovanja na tiskalnik.\n\n";

   while (infile.get(one_char))
   {
      outfile.put(one_char);
      printer.put(one_char);
   }

   printer << "\n\nTu pa se tiskanje konča.\n";

   infile.close();
   outfile.close();
   printer.close();

   return 0;
}
```

Primer:
Z datoteke, ki je prikazana na koncu primera, preberemo dva vektorja realnih števil. Prebrana števila najprej izpišemo na zaslon z natančnostjo dveh decimalk in kot desno poravnana. Nato ista števila izpisemo v E opisu (z natančnostjo treh decimalk). Na koncu izpišemo prebrana števila na poljubno datoteko na enak način kot na zaslon.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;
main()
{
ifstream vh_dat;
ofstream iz_dat;
string ime_dat, nasl_vrstic;
int st_vrstic;
vector <float> x, y;
cout << "Podaj ime vhodne datoteke: ";
cin >> ime_dat;
//odpri vhodno datoteko
vh_dat.open(ime_dat.c_str());
//ce datoteke ni mozno odpreti javi napako
if(!vh_dat)
{
         cerr << "Ne morem odpreti datoteke " << ime_dat << endl;
         return -1;
}
//preberi naslovno vrstico
vh_dat >> nasl_vrstic;
vh_dat >> nasl_vrstic;
//preberi koliko je vrstic, v katerih so izpisana realna stevila
vh_dat >> st_vrstic;
x.resize(st_vrstic);
y.resize(st_vrstic);
//beri realna stevila
for(int i = 0; i < st_vrstic; i++)
        vh_dat >> x[i] >> y[i];
//zapri vhodno datoteko
vh_dat.close();
//prebrana stevila izpisi na zaslon v decimalnem opisu
//z natancnostjo 2 decimalk, stevila naj bodo desno poravnana
cout.precision(2);
for(i = 0; i < st_vrstic; i++)
        cout << setw(6) << fixed << x[i] << " " << setw(6) << y[i] <<
endl;
//prebrana stevila izpisi na zaslon v E opisu
//z natancnostjo 3 decimalk, stevila naj bodo desno poravnana
cout.precision(3);
for(i = 0; i < st_vrstic; i++)
        cout << uppercase << scientific << setw(12) << x[i] << " " <<
setw(12) << y[i] << endl;
//Preberi ime izhodne datoteke z zaslona
cout << "Podaj ime izhodne datoteke: ";
cin >> ime_dat;
//odpri izhodno datoteko
iz_dat.open(ime_dat.c_str());
```

```
if(!iz_dat)
{
        cerr << "Ne morem odpreti datoteke " << ime_dat << endl;
        return -1;
}
//stevila prebrana iz vhodne datoteke izpisi na izhodno
//datoteko na enak nacin kot na zaslon
iz_dat.precision(2);
for(i = 0; i < st_vrstic; i++)
        iz_dat << setw(6) << fixed << x[i] << " " << setw(6) << y[i] <<
endl;
iz_dat.precision(3);
for(i = 0; i < st_vrstic; i++)
        iz_dat << uppercase << scientific << setw(12) << x[i] << " " <<
setw(12) << y[i] << endl;
//zapri izhodno datoteko
iz_dat.close();
return 0;
}
```

Vhodna datoteka je lahko definirana v naslednji obliki:

```
Vhodna datoteka
4
1.6 2.0
-3.0 6.8
5.45 10.0
7.0 14.0
```

Potem ima izhodna datoteka naslednjo obliko:

```
  1.60  2.00
 -3.00  6.80
  5.45 10.00
  7.00 14.00
  1.600E+000  2.000E+000
 -3.000E+000  6.800E+000
  5.450E+000  1.000E+001
  7.000E+000  1.400E+001
```

Primer iz predavanj – zaporedno branje:

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>

struct Podatki_stranke {
    int stev_rac;
    char priimek[15];
    char ime[10];
    float stanje;
};

void izpisi_vrsto(ostream &output, Podatki_stranke s)
{
```

```
output << setiosflags(ios::left) << setw(6) << s.stev_rac
                     << setw(16) << s.priimek
                    << setw(11) << s.ime
                    << setw(12) << setprecision(2)
                    << setiosflags(ios::fixed | ios::right)
                    << s.stanje << endl;}
main(){
     ifstream vho_racuni("racuni.dat", ios::in);
     if (!vho_racuni) {
            cerr << "Zbirke ni moc odpreti." << endl;
            exit(1);
     }
     cout << setiosflags(ios::left) << setw(6) << "Racun“
              << setw(16) << "Priimek" << setw(11)
              << "Ime" << setiosflags(ios::right)
              << setw(12) << "Stanje" << endl;
     Podatki_stranke stranka;
     vho_racuni.read((char *)&stranka, sizeof(Podatki_stranke));
     while (!vho_racuni.eof()) {
            if (stranka.stev_rac != 0)
                   izpisi_vrsto(cout, stranka);
            vho_racuni.read((char *)&stranka, sizeof(Podatki_stranke));
     }      return 0;}
```

Primer iz predavanj – branje in izpis zaporedne datoteke:

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdlib.h>

void izpisi_vrsto(int rac, char *priimek, float sta){
     cout << setiosflags(ios::left) << setw(10) << rac
              << setw(13) << priimek << setw(9)
              << setprecision(2)
              << setiosflags(ios::fixed | ios::right)
              << sta << endl;}
main(){
     ifstream vho_zbirka_strank("stranke.dat", ios::in);
     if (!vho_zbirka_strank) {
            cerr << "Zbirke ni moc odpreti" << endl;
            exit(1);       }
     int racun;
     char priimek[10];
     float stanje;
     cout << setiosflags(ios::left) << setw(10) << "Racun“
              << setw(16) << "Priimek" << "Stanje" << endl;
     while (vho_zbirka_strank >> racun >> priimek >> stanje)
            izpisi_vrsto(racun, priimek, stanje);
     return 0;}
```

Primer iz predavanj – pisanje na datoteko z naključnim dostopom:

```
#include <iostream.h>
 #include <fstream.h>
#include <stdlib.h>
struct Podatki_stranke {
int stev_rac;
char priimek[15];
```

```
char ime[10];
float stanje; };
main() {
 ofstream izh_racuni("racuni.dat", ios::ate);
 if (!izh_racuni) {
 cerr << "Zbirke ni moc odpreti." << endl; exit(1); }
 cout << "Vnesi stevilko racuna " <<
 "(med 1 in 100, 0 za konec vnosa)" << endl << "? "; Podatki_stranke
stranka;
cin >> stranka.stev_rac;
 while (stranka.stev_rac > 0 && stranka.stev_rac <= 100) {
 cout << "Vnesi priimek, ime in stanje" << endl << "? ";
 cin >> stranka.priimek >> stranka.ime >> stranka.stanje;
izh_racuni.seekp((stranka.stev_rac - 1) * sizeof(Podatki_stranke));
 izh_racuni.write((char *)&stranka, sizeof(Podatki_stranke));
 cout << "Vnesi stevilko racuna" << endl << "? ";
cin >> stranka.stev_rac;
 }
return 0; }
```

# REKURZIJA

**Rekurzivne funkcije** so takšne funkcije, ki kličejo same sebe. Uporabljamo jih takrat, ko lažje izrazimo rešitev problema tako, da se pri rešitvi sklicujemo na rešitev samo.

Primer – Faktoriela:

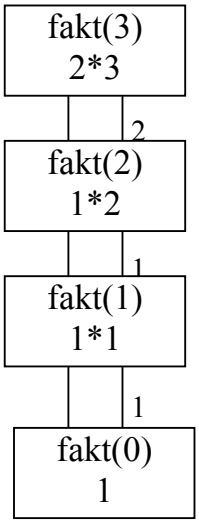
Rekurzivna definicija:

$$F(0) = 1$$
$$F(n) = F(n-1) * n$$

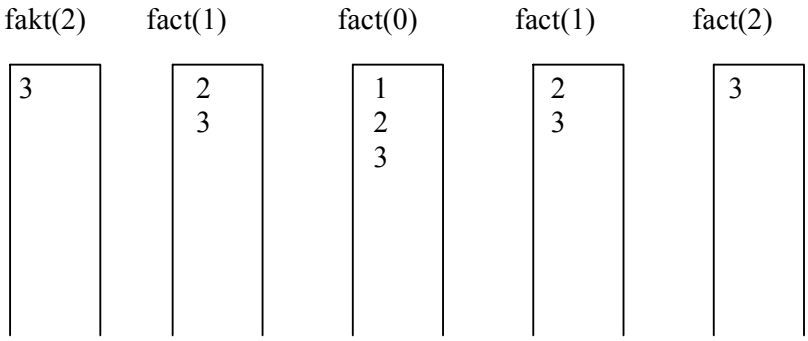
Implementacija funkcije:

```
int faktoriela(int n)
{
  if (n>0)
      return faktoriela(n-1)*n;
  else
      if (n==0)
        return 1;
      else
      {
        cout << "Taksne vrednosti ne morem "
  cout << "izracunati!";
  return -1;
}
}
```

**Klici in vračanje vrednosti:**




**Slika sklada:**



Primer - Fibonaccijeva števila:

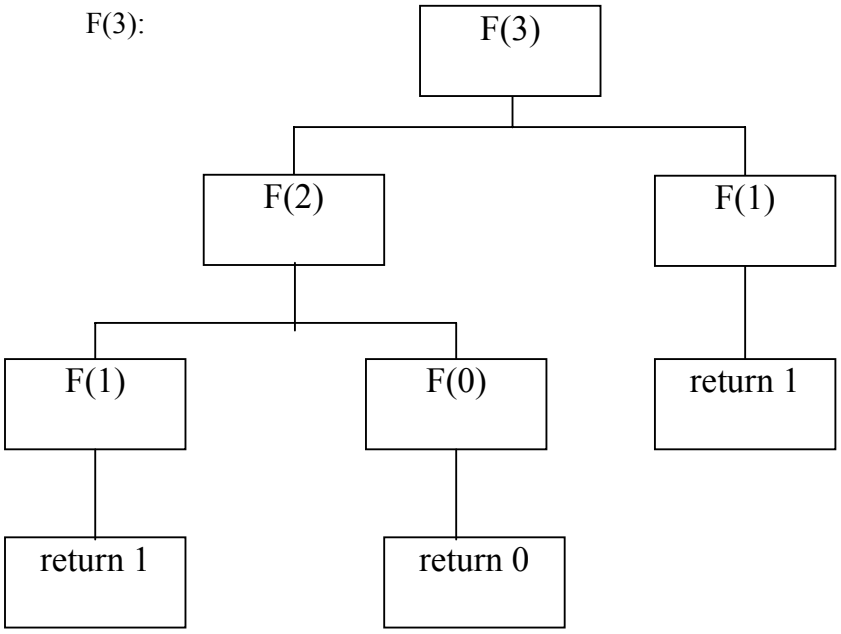
Rekurzivna definicija:

$$Fib(0) = 0$$
$$Fib(1) = 1$$
$$Fib(n) = Fib(n-1) * Fib(n-2)$$

Implementacija funkcije:

```
int fib(int n)
{
  if ((n==0) || (n==1))
    return n;
  if (n>0)
return fib(n-1)*fib(n-2);
}
```

F(3):




**Primer -** Quicksort:

Tudi algoritem sortiranja elementov polja lahko rešimo z rekurzijo.

Reševali bomo na ta način (poglejte si algoritem Quick na materialih):

Najprej določimo indeks elementa , ki je na sredini polja je na sredini polja (x = polje[l+r], kjer l pomeni indeks levega dela (na začetku je 0), r pa pomeni indeks desnega dela (na začetku je n)). Vsi elementi, ki imajo indeks manjši od indeksa sredinskega elementa, morajo biti manjši od sredinskega elementa. Vsi elementi, ki imajo indeks večji od indeksa sredinskega elementa pa morajo biti večji od sredinskega elementa. Ko temu kriteriju zadostimo, kličemo funkcijo rekurzivno, tako da na isti način sortiramo še elemente na levi strani od sredinskega elementa (QuickSort(Polje,l, j)) in elemente na desni strani od sredinskega elementa (QuickSort(Polje,i,r)).

```
void QuickSort(int Polje[], int m, int n)
{
        int i, j;
     if (m < n) {
             i = m; j = n;
             Delitev(Polje,i,j);
             QuickSort(Polje,m,j);
             QuickSort(Polje,i,n);
     }
}

main()
{
   const int n=16;
    int polje[n]={44,55,12,42,94,6,18,67,3,13,99,15,23,77,59,17};
    Izpisi(polje, n);
   QuickSort(polje,0,n);
   Izpisi(polje,n);
   return 0;
}
void Delitev (int Polje[], int& i, int& j)
{
     int Srednji, Temp;
     Srednji = Polje[ (i + j) / 2 ] ;
     do {
             while (Polje[i] < Srednji) i++;
             while (Polje[j] > Srednji) j--;
             if (i <= j) {
                Temp = Polje[i];
                Polje[i] = Polje[j];
                Polje[j] = Temp;
                 i++; j--;
             }
     } while (i <= j);
}
```

Primer izpisa:
Začetno polje:

```
44, 55, 12, 42, 94,  6, 18, 67,  3, 13, 99, 15, 23, 77, 59, 17,
```

Program izvaja naslednje zamenjave:

```
 3, 55, 12, 42, 94,  6, 18, 67, 44, 13, 99, 15, 23, 77, 59, 17,
 3, 17, 12, 42, 94,  6, 18, 67, 44, 13, 99, 15, 23, 77, 59, 55,
 3, 17, 12, 42, 23,  6, 18, 67, 44, 13, 99, 15, 94, 77, 59, 55,
 3, 17, 12, 42, 23,  6, 18, 15, 44, 13, 99, 67, 94, 77, 59, 55,
```

```
3, 17, 12, 42, 23,  6, 18, 15, 13, 44, 99, 67, 94, 77, 59, 55,

3, 17, 12, 13, 23,  6, 18, 15, 42, 44, 99, 67, 94, 77, 59, 55,
3, 17, 12, 13, 15,  6, 18, 23, 42, 44, 99, 67, 94, 77, 59, 55,
3,  6, 12, 13, 15, 17, 18, 23, 42, 44, 99, 67, 94, 77, 59, 55,

3,  6, 12, 13, 15, 17, 18, 23, 42, 44, 55, 67, 94, 77, 59, 99,
3,  6, 12, 13, 15, 17, 18, 23, 42, 44, 55, 67, 59, 77, 94, 99,
3,  6, 12, 13, 15, 17, 18, 23, 42, 44, 55, 59, 67, 77, 94, 99,
```

Končna razvrstitev je naslednja:

```
3,  6, 12, 13, 15, 17, 18, 23, 42, 44, 55, 59, 67, 77, 94, 99,
```

# Objektno programiranje

## Kapsuliranje

**Kapsuliranje (encapsulation) –** je mehanizem za implementacijo abstraktnega podatkovnega tipa. Omogoča skupno obravnavo tako podatkov kot tudi operacij nad njimi.

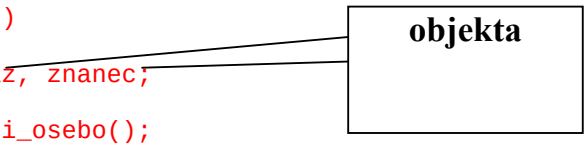
Primer programa za evidenco oseb z uporabo kapsuliranja:

```
#include <iomanip.h>
#include <iostream.h>


struct naslov
{
  // podatki za naslov
  char  ulica[20];
  char  mesto[20];
  int   postna_stevilka;

  // metode za naslov
  void vnesi_naslov ();
  void izpisi_naslov();

};

void naslov::vnesi_naslov ()

{
     cout << "Vpisi ulico:";
     cin  >> ulica;
          cout << "Vpisi mesto:";
     cin >>  mesto;
     cout << "Vpisi poštno stevilko:";
          cin >>  postna_stevilka;
}



void naslov::izpisi_naslov ()

{

     cout    << ulica << " "
             << postna_stevilka
             << " " << mesto << endl;

}


struct datum
{
  // podatki za datum
```

```
  int  dan;
  int  mesec;
  int  leto;

  // metode za datum
  void vnesi_datum();
  void izpisi_datum();
};


void datum::vnesi_datum ()

{
     cout << "Vpisi dan:";
     cin  >> dan;
          cout << "Vpisi mesec:";
     cin >>  mesec;
     cout << "Vpisi leto:";
          cin >> leto;
}

void datum::izpisi_datum ()

{

     cout    << dan << "."
             << mesec
             << "." << leto << endl;
  }


const int max_otrok = 10;

struct otrok
{
  // podatki za otroka
  char  ime[20];
  int   starost;

  // metoda za otroka
  void vnesi_otroka();
  void izpisi_otroka();
};

void otrok::vnesi_otroka()
{

       cout << "Vpisi ime:";
       cin >> ime;
       cout << "Vpisi starost:";
       cin >> starost;

}

void otrok::izpisi_otroka()
{

       cout << ime << "   " << starost << endl;

}
```

```
 struct otroci
{
  // podatki za otroke
  int  st_otrok;
  otrok  njegov_otrok[max_otrok];

  // metode za otroke
  void vnesi_otroke();
  void izpisi_otroke();
};


void otroci::vnesi_otroke()
{
     cout << "Vnesi stevilo otrok:";
     cin >> st_otrok;
     for (int i=0; i<st_otrok; i++)
      njegov_otrok[i].vnesi_otroka();

}


 void otroci::izpisi_otroke()
{
     cout << "Ime      Starost" << endl;
     for (int i=0; i<st_otrok; i++)
      njegov_otrok[i].izpisi_otroka();

}


 struct oseba
{
  // podatki za osebo
  char ime[30];
  char priimek[30];
  char spol;
  naslov  naslov_doma;
  naslov  naslov_v_sluzbi;
  datum   datum_rojstva;
  otroci njegovi_otroci;

  // metode za osebo
  void vnesi_osebo();
  void izpisi_osebo();

};


void oseba::vnesi_osebo ()

{
     cout << "Vpisi ime in priimek:";
     cin  >> ime;
     cin  >> priimek;
          cout << "Vpisi spol:";
     cin >> spol;
     cout << "Vpisi naslov doma:" << endl;
```

```
      naslov_doma.vnesi_naslov ();
      cout << "Vpisi naslov v sluzbi" << endl;
      naslov_v_sluzbi.vnesi_naslov ();
      cout << "Vpisi datum rojstva" << endl;
      datum_rojstva.vnesi_datum();
      cout << "Vnesi otroke" << endl;
      njegovi_otroci.vnesi_otroke();
}

void oseba::izpisi_osebo()

{
      cout  <<"Ime in priimek osebe:" << ime
               << "  " << priimek << endl;
          cout << "Spol: " << ((spol=='m')?"moški" : "ženski") << endl;
      cout << "Naslov doma: " << endl;
      naslov_doma.izpisi_naslov ();
      cout << "Naslov v sluzbi" << endl;
      naslov_v_sluzbi.izpisi_naslov ();
      cout << "Datum rojstva" << endl;
      datum_rojstva.izpisi_datum();
      if (njegovi_otroci.st_otrok>0)
      {
             cout<<"Otroci:" << endl;
                    njegovi_otroci.izpisi_otroke();
      }
      else
             cout << "Oseba nima otrok" << endl;
}

// primer glavne funkcije

void main()
{
  oseba jaz, znanec;

  jaz.vnesi_osebo();
  znanec.vnesi_osebo();
  jaz.izpisi_osebo();
  znanec.izpisi_osebo();

}
```




Spremenljivka strukture oziroma abstraktnega podatkovnega tipa se imenuje **objekt.** V danem trenutku lahko obstaja nič, eden ali več objektov nekega razreda. Vsi objekti iste strukture ali abstraktnega podatkovnega tipa imajo iste metode in enake podatke, seveda pa ima lahko vsak objekt v podatkih svoje vrednosti.

## SKRIVANJE ELEMENTOV

```
 struct oseba
{
 private:
  // podatki za osebo
  char ime[30];
```

```
  char priimek[30];
  char spol;
  naslov  naslov_doma;
  naslov  naslov_v_sluzbi;
  datum   datum_rojstva;
  otroci njegovi_otroci;

public:
  // metode za osebo
  void vnesi_osebo();
  void izpisi_osebo();

};
```

V prejšnjem primeru vpisa osebe bi lahko vsaka metoda, seveda tudi tista, ki ni definirana znotraj strukture oseba, dostopala do podatkov ali metod, ki so definirane znotraj strukture oseba. Če želimo doseči, da določene elemente (metode ali podatke) skrijemo, moramo uvesti mehanizem skrivanja. To uvedemo, tako da definiramo nekatere metode in podatke javne, ostalo pa skrijemo. Z besedo **private** definiramo tiste metode in podatke, ki bodo skriti. Z besedo **public** pa definiramo tiste podatke in metode, ki bodo javni.

## Razredi

Razred objektov **definiramo z rezervirano besedo** class.

**Primer razreda oseba:**

```
class oseba
{
// podatki za osebo
  char ime[30];
  char priimek[30];
  char spol;
  naslov  naslov_doma;
  naslov  naslov_v_sluzbi;
  datum   datum_rojstva;
  otroci njegovi_otroci;

// metode za osebo
  void vnesi_osebo();
  void izpisi_osebo();

};
```

Razlika med razredom in strukture je v tem, da so pri razredu, če eksplicitno ne definiramo vsi podatki in metode skriti (private) pri strukturi pa so vsi podatki in metode javni (public).

V našem primeru bi lahko ekvivalentno napisali:

```
struct oseba
```

```
{

  // metode za osebo
  void vnesi_osebo();
  void izpisi_osebo();


 private:
  // podatki za osebo
  char ime[30];
  char priimek[30];
  char spol;
  naslov  naslov_doma;
  naslov  naslov_v_sluzbi;
  datum   datum_rojstva;
  otroci njegovi_otroci;

};
```

**ali**

```
class oseba
{
// podatki za osebo
  char ime[30];
  char priimek[30];
  char spol;
  naslov  naslov_doma;
  naslov  naslov_v_sluzbi;
  datum   datum_rojstva;
  otroci njegovi_otroci;

public:
  // metode za osebo
  void vnesi_osebo();
  void izpisi_osebo();

};
```

Primeri programov:

```
// Tvorjenje strukture, dolocitev njenih
// elementov in izpis.
#include <iostream.h>

struct Cas {     // opredelitev strukture
     int ura;      // 0-23
     int minuta;   // 0-59
     int sekunda;  // 0-59
};

void izpisi_24h(const Cas &);  // prototip
void izpisi_12h(const Cas &);  // prototip

int  main()
{
```

```
      Cas cas_vecerje;    // spremenljivka novega
    // tipa Cas
      // postavitev elementov na pravilne vrednosti
      cas_vecerje.ura = 18;
      cas_vecerje.minuta = 30;
      cas_vecerje.sekunda = 0;

      cout << "Vecerja bo ob ";
      izpisi_24h(cas_vecerje);
      cout << " v 24-urni obliki," << endl
 << "kar je ";
      izpisi_12h(cas_vecerje);
      cout << " v 12-urni obliki." << endl;
      // postavitev elementov na napacne vrednosti
      cas_vecerje.ura = 29;
      cas_vecerje.minuta = 73;
      cas_vecerje.sekunda = 103;
      cout << endl
      << "Cas z napacnimi vrednostmi: ";
      izpisi_24h(cas_vecerje);
      cout << endl;
      return 0;
}


// Izpis casa v 24-urni obliki
void izpisi_24h(const Cas &c)
{
      cout << (c.ura < 10 ? "0" : "") << c.ura
 << ":" << (c.minuta < 10 ? "0" : "")
 << c.minuta << ":"
 << (c.sekunda < 10 ? "0" : "")
 << c.sekunda;
}



// Izpis casa v 12_urni obliki
void izpisi_12h(const Cas &c)
{
      cout << ((c.ura == 0 || c.ura == 12) ?
    12 : c.ura % 12)
 << ":" << (c.minuta < 10 ? "0" : "")
 << c.minuta
              << ":" << (c.sekunda < 10 ? "0" : "")
 << c.sekunda
 << (c.ura < 12 ? " AM" : " PM");
}
```

Rezultat:

```
Vecerja bo ob 18:30:00 v 24-urni obliki,
kar je 6:30:00 PM v 12-urni obliki.

Cas z napacnimi vrednostmi: 29:73:103
```

```
// Razred Cas.
#include <iostream.h>

// Opredelitev abstraktnega podatkovnega tipa
// Cas.
class Cas {
public:
     Cas();     // konstruktor
     void nastavi_cas(int, int, int);  // nastavi // ure, minute in
sekunde
     void izpisi_24h(); // izpis casa v 24-urni
// obliki
     void izpisi_12h(); // izpis casa v 12-urni
// obliki
private:
     int ura;       // 0 - 23
     int minuta;    // 0 - 59
     int sekunda;   // 0 - 59
};


// Konstruktor Cas nastavi vse podatkovne
// elemente na nic.
// S tem se zagotovi zacetna pravilnost vseh
// objektov Cas.
Cas::Cas() { ura = minuta = sekunda = 0; }


// Postavi novo vrednost Casa v 24-urni obliki.
// Izvedi preverjanje veljavnosti podatkovnih
// vrednosti. Postavi napacne vrednosti na nic.
void Cas::nastavi_cas(int u, int m, int s)
{
     ura = (u >= 0 && u < 24) ? u : 0;
     minuta = (m >= 0 && m < 60) ? m : 0;
     sekunda = (s >= 0 && s < 60) ? s : 0;
}


// Izpisi Cas v 24_urni obliki
void Cas::izpisi_24h()
{
     cout << (ura < 10 ? "0" : "") << ura << ":"
               << (minuta < 10 ? "0" : "")
  << minuta << ":"
               << (sekunda < 10 ? "0" : "")
  << sekunda;
}


// Izpisi Cas v 12-urni obliki
void Cas::izpisi_12h()
{
     cout << ((ura == 0 || ura == 12) ? 12 :
 ura % 12)
               << ":" << (minuta < 10 ? "0" : "")
  << minuta
               << ":" << (sekunda < 10 ? "0" : "")
  << sekunda
               << (ura < 12 ? " AM" : " PM");
```

```
}


// Gonilnik za preizkus preprostega razreda Cas.
int  main()
{
     Cas c;  // uvedba elementa c razreda Cas

     cout << "Zaceten cas v 24-urni obliki je ";
     c.izpisi_24h();
     cout << endl
 << "Zaceten cas v 12-urni obliki je ";
     c.izpisi_12h();

     c.nastavi_cas(13, 27, 6);
     cout << endl << endl
    << "Cas v 24-urni obliki po nastavi_cas je ";
     c.izpisi_24h();
     cout << endl
    << "Cas v 12-urni obliki po nastavi_cas je ";
     c.izpisi_12h();


c.nastavi_cas(99, 99, 99);  // poskus
// postavitve napacnih vrednosti
     cout << endl << endl
<< "Po poskusu postavitve napacnih vrednosti:"
         << endl << "Cas v 24-urni obliki: ";
     c.izpisi_24h();
     cout << endl << "Cas v 12-urni obliki: ";
     c.izpisi_12h();
     cout << endl;
     return 0;
}
```

Rezultat:

```
Zaceten cas v 24-urni obliki je 00:00:00
Zaceten cas v 12-urni obliki je 12:00:00 AM

Cas v 24-urni obliki po nastavi_cas je 13:27:06
Cas v 12-urni obliki po nastavi_cas je 1:27:06 PM

Po poskusu postavitve napacnih vrednosti:
Cas v 24-urni obliki: 00:00:00
Cas v 12-urni obliki: 12:00:00 AM
```

Naslednji primer:

```
// Uporaba operatorjev . in -> za dostop do
// elementov razreda
//
// POZOR: V PRIHODNJIH PRIMERIH SE IZOGIBAMO
// JAVNIH (public) PODATKOV!
#include <iostream.h>

// Preprost razred Stevec
class Stevec {
```

```
public:
     int x;
     void izpisi() { cout << x << endl; }
};


main()
{
     Stevec stev,         // tvorimo objekt stev
    *kaz_stev = &stev,   // kazalec na stev
&sklic_stev = stev;  // sklicevanje na stev

cout << "Priredi 7 elementu x in izpisi z  uporabo imena objekta: ";
     stev.x = 7;       // priredi 7 elementu x
     stev.izpisi();    // klici metodo izpisi

cout << "Priredi 8 elementu x in izpisi z uporabo sklicevanja: ";
     sklic_stev.x = 8;    // priredi 8 elementu x
     sklic_stev.izpisi();  // klici metodo izpisi

cout << "Priredi 10 elementu x in izpisi z uporabo kazalca: ";
     kaz_stev->x = 10;   // priredi 10 elementu x
     kaz_stev->izpisi(); // klici metodo izpisi
     return 0;
}
```

Rezultat:

```
Priredi 7 elementu x in izpisi z uporabo imena objekta: 7
Priredi 8 elementu x in izpisi z uporabo sklicevanja: 8
Priredi 10 elementu x in izpisi z uporabo kazalca: 10
```

Izpis za primer št. 4:

```
Zacetni cas v 24-urni obliki je 00:00:00
Zacetni cas v 12-urni obliki je 12:00:00 AM

Cas v 24-urni obliki po nastavi_cas je 13:27:06
Cas v 12-urni obliki po nastavi_cas je 1:27:06 PM

Po poskusu nastavitve napacnih vrednosti:
24-urna oblika: 00:00:00
12-urna oblika: 12:00:00 AM
```

Predstavitev uporabne funkcije:

```
#include <iostream.h>
#include "prodajal.h"

main()
{
     Prodajalec p;        // tvorba objekta p tipa
    // Prodajalec
     double znesek_prodaje;

     for (int i = 1; i <= 12; i++) {
```

```
                cout << "Vnesi znesek prodaje za mesec "
                          << i << ": ";
                cin >> znesek_prodaje;
                p.doloci_prodajo(i, znesek_prodaje);
        }

        p.izpisi_letno_prodajo();

        return 0;
  }
```

Metode za razred Prodajalec:

```
  #include <iostream.h>
  #include <iomanip.h>
  #include "prodajal.h"

  // Konstruktor vstavi zacetne vrednosti v polje
  Prodajalec::Prodajalec()
  {
        for (int i = 0; i < 12; i++)
                zneski[i] = 0.0;
  }


  // Funkcija doloci znesek za enega izmed 12
  // mesecev
  void Prodajalec::doloci_prodajo(int mesec,
    double znesek)
  {
        if (mesec >= 1 && mesec <= 12 && znesek > 0)
                zneski[mesec - 1] = znesek;
        else
                cout << "Napacen mesec ali znesek "
   << "prodaje" << endl;
  }



  // Zasebna uporabna funkcija za izracun skupne
  // letne prodaje
  double Prodajalec::skupna_letna_prodaja()
  {
        double vsota = 0.0;

        for (int i = 0; i < 12; i++)
                vsota += zneski[i];

        return vsota;
  }


// Izpis skupne letne prodaje
  void Prodajalec::izpisi_letno_prodajo()
  {
        cout << setprecision(2)
   << setiosflags(ios::fixed |
     ios::showpoint)
   << endl << "Skupna letna prodaja je : "
                << skupna_letna_prodaja() << " SIT"
```

```
 << endl;
}


// Definicija razreda Prodajalec
// Metode so opredeljene v prodajal.cpp

#ifndef PRODAJAL_H
#define PRODAJAL_H

class Prodajalec {
public:
Prodajalec();        // konstruktor
     // Uporabnik vnese znesek prodaje za en
// mesec:
     void doloci_prodajo(int, double);
     void izpisi_letno_prodajo();

private:
     double zneski[12];    // zneski prodaje za 12
 // mesecev
     double skupna_letna_prodaja();   // uporabna
   //funkcija
};

#endif
```

Rezultat:

```
Vnesi znesek prodaje za mesec 1: 25000
Vnesi znesek prodaje za mesec 2: 12000
Vnesi znesek prodaje za mesec 3: 5000
Vnesi znesek prodaje za mesec 4: 10000
Vnesi znesek prodaje za mesec 5: 3000
Vnesi znesek prodaje za mesec 6: 8000
Vnesi znesek prodaje za mesec 7: 4000
Vnesi znesek prodaje za mesec 8: 9000
Vnesi znesek prodaje za mesec 9: 30000
Vnesi znesek prodaje za mesec 10: 23000
Vnesi znesek prodaje za mesec 11: 16000
Vnesi znesek prodaje za mesec 12: 19000

Skupna letna prodaja je : 164000.00 SIT
```

Predstavitev konstruktorja s privzetimi vrednostmi za razred Cas:

```
#include <iostream.h>
#include "cas2.h"

main()
{
     Cas c1, c2(2), c3(21, 34), c4(12, 25, 42),
c5(27, 74, 99);

     cout << "Uvedeno z:" << endl
               << "vsemi argumenti privzetimi:"
 << endl << "   ";
```

```
     c1.izpisi_24h();
     cout << endl << "   ";
     c1.izpisi_12h();

     cout << endl << "uro doloceno; minuto in"
 << sekundo privzetima:"
 << endl << "   ";
     c2.izpisi_24h();
     cout << endl << "   ";
     c2.izpisi_12h();

     cout << endl << "uro in minuto dolocenima;"
 << sekundo privzeto:"
 << endl << "   ";
     c3.izpisi_24h();
     cout << endl << "   ";
     c3.izpisi_12h();

     cout << endl << "uro, minuto in sekundo"
 << dolocenimi:" << endl << "   ";
     c4.izpisi_24h();
     cout << endl << "   ";
     c4.izpisi_12h();

     cout << endl << "vsemi napacnimi vrednostmi:"
 << endl << "   ";
     c5.izpisi_24h();
     cout << endl << "   ";
     c5.izpisi_12h();
     cout << endl;

     return 0;
}


// Definicije metod za razred Cas.
#include <iostream.h>
#include "cas2.h"

// Konstruktor za zacetno nastavitev zasebnih
// podatkov. Privzete vrednosti so 0 (glej
// definicijo razreda).

Cas::Cas(int ura, int min, int sek)
     { nastavi_cas(ura, min, sek); }


// Nastavi vrednosti za uro, minuto in sekundo.
// Napacne vrednosti se postavijo na 0.
void Cas::nastavi_cas(int u, int m, int s)
{
     ura = (u >= 0 && u < 24) ? u : 0;
     minuta = (m >= 0 && m < 60) ? m : 0;
     sekunda = (s >= 0 && s < 60) ? s : 0;
}


// Izpis casa v 24-urni obliki : UU:MM:SS
void Cas::izpisi_24h()
{
```

```
     cout << (ura < 10 ? "0" : "") << ura << ":"
             << (minuta < 10 ? "0" : "")
 << minuta << ":"
             << (sekunda < 10 ? "0" : "")
 << sekunda;
}

 // Izpis casa v 12-urni obliki: UU:MM:SS AM
// (ali PM)
void Cas::izpisi_12h()
{
     cout << ((ura == 0 || ura == 12) ? 12 :
 ura % 12)
             << ":" << (minuta < 10 ? "0" : "")
 << minuta
             << ":" << (sekunda < 10 ? "0" : "")
 << sekunda
             << (ura < 12 ? " AM" : " PM");
}

// Deklaracija razreda Cas.
// Metode razreda so opredeljene v cas2.cpp

// zascita pred veckratnim vkljucevanjem vkljucitvene zbirke
#ifndef CAS2_H
#define CAS2_H

class Cas {
public:
     Cas(int = 0, int = 0, int = 0); //konstruktor
  // s privzetimi vrednostmi
     void nastavi_cas(int, int, int);
     void izpisi_24h();
     void izpisi_12h();

private:
     int ura;
     int minuta;
     int sekunda;
};

#endif
```

Rezultati izvedbe:

```
Uvedeno z:
vsemi argumenti privzetimi:
   00:00:00
   12:00:00 AM
uro doloceno; minuto in sekundo privzetima:
   02:00:00
   2:00:00 AM
uro in minuto dolocenima; sekundo privzeto:
   21:34:00
   9:34:00 PM
uro, minuto in sekundo dolocenimi:
   12:25:42
   12:25:42 PM
vsemi napacnimi vrednostmi:
   00:00:00
   12:00:00 AM
```

Prikaz vrstnega reda, po katerem se klicejo konstruktorji in destruktorji:

```
#include <iostream.h>
#include "tvori.h"

void tvori(void);   // prototip

Tvori_in_unici prvi(1);  // globalen objekt

main()
{
     cout << "   (globalen tvorjen pred main)"
 << endl;

     Tvori_in_unici drugi(2);    // lokalen objekt
     cout << "   (lokalen automatic v main)"
 << endl;

     static Tvori_in_unici tretji(3);  // lokalen
// objekt
     cout << "   (lokalen static v main)" << endl;

     tvori();  // klic funkcije za tvorbo objektov

     Tvori_in_unici cetrti(4);  // lokalen objekt
     cout << "   (lokalen automatic v main)"
 << endl;
     return 0;
}
// Funkcija za tvorbo objektov
void tvori(void)
{
     Tvori_in_unici peti(5);
     cout << "   (lokalen automatic v tvori)"
 << endl;

     static Tvori_in_unici sesti(6);
     cout << "   (lokalen static v tvori)"
 << endl;

     Tvori_in_unici sedmi(7);
     cout << "   (lokalen automatic v tvori)"
 << endl;
}
// Definicije metod za razred Tvori_in_unici
#include <iostream.h>
#include "tvori.h"

Tvori_in_unici::Tvori_in_unici(int vrednost)
{
     podatek = vrednost;
     cout << "Objekt " << podatek << "   konstruktor";
}


Tvori_in_unici::~Tvori_in_unici()
     { cout << "Objekt " << podatek
   << "   destruktor " << endl; }
```

```
// Definicija razreda Tvori_in_unici
// Metode so opredeljene v tvori.cpp.

#ifndef TVORI_H
#define TVORI_H

class Tvori_in_unici {
public:
     Tvori_in_unici(int);  // konstruktor
     ~Tvori_in_unici();    // destruktor
private:
     int podatek;
};

#endif
```

Program izpiše:

```
Objekt 1   konstruktor   (globalen tvorjen pred main)
Objekt 2   konstruktor   (lokalen automatic v main)
Objekt 3   konstruktor   (lokalen static v main)
Objekt 5   konstruktor   (lokalen automatic v tvori)
Objekt 6   konstruktor   (lokalen static v tvori)
Objekt 7   konstruktor   (lokalen automatic v tvori)
Objekt 7   destruktor
Objekt 5   destruktor
Objekt 4   konstruktor   (lokalen automatic v main)
Objekt 4   destruktor
Objekt 2   destruktor
Objekt 6   destruktor
Objekt 3   destruktor
Objekt 1   destruktor
```

# KONSTRUKTORJI IN DESTRUKTORJI

**Konstruktor** namesti objekt v pomnilnik (rezervira pomnilnik za objekt in inicializira njegove podatke). Inicializacija podatkov se prenese v konstruktor is se avtomatsko izvrši ob tvorjenju objektov.

**Destruktor** objekt zbriše (sprosti rezerviran pomnilniški prostor – podatki osnovnih tipov se sprostijo avtomatsko, sami pa moramo sprostiti dinamično rezerviran pomnilniški prostor, ki ga rezerviramo z operatorjem new).

**Ločimo naslednje vrste konstruktorjev:**

- **Privzeti konstruktor**
- **Kopirni konstruktor**
- **Pretvorbeni konstruktor in**
- **Ostali konstruktorji**

Primer:

```
class Kompleksno {
public:
     Kompleksno();                                                    //
privzeti konstruktor
     Kompleksno(double, double);                  // konstruktor
     Kompleksno (const Kompleksno& k);         // kopirni konstruktor
     Kompleksno (double r);                              // pretvorbeni
konstruktor
     ~Kompleksno() { }                            // destruktor


     void izpisi() const;                           // izhod

private:
     double realni;       // realni del
     double imaginarni;   // imaginarni del
};
```

**Privzeti konstruktor** nastavi podatke na privzete vrednosti

```
Kompleksno::Kompleksno()
{ realni = imaginarni = 0.0;
}
```

**Kopirni konstruktor** ustvari nov objekt z istimi vrednostmi, kot jih ima argument

```
Kompleksno::Kompleksno(const Kompleksno& k)
{ realni = k.realni;
  imaginarni = k.imaginarni;
```

```
}
```

**Pretvorbeni konstruktor** pretvori realno število v kompleksno število

```
Kompleksno::Kompleksno(double r)
{ realni = r;
  imaginarni = 0.0;
}
```

**Konstruktor, ki ustvari kompleksno število iz dveh realnih števil**

```
Kompleksno::Kompleksno(double r, double i)
{
     realni = r;
     imaginarni = i;
}
```

Za ilustracijo delovanja uporabe konstruktorjev, smo definirali še metodo za izpis kompleksnega števila

```
void Kompleksno::izpisi() const
     { cout << '(' << realni << ", "
   << imaginarni << ')';
 }
```

Glavna funkcija za ilustracijo delovanja uporabe konstruktorjev ima naslednjo obliko:

```
int main()
{
     Kompleksno x, y(4.3, 8.2), z(3.3);

     cout << "x: ";
     x.izpisi();
     cout << endl << "y: ";
     y.izpisi();
     cout << endl << "z: ";
     z.izpisi();
          Kompleksno w(y);
     cout << endl << "w: ";
     w.izpisi();
     cout << endl;

     return 0;
}
```

Program izpiše naslednje

```
x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 0)
w: (4.3, 8.2)
```

Uporabimo lahko tudi konstruktorje s privzetimi argumenti:

```
Kompleksno::Kompleksno(double r=0.0, double i=0.0)
{
     realni = r;
     imaginarni = i;
}
```

**Inicializacijski seznami**

Večina onstruktorjev opravlja samo inicializacijo podatkov razreda. C++ pozna poenostavljeno sintakso za inicializacijo:

```
Kompleksno::Kompleksno(double r=0.0, double i=0.0)
: realni(r), imaginarni(i)
{
}
```

ali npr. pri kopirnem konstruktorju

```
Kompleksno::Kompleksno(const Kompleksno& k)
: realni (k.realni), imaginarni (k.imaginarni)
{
}
```

**Ta seznam imenujemo** inicializacijski seznam:

Primer uporabe destruktorja:

```
int main()
{
     Kompleksno* x;
          x = new Kompleksno(1.0,1.0);    // klic konstruktorja za
ustvarjanje
//dinamičnega objekta
     cout << "x: ";
     x->izpisi();
     delete x;                                        // klic destruktorja
     return 0;
}
```

Za dostop do metod ali podatkov v objektih, ki so dinamično generirani (generirani so z uporabo kazalcev) uporabljamo operator ->.

V našem primeru smo uporabili ta operator v **x ->izpisi().** Enakovreden zapis je tudi: **(*x).izpisi()**;

## Konstantni objekti

To so objekti, ki jim ne moremo spremeniti vrednosti podatkov.

Primer:

```
const  Kompleksno i(0.0,1.0);
```

Konstantne metode

V konstantnem objektu lahko kličemo samo konstantne metode, to so tiste metode, ki ne spremenijo podatkov v objektu. Primer takšne metode je metoda izpisi:

```
void Kompleksno::izpisi() const
     { cout << '(' << realni << ", "
   << imaginarni << ')';
 }
```

Konstantni podatki

Poleg konstantnih metod lahko definiramo v razredu tudi konstantne podatke. Primer:

```
class Kompleksno {
public:
     Kompleksno();                                                    //
privzeti konstruktor
     Kompleksno(double, double);                  // konstruktor

private:
     double realni;       // realni del
     double imaginarni;   // imaginarni del
          const double pi;
};

Konstruktor kličemo na naslednji način:
Kompleksno::Kompleksno(double r=0.0, double i=0.0)
: realni(r), imaginarni(i), pi(3.14)
{
}
```

## Kazalec this

Pri vsakem klicu metode se vanjo kot argument prenese še naslov objekta nad katerim ta metoda deluje. Naslov objekta, ki je klical metodo lahko uporablja tudi programer.

Primer:

```
class Preizkus {
public:
     Preizkus(int = 0);    // privzet konstruktor
     void izpisi() const;
```

```
private:
     int x;
};


Preizkus::Preizkus(int a) { x = a; } // konstruktor

void Preizkus::izpisi() const
{
     cout << "        x = " << x << endl
               << "  this->x = " << this->x << endl
               << "(*this).x = " << (*this).x << endl;
}

main()
{
     Preizkus a(12);
    a.izpisi();
    return 0;
}
```

Program izpiše:

```
x = 12
  this->x = 12
(*this).x = 12
```

## Prekrivanje operatorjev (operator overloading)

Določene operatorje želimo prekriti. Npr. operacije + nimamo definirane za kompleksna števila.

`Primer:`

```
class Kompleksno {
public:
     Kompleksno(double = 0.0, double = 0.0);
     // prekrivanje operatorja za setevanje
    Kompleksno operator+(const Kompleksno &) const;
     // prekrivanje operatorja za odstevanje
    Kompleksno operator-(const Kompleksno &) const;
    // prekrivanje operatorja za prirejanje
    Kompleksno &operator=(const Kompleksno &);
    void izpisi() const;
private:
     double realni;       // realni del
     double imaginarni;   // imaginarni del
};

// konstruktor
Kompleksno::Kompleksno(double r, double i)
{
```

```
        realni = r;
        imaginarni = i;
  }
  // Prekrivni operator sestevanja
  Kompleksno Kompleksno::operator+(const Kompleksno &operand2) const
  {
        Kompleksno vsota;
        vsota.realni = realni + operand2.realni;
        vsota.imaginarni =     imaginarni +
  operand2.imaginarni;
        return vsota;
  }

  // Prekrivni operator odstevanja
  Kompleksno Kompleksno::operator-(const Kompleksno &operand2) const
  {
        Kompleksno razlika;
        razlika.realni = realni - operand2.realni;
        razlika.imaginarni = imaginarni –
  operand2.imaginarni;
        return razlika;
  }

  // Prekrivni operator =
  Kompleksno& Kompleksno::operator=(const Kompleksno &desno)
  {
        realni = desno.realni;
        imaginarni = desno.imaginarni;
        return *this;   // omogoca veckratno prirejanje
  }

  // Prikazi objekt Kompleksno v obliki: (a, b)
  void Kompleksno::izpisi() const
        { cout << '(' << realni << ", "
     << imaginarni << ')';
   }
```

Primer glavne funkcije:

```
  main()
  {
        Kompleksno x, y(4.3, 8.2), z(3.3, 1.1);
        cout << "x: ";
        x.izpisi();
        cout << endl << "y: ";
        y.izpisi();
        cout << endl << "z: ";
        z.izpisi();
        x = y + z;
        cout << endl << endl << "x = y + z:" << endl;
        x.izpisi();
        cout << " = ";
        y.izpisi();
        cout << " + ";
        z.izpisi();
        x = y - z;
        cout << endl << endl << "x = y - z:" << endl;
        x.izpisi();
        cout << " = ";
        y.izpisi();
```

```
        cout << " - ";
        z.izpisi();
        cout << endl;
        return 0;
}
```

Program izpiše:

```
x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 1.1)

x = y + z:
(7.6, 9.3) = (4.3, 8.2) + (3.3, 1.1)

x = y - z:
(1, 7.1) = (4.3, 8.2) - (3.3, 1.1)
```

## Vmesnik razreda

Vmesnik razreda predstavljajo vsi javni elementi razreda. Uporabnik nekega razreda mora poznati le njegov vmesnik, podrobnosti o implementaciji pa zanj niso pomembne. Tako je programska koda neodvisna od implementacije.

## Prijateljske funkcije

Do privatnih metod ali podatkov razreda lahko dostopajo samo prijateteljske funkcije razreda (friend funkcije), čeprav niso člani razreda.

Primer:

```
class Stevec {
        friend void doloci_x(Stevec &, int);
 // opredelitev friend funkcije
public:
        Stevec() { x = 0; }       // konstruktor
        void izpisi() const { cout << x << endl; }
 // izhod
private:
        int x;  // podatkovni element razreda
};

void doloci_x(Stevec &c, int vred)
{
        c.x = vred;  // dovoljeno: doloci_x je friend
// funkcija Stevca
}
```

```
Glavna funkcija ima naslednjo obliko:

main()
{
     Stevec objekt;

     cout << "objekt.x po uvedbi: ";
     objekt.izpisi();
     cout << "objekt.x po klicu friend funkcije
             doloci_x: ";
     doloci_x(objekt, 8);  // dolocitev x s friend
// funkcijo
     objekt.izpisi();

     return 0;
}
```

Program izpiše:

```
objekt.x po uvedbi: 0
objekt.x po klicu friend funkcije doloci_x: 8
```

PRIMER OBJEKTNEGA PROGRAMIRANJA

Z uporabo razredov bomo naredili program, ki vodi evidenco o bančnih računih uporabnikov. Za vsak račun imamo podane dvige in vloge, številko računa ter ime in priimek uporabnika. O vsakem dvigu in vlogi imamo podan datum in znesek. Program lahko vnaša in izpisuje osnovne podatke o varčevalcih in transakcije (dvige in vloge). Program naj javlja, če transakcija ni mogoča. Transakcija ni mogoča, če vpišemo številko računa, ki ne obstaja ali če hočemo izvesti dvig denarja pa ga nimamo dovolj na računu. Program naj tudi sortira varčevalce glede na vsoto privarčevanega denarja.

Program smo si zamislili tako, da bomo najprej definirali razrede za **posamezno transakcijo, transakcije, posamezen bančni račun** in **bančne račune varčevalcev.**

**Najprej bomo definirali konstante**

```
const max_niz = 16;         // največja dolžina niza
const max_trans = 10;                // največje število transakcij
const max_rac = 10;         // največje število bančnih računov
```

**in naštevalni tip:**

```
enum tip_transakcije {vloga, dvig};      // tip transakcije
```

**Nato bomo definirali razred in metode za posamezno transakcijo**

```
class Transakcija
{
     public:
            Transakcija() {};
            Transakcija(tip_transakcije tip, char dat[], float zne);
            void Izpisi_Transakcijo();
            float VrniZnesek() { return znesek; };
     private:
            tip_transakcije tip_tr;
            char datum[max_niz];
            float znesek;
};
```

**Konstruktor za transakcijo:**

```
Transakcija :: Transakcija(tip_transakcije tip, char dat[], float zne)
{
     tip_tr = tip;

     for (int i=0; i<max_niz; i++)
            datum[i] = dat[i];

        znesek = zne;
}
```

**Funkcija za izpis transakcije:**

```
void Transakcija :: Izpisi_Transakcijo()
{
     if (tip_tr==dvig)
            cout << setw(35) << "Dvig";
     else
            cout << setw(35) << "Vloga";
     cout << setw(22) << datum << setw(15) << znesek << endl;
}
```

**Nato definiramo razred in metode za bančni račun:**

```
class Bancni_Racun  {
     public:
            Bancni_Racun() {};
            Bancni_Racun(char im[], char pr[], int st_r);
            void Izpisi_Varcevalca();
            void Vnesi_Transakcijo();
            void Izpisi_Transakcije();
```

```
            void Izpisi_Varcevalca_Komplet();
            int  VrniStevRac() { return stevilka_racuna; };
            float VrniVsoto()  { return vsota_na_racunu; };
      private:
            char ime[max_niz];
            char priimek[max_niz];
            int  stevilka_racuna;
            int  stevilo_transakcij;
            Transakcija trans[max_trans];
            float vsota_na_racunu;
};
```

**Konstruktor za bančni račun:**

```
Bancni_Racun ::     Bancni_Racun(char im[], char pr[], int st_r)
{
     int i;
     for (i=0; i<max_niz; i++)
           ime[i] = im[i];

     for (i=0; i<max_niz; i++)
           priimek[i] = pr[i];

     stevilka_racuna = st_r;
     stevilo_transakcij = 0;
     vsota_na_racunu = 0.0;
}
```

**Funkcija za izpis osnovnih podatkov o varčevalcu**

```
void Bancni_Racun :: Izpisi_Varcevalca()
{
    cout << setw(15) << ime;
    cout << setw(15) << priimek;
    cout << setw(20) << stevilka_racuna;
    cout << setw(20) << vsota_na_racunu << endl;
}
```

**Funkcija za vnašanje  transakcij nad bančnim računom:**

```
void Bancni_Racun :: Vnesi_Transakcijo()
{
     if (stevilo_transakcij == max_trans)
          //preverjamo ali je ze dosezeno//maks. stev. transakcij
     {       cout << " Najvecje stevilo transakcij je ze dosezeno...";
             return;
     }
     int tip;
     float znesek;
     char datum[max_niz];
     tip_transakcije tip_t;
```

```
        cout << "Vnesi transakcijo 1 - dvig, 2 - vloga: ";
        cin  >> tip;
        if (tip==1)     // če je tip transakcije dvig
                tip_t = dvig;
        else                // če je tip transakcije vloga
                tip_t = vloga;
        cout << "Vnesi znesek transakcije: ";
        cin  >> znesek;

        if ((tip==1) && (vsota_na_racunu - znesek < 0))
            //preverjamo ali je prišlo do prekoračitve (minusa)
             {   cout << endl
                        << " Varcevalec ne more dvigniti toliko
denarja.";
                return;
        }

        cout << "Vnesi datum transakcije: ";
        cin  >> datum;

        Transakcija tran(tip_t,datum,znesek); //klic prired. konstruktorja
        trans[stevilo_transakcij] = tran;

        if (tip==1)      // vsoto na racuno povecamo ali zmanjsamo
                vsota_na_racunu -= trans[stevilo_transakcij].VrniZnesek();
        else
                vsota_na_racunu += trans[stevilo_transakcij].VrniZnesek();

        stevilo_transakcij++;//stevilo transakcij se poveca za ena
}
```

**Funkcija za izpis vseh transakcij povezanih s tem bančnim računom:**

```
void Bancni_Racun :: Izpisi_Transakcije()
{
        cout << endl;
        cout << setw(35) << "Tip transakcije"
                 << setw(22) << "Datum"
                 << setw(15) << "Znesek" << endl;
        for (int i=0; i<stevilo_transakcij; i++)
        {
                trans[i].Izpisi_Transakcijo();
        }
}
```

**Za bančne račune definiramo naslednji razred in naslednje metode:**

```
class Bancni_ Racuni{
public:
  Bancni_Racuni() { stev_varcevalcev = 0; };
  void Vpisi_Varcevalce();
  void Izpisi_Varcevalce();
```

```
  void Izpisi_Transakcije();
  void Vnesi_Transakcije();
  void Sortiraj_Varcevalce();  //sortira varčevalce po vsoti na računu
  void Zamenjaj(int prvi, int drugi);
  int  Isci_Varcevalca(int st_r);  // išče varčevalca po številki računa
                                                  // vrne -1, če ga ne
najde oz. njegovo mesto
 private:
  Bancni_Racun racun[max_rac];
  int stev_varcevalcev;
};
```

**Funkcija za vpis varčevalcev:**

```
void Bancni_Racuni :: Vpisi_Varcevalce()
{
     int st,st_rac_var;
     char ime_var[max_niz],priimek_var[max_niz];
     cout << endl << "Koliko varcevalcev zelis vnesti: ";
     cin  >> st;          //preberemo število vpisanih varčevalcev
     if (st + stev_varcevalcev > max_rac) // preverjamo, da ne pride do
                                                                    //
prekoračitve
       cout  << "Na razpolago imas samo "
                    <<max_rac-stev_varcevalcev<<" prostih mest";
     else
     {
       for (int i=0; i<st; i++)
        {
          cout << endl;
          cout << " Vnesi ime osebe, ki zeli odpreti racun: ";
          cin  >> ime_var;
               cout << " Vnesi priimek osebe, ki zeli odpreti racun: ";
          cin  >> priimek_var;
               cout << " Vnesi stevilko racuna: ";
          cin  >> st_rac_var;
               if (Isci_Varcevalca(st_rac_var) == -1) //preverjamo ali že

//obstaja račun s to številko
         { racun[stev_varcevalcev] =
                Bancni_Racun(ime_var,priimek_var,st_rac_var);
           stev_varcevalcev++;                    }
         else
          {
             cout << endl << " Bancni racun s to stevilko je ze
odprt...";
          }
     }
     }
}
```

**Funkcija za  izpis vseh varčevalcev (na splošen način):**

```
void Bancni_Racuni :: Izpisi_Varcevalce()
```

```
{
  if (stev_varcevalcev==0) //preverjamo, če je odprt sploh kakšen račun
     {
            cout << " V banki ni odprt nobeden bancni racun...";
            return;
     }
   cout  << endl << setw(15) << "Ime"
           << setw(15) << "Priimek"
           << setw(20) << "Stevilka racuna"
           << setw(20) << "Vsota na racunu" << endl;

  for (int i=0; i<stev_varcevalcev; i++) //gremo skozi vse odprte račune
  {
    racun[i].Izpisi_Varcevalca(); //izpis posameznega varčevalca
   }
}
```

**Funkcija za izpis varčevalca in vseh njegovih transakcij:**

```
void Bancni_Racuni :: Izpisi_Transakcije()
{
    if (stev_varcevalcev==0) //preverjamo ali je sploh kakšen odprt račun
     {
            cout << " V banki ni odprt nobeden bancni racun...";
            return;
     }
     int st;
     cout << endl << " Vpisi stevilko racuna: ";
     cin  >> st;  //preberemo stevilko zeljenega racuna, ki ga bomo
                          //podrobno izpisali
         int pozicija = Isci_Varcevalca(st); //iskanje pozicije tega

//varcevalca
         if (pozicija != -1) //če obstaja tak bančni račun ga izpišemo
     {
       cout  << endl << setw(15) << "Ime"
                     << setw(15) << "Priimek"
                     << setw(20) << "Stevilka racuna"
                      << setw(20) << "Vsota na racunu" << endl;


     racun[pozicija].Izpisi_Varcevalca();//izpis varčevalca
     racun[pozicija].Izpisi_Transakcije();//izpis vseh njegovih
transakcij
     }
     else //če ne obstaja izpišemo sporočilo
     {
            cout << endl << " Bancni racun s to stevilko ni odprt...";
     }
}
```

**Funkcija, ki išče varčevalca in nam vrne njegovo pozicijo oziroma -1, če varčevalca ne najdemo:**

```
int Bancni_Racuni :: Isci_Varcevalca(int st_r) //VHOD: številka računa
{
	for (int i=0; i<stev_varcevalcev; i++) //pregled vseh varčevalcev
	{
		if (st_r == racun[i].VrniStevRac()) //če ga najdemo
		{
			return i; //vrnemo njegovo pozicijo
		}
	}
	return -1; //če ga ne najdemo, vrnemo -1
}
```

**Funkcija za vnos transakcij na bančni račun:**

```
 void Bancni_Racuni :: Vnesi_Transakcije()
{
	int st;
	cout << endl << "Vnesi stevilko bancnega racuna: ";
	cin  >> st;
	int pozicija = Isci_Varcevalca(st); //iscemo njegovo pozicijo
	if (pozicija != -1) //če ga najdemo
		racun[pozicija].Vnesi_Transakcijo();
	else
		cout << endl << " Ta racun ni odprt...";
}
```

**Funkcija zamenja dva bančna racuna -> uporaba pri sortiranju:**

```
void Bancni_Racuni :: Zamenjaj(int prvi, int drugi)
{
	Bancni_Racun vmesni;
	vmesni = racun[prvi];
	racun[prvi] = racun[drugi];
	racun[drugi] = vmesni;
}
```

**Funkcija sortira bančne račune po vsoti na računu:**

```
void Bancni_Racuni :: Sortiraj_Varcevalce()
{
	for (int i=0; i<stev_varcevalcev-1; i++)
	{
		for (int j=i+1; j<=stev_varcevalcev; j++)
		{
			if (racun[i].VrniVsoto()<racun[j].VrniVsoto())
				Zamenjaj(i,j);
		}
	}
}
```

**Funkcija izpise glavni menu:**

```
void izpisi_glavni_menu()
{
         cout << endl;
         cout << "Vpisi:" << endl;
         cout << "1 - za vnos osnovnih podatkov o varcevalcih" << endl;
         cout << "2 - za izpis osnovnih podatkov o varcevalcih" << endl;
         cout << "3 - sortiranje varcevalcev po visini denarja na racunu"
                 <<  endl;
         cout << "4 - za vnos transakcij" << endl;
         cout << "5 - za izpis vseh podatkov o posameznih varcevalcih" <<
endl;

}  // izpisi_glavni_menu
```

**Glavna funkcija ima naslednjo obliko:**

```
int main()
{
    int izbira;            //  izbira klica podprograma iz menija
    char odgovor;         //  ali želimo se izvajati program (d,n)
    Bancni_Racuni racuni; //  racuni varcevalcev

    do
    {
         // izpis menija in izbor podprograma, ki ga zelimo uporabiti
         izpisi_glavni_menu();
        do
         {                    cin >> izbira;
                   if ((izbira <= 0)|| (izbira >5))
                              {
                                  cout << "Nisi vnesel prave stevilke"
<< endl;
                                   izpisi_glavni_menu();
                              }
         } while ((izbira <= 0)|| (izbira >5));

        // klici podprogramov glavnega programa na podlagi izbire iz
menija
         switch (izbira)
         {
      case 1:
                    racuni.Vpisi_Varcevalce();
                    break;
           case 2:
                    racuni.Izpisi_Varcevalce();
                    break;
           case 3:
                    racuni.Sortiraj_Varcevalce();
                    break;
           case 4:
                    racuni.Vnesi_Transakcije();
```

```
                        break;
                case 5:
                        racuni.Izpisi_Transakcije();
                        break;
            }



    do
      {
       cout << endl<<"Ali zelis se izvajati program - vpisi d ali n:";
            cin >> odgovor;
       }
    while ((odgovor !='d') && (odgovor !='n') &&
                    (odgovor !='D') && (odgovor !='N'));

}
while ((odgovor =='d') || (odgovor =='D'));
return 0;
}
```

# HIERARHIJE RAZREDOV

Z uporabo **dedovanja** iz obstoječih razredov tvorimo nove razrede. Takšno tvorjenje razredov imenujemo **izpeljava** in razrede, ki tako nastanejo **izpeljani razredi** ali **podrazredi**. Izpeljani razred ima torej svoj **nadrazred**, tj. razred, iz katerega je izpeljan. Dedovanje je lahko **enkratno** ali **večkratno** – v tem primeru je razred izpeljan iz dveh ali več nadrazredov.

Izpeljani razred **podeduje** vse elemente svojega nadrazreda. K tem elementom pa lahko doda še svoje elemente (podatke in metode) in ponovno definira metode iz svojega nadrazreda. Zato lahko izpeljane razrede razumemo kot posebne primere svojih nadrazredov. Tako izpeljani razredi predstavljajo **specializacijo**, nadrazredi pa **posplošitev**. Ker izpeljan razred podeduje vse elemente svojega nadrazreda, je objekt izpeljanega razreda hkrati tudi objekt nadrazreda. Obratno pa ne velja, kajti objektu nadrazreda manjkajo podatki in metode, ki so definirani v izpeljanem razredu.

**Primer izpeljave razreda student iz razreda oseba:**

Razred student predstavlja sprecializacijo razreda oseba. K podatkom o določeni osebi bomo torej dodali podatke, ki določajo, da je ta oseba pravzaprav študent. Zaradi enostavnosti smo izbrali podatek "smer", ki predstavlja smer študija posameznega študenta.

```
#include<iostream.h>

enum spol {moski, zenski};

class oseba
{
     public:
             oseba(char im[], char p[], spol s, char dat_r[]);
             oseba() {};
             void Vpisi();
             void Izpisi();
     private:
             char ime[15];
             char priimek[15];
             spol spol_o;
             char datum_r[10];
};

class student : public oseba
{
     public:
             void Vpisi();
             void Izpisi();
     private:
             char smer[20];
};
```

Beseda **public**, ki je zapisana pri izpeljavi nam pove, da so javni ("public") elementi nadrazreda (oseba) javni tudi v izpeljanem razredu (student).

```
oseba :: oseba(char im[], char p[], spol s, char dat_r[])
{

      for (int i=0; i<15; i++)
        ime[i] = im[i];

      for (i=0; i<15; i++)
        priimek[i] = p[i];

      spol_o = s;

      for (i=0; i<10; i++)
        datum_r[i] = dat_r[i];
}

void oseba :: Vpisi()
{
      int sp;
      cout << endl;
      cout << " Vpisi ime: ";
      cin  >> ime;
      cout << " Vpisi priimek: ";
      cin  >> priimek;
      cout << " Vpisi spol 1 - zenski, 2 - moski: ";
      cin  >> sp;

      if (sp == 1)
              spol_o = zenski;
      else
              spol_o = moski;

      cout << " Vpisi datum rojstva: ";
      cin  >> datum_r;
}


void student :: Vpisi()
{
      cout << " Vpisi smer: ";
      cin  >> smer;
}


void oseba :: Izpisi()
{
      cout << endl << " Ime: " << ime;
      cout << endl << " Priimek: " << priimek;
      if (spol_o == zenski)
              cout << endl << " Spol: Zenski";
      else
              cout << endl << " Spol: Moski";

      cout << endl << " Datum rojstva: " << datum_r << endl;
}
```

```
void student :: Izpisi()
{
     cout << " Smer: " << smer << endl;
}


int main()
{
     student luka;
     luka.oseba::Vpisi();
     luka.Vpisi();
     luka.oseba::Izpisi();
     luka.Izpisi();
     return 0;
}
```

## Zaščiteni elementi

Poleg javnih in privatnih elementov lahko v razredih definiramo tudi zaščitene elemente z določilom **protected**. **Zaščiteni elementi** so vmesna stopnja med javnimi in privatnimi elementi. Do **javnih elementov** nekega razreda lahko z objektom tega razreda dostopa kdorkoli v programu. Do **privatnih elementov** nekega razreda lahko dostopajo samo ta razred in njegovi prijatelji. Do **zaščitenih elementov** nekega razreda lahko dostopajo enako kot do privatnih elementov ta razred in njegovi prijatelji, poleg tega pa še vsi izpeljani razredi in prijatelji teh razredov.

## Tipi izpeljav

Izpeljavo, v kateri uporabimo ključno besedo public, kot npr. v primeru:

```
class student : public oseba
```

imenujemo **javna izpeljava**. Poleg te ima jezik C++ še **zaščiteno** in **privatno** izpeljavo. Ta dva tipa izpeljav določimo s ključnima besedama protected in private:

```
class student : protected oseba
```

in

```
class student : private oseba
```

Zaščitena in privatna izpeljava v izpeljanih razredih omejita dostop do zaščitenih in javnih elementov nadrazreda. Dostop do elementov nadrazreda
je predstavljen v naslednji tabeli:

| tip elementa | tip izpeljave | | |
|---|---|---|---|
| | public | protected | private |
| public | public | protected | private |
| protected | protected | protected | private |
| private | private | private | private |

Podatki v tabeli povedo, kakšen je dostop do elementa nadrazreda v izpeljanih razredih, če imamo podan tip elementa in tip izpeljave. Pri **zaščiteni izpeljavi** se javni elementi nadrazreda obravnavajo kot zaščiteni elementi, pri **privatni izpeljavi** pa se javni in zaščiteni elementi obravnavajo kot privatni elementi.

# POLIMORFIZEM

**Dedovanje** in **polimorfizem** sta dve najpomembnejši lastnosti objektnega programiranja. Prvo smo že spoznali, druga pa označuje princip, da lahko različni objekti razumejo isto sporočilo in se nanj odzovejo vsak na svoj način. Obe lastnosti sta osnova objektnega programiranja in omogočata hitrejši razvoj in lažje vzdrževanje programske kode.

**Primer:**
Imejmo poleg razreda `student`, iz razreda `oseba` izpeljana še razreda `profesor` in `asistent`. Oba naj imata po en dodaten element: profesor naj ima predmet, ki ga poučuje, asistent pa predmet, pri katerem ima vaje. Vsak od njiju ima svojo inačico metode izpiši, ki izpiše podatke tega objekta.

Oglejmo si primer, ko želimo vnašati in izpisovati več objektov tipa študent, asistent ali profesor. Za shranjevanje bomo uporabili polje kazalcev. Ker ne vemo kakšne elemente bomo gradili (ta podatek bo znan šele v izvajanju), bomo shranjevali kazalce na skupni nadrazred.

```
#include  "student.h"
#include  "asistent.h"
#include  "profesor.h"
#include  <iostream.h>

const int max_oseb = 4;

int main()
{
   oseba* ptr_polje_oseb[max_oseb];  //polje za shranjevanje oseb
   oseba* ptr_začasna;

   cout << "Vnesi " << max_oseb << " osebe !" << endl;

   char tip_osebe[];
   char ime[vel_ime];
   char naslov[vel_naslov];
   char smer[vel_smer];
   char vaje[vel_vaje];
   char predmet[vel_predmet];


   for (int i = 0;  i  < max_oseb;  i++)
   {
     cout <<  endl << endl << "Vnesi tip osebe (S, A, P):  ";
     cin >> tip_osebe;
     cout  << "Vnesi ime           : ";
     cin << ime;
     cout  << "Vnesi naslov        : ";
     cin >> naslov;

     if  (tip_osebe  ==  'S')
       {
            cout << "Smer studenta               : ";
            cin >> smer;
```

```
                ptr_zacasna  =  new student (ime, naslov, smer);
          }
         else  if  (tip_osebe  ==  'A')
          {
                cout << "Predmet asistenta    : ";
                cin >> vaje;
                ptr_zacasna  =  new asistent (ime, naslov, vaje);
          }
         else  if  (tip_osebe  ==  'P')
          {
                cout << "Predmet profesorja   : ";
                cin >> vaje;
                ptr_zacasna  =  new profesor (ime, naslov, predmet);
          }
          else
            {
                cout  << "Napacen tip osebe!" << endl;
                cout  << "Ustvarilo bom objekt razreda oseba!";
                ptr_zacasna  =  new oseba (ime, naslov);
            }
           ptr_polje_oseb[i]  = ptr_zacasna;
     }

    //izpis vnesenih oseb
    for (i = 0;  i < max_oseb;  i++)
    {
       ptr_polje_oseb[i] ->izpisi();
       cout << endl;
    }


    //brisanje polja kazalcev
    for (i = 0;  i < max_oseb;  i++)
       delete ptr_polje_oseb[i];

     return 0;
}
```

Razložimo program! Najprej v zanki preberemo tip osebe ter ime in naslov osebe. Glede na tip osebe preberemo še smer, vaje ali predmet in ustvarimo objekt ustreznega razreda s pomočjo operatorja `new`. Ta nam vrne kazalec na objekt (`student*` ali `asistent*` oz. `profesor*`). Ta kazalec shranimo v spremenljivko `ptr_zacasna`, ki pa je tipa `oseba*`. Kako lahko to storimo? To je mogoče zato, ker je vsak študent (oz. asistent ali profesor) tudi oseba, zato lahko napravimo pretvorbo kazalca: npr. `student*` v `oseba*`. Pretvorba se napravi implicitno pri operatorju `new`. Pretvorbo bi lahko zahtevali tudi eksplicitno z naslednjim zapisom:

```
ptr_zacasna = (oseba*)new student(ime,naslov,smer);
```

Kadar kot tip osebe vnesemo nepravilno vrednost, ustvarimo objekt tipa oseba. Ko smo vnesli vse osebe, jih še izpišemo in nato zbrišemo (sproščanje dinamično dodeljenega pomnilnika!).

Prikaz delovanja programa:

```
Vnesi 4 osebe!

Vnesi tip osebe (S, A, P):   S
Vnesi ime                    :  Janez
Vnesi naslov                 :  Mladinska
Smer studenta                :  Anglescina

Vnesi tip osebe (S, A, P):   A
Vnesi ime                    :  Metka
Vnesi naslov                 :  Slovenska
Predmet asistenta            :  Nemscina

Vnesi tip osebe (S, A, P):   P
Vnesi ime                    :  France
Vnesi naslov                 :  Ljubljanska
Predmet profesorja   :  Računalnistvo

Vnesi tip osebe (S, A, P):   k
Vnesi ime                    :  Marija
Vnesi naslov                 :  Betnavska
Napacen tip osebe!
Ustvaril bom objekt razreda oseba!


Ime osebe            :  Janez
Naslov                    :  Mladinska

Ime osebe            :  Metka
Naslov                    :  Slovenska

Ime osebe            :  France
Naslov                    :  Ljubljanska

Ime osebe            :  Marija
Naslov                    :  Betnavska
```

Če si ogledamo izpis programa opazimo, da so se za vsak vnešeni objekt izpisali samo podatki, ki so zapisani v razredu `oseba` in ne vsi podatki študenta, asistenta in profesorja. Zakaj?

Ko smo objekte shranjevali, smo kazalec pretvorili v tip `oseba*`. Ko kličemo metodo `izpisi`, zato kličemo metodo `oseba::izpisi`. Vidimo, da navadna redefinicija metode v tem primeru ne zadošča. Čeprav imamo samo kazalce na razred `oseba`, želimo, da vsak objekt ohrani sebi lastno obnašanje ob klicu metode `izpisi`. Uporabiti moramo **polimorfno redefinicijo**.

Polimorfno redefinicijo omogočimo z definicijo **virtualne** metode. Metodo `izpisi` moramo v razredu `oseba` definirati kot virtualno, kar storimo na naslednji način:

```
virtual void izpisi() const;
```

Ta definicija pove, da naj izpeljani objekt kliče svojo funkcijo `izpisi`, čeprav bi bil klic izvršen preko kazalca na nadrazred (`oseba*`). Ob takšni definiciji metode `izpisi` v razredu `oseba`, bo prejšnji primer izpisal naslednje:

```
Vnesi 4 osebe!

Vnesi tip osebe (S, A, P):   S
Vnesi ime                    :  Janez
Vnesi naslov                 :  Mladinska
Smer studenta                :  Anglescina

Vnesi tip osebe (S, A, P):   A
Vnesi ime                    :  Metka
Vnesi naslov                 :  Slovenska
Predmet asistenta            :  Nemscina

Vnesi tip osebe (S, A, P):   P
Vnesi ime                    :  France
Vnesi naslov                 :  Ljubljanska
Predmet profesorja   :  Računalnistvo

Vnesi tip osebe (S, A, P):   k
Vnesi ime                    :  Marija
Vnesi naslov                 :  Betnavska
Napacen tip osebe!
Ustvaril bom objekt razreda oseba!


Ime osebe            :  Janez
Naslov                    :  Mladinska
Smer                      :  Angleščina

Ime osebe            :  Metka
Naslov                    :  Slovenska
Vaje pri                  :  Nemščina

Ime osebe            :  France
Naslov                    :  Ljubljanska
Predmet                   :  Računalništvo

Ime osebe            :  Marija
Naslov                    :  Betnavska
```

Problem se pojavi tudi pri brisanju dinamično alociranih objektov. Pri klicu operatorja `delete` se bo klical destruktor za razred `oseba` namesto destruktor ustreznega objekta. Zato moramo tudi destruktor definirati kot virtualno metodo:

```
virtual ~oseba();
```

## Abstraktni razredi

Včasih imamo hierarhijo razredov, v kateri imamo nadrazred, za katerega vemo, da ne bomo uporabljali objektov tega razreda. Tak razred hočemo imeti v hierarhiji samo zaradi definicije polimorfnega obnašanja nekaterih funkcij. Take razrede imenujemo **abstraktni razredi**. Primeri takih abstraktnih razredov so lahko na primer `oblika`, `telo`, `lik`, ...

Razred napravimo abstrakten tako, da definiramo vsaj eno izmed njegovih metod kot **čisto virtualno** (*pure virtual function*). Funkcija je čisto virtualna, kadar za definicijo vsebuje še inicializacijo = 0.

```
virtual void izracunaj_ploscino() const  =  0;
```

Funkcijo `izracunaj_ploscino` lahko napravimo čisto virtualno za razred `lik`. Funkcija omogoča polimorfno definicijo funkcije za vsakega izmed izpeljanih razredov, hkrati pa definira razred `lik` kot abstraktni razred, ki ga ne moremo uporabiti za tvorjenje objektov. To tudi pomeni, da moramo funkcijo `izracunaj_ploscino` redefinirati v vsakem razredu, ki ga izpeljemo iz razreda `lik` – kar pa je tudi smiselno, saj v razredu `lik` nimamo podatkov za izračun ploščine.

Čisto virtualne funkcije moramo redefinirati v vseh izpeljanih razredih, navadne virtualne pa ne. Kadar izpeljani razred nima redefinirane navadne virtualne funkcije, se bo klicala funkcija nadrazreda.

## Večkratno dedovanje

Razred je lahko izpeljan tudi iz več kot enega razreda. V tem primeru govorimo o večkratnem dedovanju. Definirajmo razred `student_asistent`, ki naj predstavlja osebe, ki so hkrati študentje in asistentje.

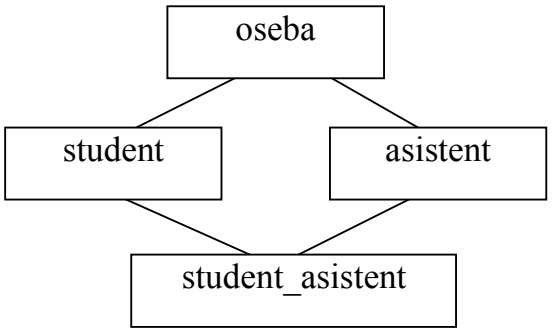
```
#include  "student.h"
#include  "asistent.h"

class  student_asistent  :  public student,  public asistent
{
   public:
     student_asistent (char i[],  char n[],  char s[],  char v[]);
     void  izpisi()  const;
}
```

V konstruktorju razreda `student_asistent` sedaj podamo ime, naslov, poleg tega pa še smer in predmet. Oglejmo si definicijo konstruktorjev in metode `izpisi()`.

```
#include  <iostream.h>

student_asistent :: student_asistent (char i,  char n,  char s,  char v)
     : student (i, n, s),  asistent ("---", "+++" , v)  {}
```

```
void student_asistent :: izpisi()  const
{
     student::izpisi();
     cout << endl;
     asistent::izpisi();
}
```

Z večkratnim dedovanjem dobi razred več primerkov nadrazredov:




V našem primeru je razred `student_asistent` izpeljan iz razredov `student` in `asistent`, ta dva pa sta izpeljana iz razreda `oseba`. Objekt tipa `student_asistent` v tem primeru vsebuje dva objekta tipa `oseba`, ki sta med seboj neodvisna in lahko vsebujeta različne podatke. Različne podatke v obeh objektih tipa oseba lahko dosežemo tako, da konstruktorjema nadrazredov `student` in `asistent` podamo različne argumente:

```
: student (i, n, s),  asistent ("---", "+++" , v)
```

Primer uporabe:

```
#include  <iostream.h>
#include  "student_asistent.h"
#include  "profesor.h"

int main()
{
     student  sosolec("Janez", "Ljubljanska 12", "Računalništvo");
     cout  << "Student sosolec" << endl;
     sosolec.izpisi();
     cout << endl << endl;

     student_asistent  sosed("Peter", "Mariborska 23",
      "Programska oprema",  "Programiranje ");
     cout  << "Student-asistent sosed"  << endl;
     sosed.izpisi();
```

```
        cout  << endl;

        return 0;
}
```

Program izpiše:

```
Student sosolec
Ime osebe    : Janez
Naslov              : Ljubljanska 12
Smer                : Računalništvo

Student-asistent sosed
Ime osebe    : Peter
Naslov              : Mariborska 23
Smer                : Programska oprema
Ime osebe    : ---
Naslov              : +++
Vaje                : Programiranje
```

Če v razredu `student_asistent` ne definiramo metode za izpis `izpisi()`, je klic `sosed.izpisi()` napačen, saj prevajalnik ne ve, katero od metod `izpisi()` naj kliče; tisto, ki je v razredu `student` ali tisto, ki je v razredu `asistent`. V tem primeru moramo sami navesti nadrazred, katerega metoda za izpis se naj kliče: `student::izpisi()` ali `asistent::izpisi()`.

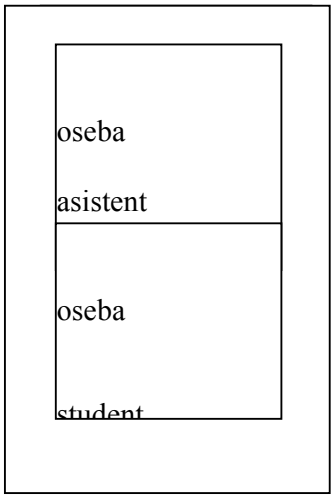
## Virtualni nadrazredi

Ugotovili smo, da se v razredih pri večkratnem dedovanju podatki podvajajo. Včasih je to potrebno, v večini primerov pa se želimo nepotrebnemu podvajanju podatkov izogniti. V ta namen lahko definiramo **virtualne nadrazrede** (*virtual base classes*).

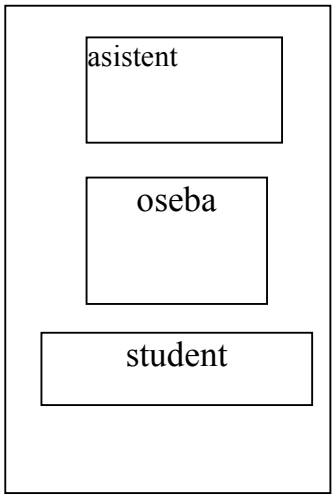
Kot virtualni nadrazred definiramo tisti razred, za katerega predvidevamo, da se bo uporabljal kot nadrazred za večkratno dedovanje. V našem primeru definiramo kot virtualni nadrazred, razred oseba. Zato zapišemo izpeljavo tazredov student, asistent in profesor na naslednji način:

```
        class student :: public virtual oseba
```

Takšno izpeljavo imenujemo tudi virtualna izpeljava. Poglejmo grafično predstavitev objekta razreda student_asistent pri navadni in virtualni izpeljavi:




(a) navadna izpeljava




(b) virtualna izpeljava

Spremeniti pa je potrebno tudi konstruktor razreda student_asistent:

```
student_asistent :: student_asistent (char i,  char n,  char s,  char v)
     : oseba(i, n),  student (i, n, s),  asistent ("---", "+++" , v)  {}
```