Pokazivači

Dinamička alokacija memorije

Sadržaj

- Ponavljanje – primjeri
- Dinamička alokacija memorije
- Dinamička alokacija polja
- Pokazivači na strukture
- Problemi s dinamičkom alokacijom memorije
  - Nedostupan objekt
  - Viseći pokazivač
- Pokazivači i funkcije
  - Pokazivači kao argumenti funkcije
  - Pokazivči kao povratna vrijednost funkcije
- Primjeri, pitanja, Literatura

Ponavljanje

- Koje od navedenih deklaracija pokazivača su dozvoljene:

```
int *pok1;
int* pok2;
int * pok3;
int*pok4;
```

Ponavljanje

- Na koje varijable se odnosi zvjezdica u sljedećoj naredbi (jesu li obje pokazivači na int ili ne?):

```
int* x, y;
```

Ponavljanje

- Primjer. Neka je zadan dio koda, odredite što će se ispisati (koje od vrijednosti su izjednačenje):

```
int a = 10, *pok1;
pok1 = &a;
int *pok2 = pok1;
cout<<"a = "<<a<<"\t&a = "<<&a<<endl;
cout<<"pok1 = "<<pok1<<"\t*pok1 = "
    <<*pok1<<"\t&pok1="<<&pok1<<endl;
cout<<"pok2 = "<<pok2<<"\t*pok2 = "
    <<*pok2<<"\t&pok2="<<&pok2<<endl;
```

Ponavljanje

- Primjer. Neka je zadan dio kod, odredite što će se ispisati:

```
int a = 10, b=2;
int *pok = &a;
*pok = a+b;
b = a+b;
pok = &b;
cout<<"a = "<<a<<"\tb = "
    <<b<<"\t*pok="<<*pok<<endl;
```

Ponavljanje

- Koja je razlika između zadane dvije naredbe:

```
...
*pok = 0;
...
```

i

```
...
pok = 0;
...
```

Ponavljanje

- Što će se ispisati na ekranu:

```
int a = 30;
int *pok = &a;
*pok = 0;
if(pok)
 cout<<*pok<<endl;
else
 cout<<"null-pokazivac.\n";
```

Ponavljanje

- Što će se ispisati na ekranu:

```
int a = 30;
int *pok = &a;
pok = 0;
if(pok)
 cout<<*pok<<endl;
else
 cout<<"null-pokazivac.\n";
```

Ponavljanje

- Komentirajte zadani isječak koda:

```
int polje[]={7,4,12,3,5};
for(int i=0;i<5;i++)
      cout<<*(polje+i)<<endl;
```

Ponavljanje

- Komentirajte zadani isječak koda:

```
int polje[]={7,4,12,3,5};
int *pok = polje;
for(int i=0;i<5;i++)
      cout<<*(pok+i)<<endl;
```

Ponavljanje

- Komentirajte zadani isječak koda:

```
int polje[]={7,4,12,3,5};
int *pok = polje;
for(int i=0;i<5;i++,pok++)
     cout<<*pok<<endl;
```

Ponavljanje

- Komentirajte zadani isječak koda:

```
int polje[]={7,4,12,3,5};
for(int i=0;i<5;i++,polje++)
      cout<<*polje<<endl;
```

Podaci u programu

- Statički podaci (objekti):
  - Zauzimaju memoriju cijelo vrijeme izvršavanja programa
- Automatski podaci (objekti):
  - automatski kreiraju prilikom deklaracije (u funkciji)
  - traju dok se funkcija izvršava (statički su u funkciji)
  - po završetku izvođenja funkcije (ili bloka naredbi unutar kojeg su deklarirani) se uništavaju (dinamički su u odnosu na razinu glavne funkcije).
- Dinamički podaci (objekti):
  - Eksplicitno alocirani i dealocirani za vrijeme izvršavanja programa uporabom C++ naredbi koje piše programer.

Automatski objekti

- engl. *automatic objects*
- Prevoditelj sam alocira memorijski prostor
- Prevoditelj se ‘sam brine’ o oslobađanju memorijskog prostora
- Memorija koju zauzima objekt se oslobađa u trenutku izlaska iz bloka u kojem je objekt deklariran
- Automatski objekti se u memoriji pohranjuju na stogu (engl. *stack*)
  - Zasebni dio memorije u koji se pohranjuju privremeni podaci

Dinamički objekti

- engl. *dynamic objects*
- Alocira ih programer korištenjem operatora **`new`**
- Smještaju se u javni dio memorije koji se naziva gomila (engl. *heap*)
- Prevoditelj ne kontrolira ‘čišćenje’ tog dijela memorije,
- Programer sam mora brinuti o oslobađanju memorije (operator **`delete`**)

Pohranjivanje podataka na stogu

- Stog: Postupak alokacije i dealokacije u nadležnosti je računala i obavlja se automatski.
- Na stog se pohranjuju pozivi funkcija
  - Za svaki poziv funkcije čuva se na stogu jedan okvir (frame).
  - U svakom okviru se čuvaju vrijednosti lokalnih varijabli i vrijednosti parametara/referenci iz poziva.
  - Sa stoga se prosljeđuju povratne vrijednosti pozivajućem kodu.
  - Okvir stoga se briše po završetku izvođenja funkcije.

Pohranjivanje podataka na gomili

- Gomila: Dinamičku alokaciju i dealokaciju obavlja programer uporabom operatora **`new`** (alokacija) i **`delete`** (dealokacija).
- Gomila osigurava dodatnu memoriju
  - Alocira se "koliko treba" (dinamički) programu za vrijeme izvođenja
  - Operator new dodjeljuje memoriju na gomili
  - Alocirana memorija se oslobađa uporabom operatora delete.

Dinamička alokacija memorije

- Dinamička alokacija memorije na gomili izvodi se pomoću pokazivača.

- Dinamička alokacija memorije koristi se:
  - kada se kreira polje čija veličina nije poznata prije pokretanja programa
  - kada je potrebno kreirati složene strukture nepoznate veličine ili oblika za vrijeme izvođenja programa
  - općenito kod kreiranja složenih podatkovnih struktura

Dinamička alokacija memorije

- Dva načina definiranja pokazivača:
  - Deklaracija i usmjeravanje na postojeću varijablu
  - Deklaracija i dinamička alokacija memorije
    - Operator **`new`** zauzima potreban memorijski prostor
      - Ako je operacija alociranja uspješno izvedena, operator vraća adresu alociranog prostora (pridružujemo je pokazivaču)
      - Ako operacija alociranja nije uspješno izvedena, operator vraća nulu, odnosno null-pokazivač
- Sintaksa:

```
ime_tipa *ime_pokazivaca = new ime_tipa();
ime_tipa *ime_pokazivaca = new ime_tipa;
```

Dinamička alokacija memorije

- Primjer. Dinamička alokacija memorije za objekt tipa **`int`**:

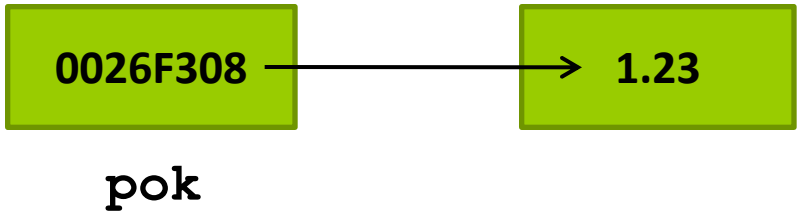
```
int *pok;
pok = new int();
```

- Primjer. Dinamička alokacija memorije za pokazivač na tip **`float`**:

```
float *pok = new float();
```

- Primjer. Deklaracija i inicijalizacija vrijednosti pokazivača koji pokazuje na tip **`float`**:

```
float *pok = new float(1.23) ;
```



Dinamička alokacija memorije

- Alocirani prostor zauzet operatorom new ne oslobađa se automatski prilikom izlaska iz bloka naredbi.
  - Uništavanje dinamičkih objekata obavlja se operatorom **`delete`**.
  - Sintaksa:

```
delete ime_pokazivaca;
```

Dinamička alokacija memorije

- Oslobođen je memorijski prostor, za sadržaj se ne garantira.
- Sadržaj objekta nakon primjene operatora **`delete`** je neodređen.
- Primjer.

```
int *varijablaKojaNestaje = new int(123);
delete varijablaKojaNestaje;
cout<<*varijablaKojaNestaje<<endl;
```

Dinamička alokacija polja

- Koristi se operator **`new`**:
  - Posebno se u uglatim zagradama navodi veličina polja
  - Sintaksa:

```
ime_tipa *ime_pokazivaca= new ime_tipa[broj_elemenata];
```

  - Veličina može biti varijabla
  - Operator **`new`** zauzima potreban memorijski prostor za predviđeni broj elemenata polja
    - Ako je operacija alociranja uspješno izvedena, operator vraća adresu alociranog prostora (pridružujemo je pokazivaču)
    - Ako operacija alociranja nije uspješno izvedena, operator vraća nulu, odnosno null-pokazivač

Dinamička alokacija polja

- Primjer. Dinamička alokacija polja cijelih brojeva:

```
// alokacija za polje veličine 12
int *polje1 = new int[12];

// alokacija za polje veličine n
int *polje2 = new int[n];
```

Dinamička alokacija polja

- Dealokacija polja:
  - operator **`delete`** []
  - iza uglatih zagrada navodi se ime polja
  - Sintaksa:

```
delete [] ime_pokazivaca;
```

- Primjer.

```
// alokacija za polje veličine n
int *polje = new int[n];
...
// dealokacija za polje
delete [] polje;
```

Dinamička alokacija polja

- Operatori **`delete`** i **`delete []`** nisu isti operatori:
  - **`delete`** oslobađa jedan objekt
  - **`delete []`** prvo provjarava duljinu polja, a potom oslobađa memoriju za svaki element polja
- Napomena: ukoliko se zaboravi staviti par uglatih zagrada kod dealokacije polja, prevoditelj neće javiti grešku. Oslobodit će se memorijski prostor za jedan element polja.

Dinamička alokacija polja

- Prednosti dinamičke alokacije polja (u usporedbi sa statičkom alokacijom polja)
  - Veličina polja ne mora biti unaprijed poznata
  - Zauzima se točno onliko memorije koliko je potrebno za pohraniti polje (za razliku od statičkog polja gdje je veličina polja unaprijed zadana)
  - Memorija se oslobađa u trenutku kada se polje više ne koristi

Dinamička alokacija polja

- Obično ne znamo unaprijed koliko prostora je potrebno za podatke programa (na primjer, koliko veliko polje će biti definirano).
- Jedno je rješenje uporaba prevelikog prostora za strukturu podataka:

```
float a[dovoljnoVeliko];
```

  To nije uvijek izvedivo.

- Dinamička alokacija memorije osigurava memoriju kada je potrebno, **za vrijeme izvođenja programa**.
- Programer u potpunosti nadzire trajanje objekta.

Ponavljanje: Pokazivači i strukture

- Primjer strukture:

```
struct STUDENT {
    char ime[30];
    int id;
    int ocjene[3]; };
    ....
    void main() {
    // deklaracija instance strukture
    STUDENT stu;
```

Ponavljanje: Pokazivači i strukture

- Možemo definirati pokazivač na instancu:

```
STUDENT *pok = &stu;
```

- Pristupanje elementu (komponenti) strukture preko pokazivača:

```
(*pok).id = 1999;
```

- ili

```
pok->id = 1999;
```

Ponavljanje: Pokazivači i strukture

1) Dane su deklaracije:

```
struct TipProizvoda {
   int kolicina;
   float cijena;};
TipProizvoda Proizvod1 = {17, 4.99};
TipProizvoda* pokazivac = &Proizvod1;
```

Koji od slijedećih izraza predstavlja ispravni pristup sadržaju podatkovnog člana strukture Proizvod1?

1. (*pokazivac).cijena

2. *pokazivac.cijena

3. pokazivac -> cijena

Dinamička alokacija strukture

- Podatke koji su tipa strukture također je moguće dinamički alocirati primjenom operatora **`new`**.
- Primjer:

```
struct student{
  string ime, prezime;
  int brIndeksa;
};
...
student *pokNaStruct = new student;
  pokNaStruct -> ime = "hrvoje";
  pokNaStruct -> prezime = "horvat";
  pokNaStruct ->brIndeksa = 123;
```

Mogući problemi s dinamičkom alokacijom memorije

- Moguće neželjene pojave koje se mogu javiti kod dinamičke alokacije memorije jesu:
  - Pojava nedostupnog objekta i curenje memorije
  - Pojava visećeg pokazivača
- Te pojave se javljaju ukoliko postupak dinamičke alokacije memorije nije izveden kako treba.

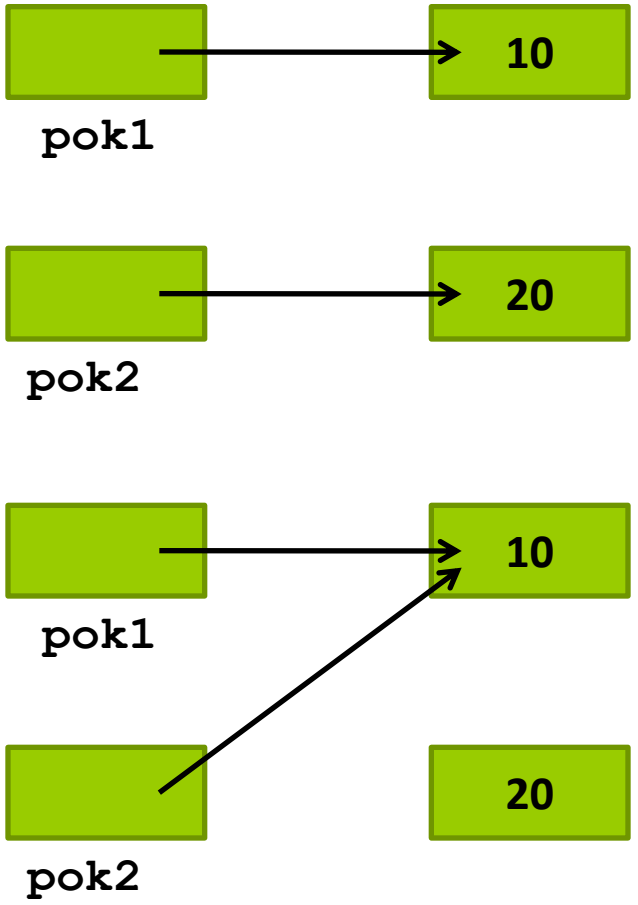
Nedostupan objekt

- Nedostupan objekt je bezimeni objekt koji je kreiran uporabom operatora new, a programer ga je uporabom naredbi ostavio bez pokazivača
- Pokazivač se preusmjeri na neki drugi (postojeći ili novi) objekt, a prvi objekt nije dealociran
- Pogreška logičkog tipa, prevoditelj je ne može otkriti
- Uzrokuje **curenje memorije**
- **Posljedica** curenja memorije je gubitak raspoloživog memorijskog prostora

Nedostupan objekt

```
int *pok1 = new int;
int *pok2 = new int;
*pok1 = 10;
*pok2 = 20;

pok2 = pok1;
```

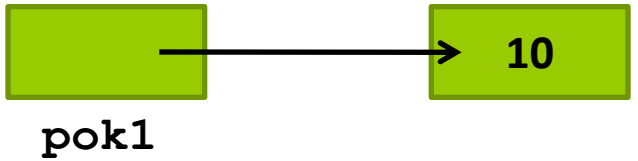


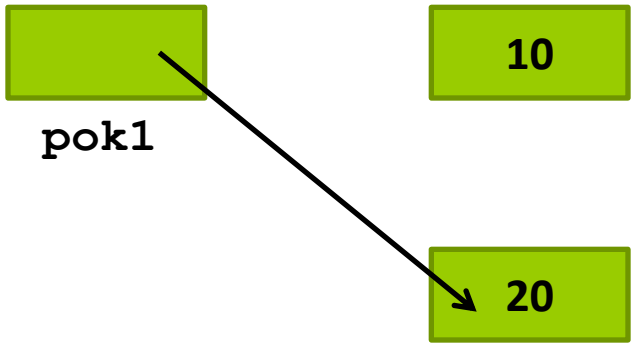
Varijabla koja sadrži broj 20 je bezimena - više joj ne možemo pristupiti, a prostor nije dealociran

Nedostupan objekt

```
int *pok1 = new int;
*pok1 = 10;

pok1 = new int (20);
```





Varijabla koja sadrži broj 10 je bezimena - više joj ne možemo pristupiti, a prostor nije dealociran

Nedostupan objekt

```
int *pok = new int[10];
pok = new int(5);
```

Pokazivač je s polja prusmjeren na lokaciju čija je vrijednos 5.
Polje ostaje nedostupno.
Memorija za 10 elementa je zauzeta.

Nedostupan objekt

- Rješenje problema:
  - prije preusmjeravanja pokazivača na drugi objekt potrebno je dealocirati memoriju
  - a tek potom preusmjeriti pokazivač na drugi objekt
  - primjer:

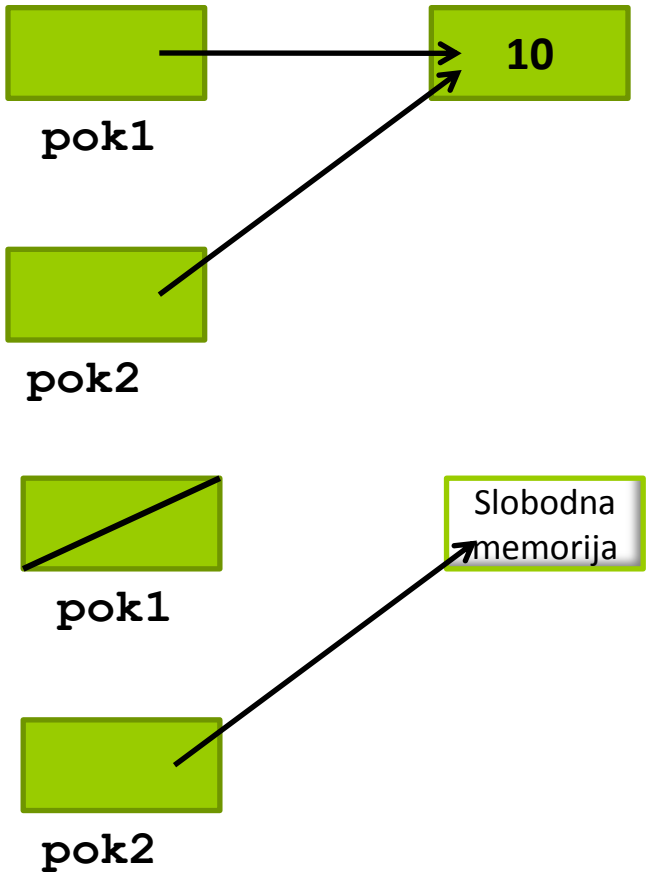
```
int *pok1 = new int;
int *pok2 = new int;
*pok1 = 10;
*pok2 = 20;
delete pok2;
pok2 = pok1;
```

Viseći pokazivač

- engl. *dangling pointer*
- Pokazivač koji pokazuje na:
  - uništeni objekt (na dinamičku memoriju koja je dealocirana)
  - nepostojeći objekt
- Rezultat dereferenciranja visećeg pokazivača je nepredvidiv:
  - Može biti ispis vrijednosti koja se nalazi na zadanoj memorijskoj adresi nakon oslobađanja memorije

Viseći pokazivač

```
int *pok1 = new int;
int *pok2;
*pok1 = 10;
pok2 = pok1;
delete pok1;
pok1 = NULL;
```



Viseći pokazivač

- Primjer.

```
int *pok, i=10;
{
    int j=100;
    pok=&j;
}
//varijabla j više ne postoji
//*pok je viseći pokazivač
cout<<*pok<<endl; //upitan ispis
```

Viseći pokazivač

- Rješenje problema:
  - potrebno je prvo preusmjeriti pokazivač na neku drugu memorijsku adresu, a tek potom izbrisati memorijski prostor
  - Ili definirati takav pokazivač kao null-pokazivač:

```
pok = 0;  //ili pok = NULL;
```

  - Prilikom dohvaćanja varijable preko pokazivača, prethodno je potrebno provjeriti je li pokazivač različit od null-pokazivača

Viseći pokazivač

- Primjer:

```
int *pok1 = new int;
int *pok2;
*pok1 = 10;
pok2 = pok1;
delete pok1;
pok1 = NULL;
pok2 = NULL;
//pok2 definria se kao null-pokazivač,
//pa više nije viseći pokazivač
```

Ponavljanje - Primjer

- Je li ispravno napisan zadani kod:

```
int *pok = new int(10);
cout<<*pok<<endl;
delete pok;
cout<<*pok<<endl;
```

- Dereferenciranje pokazivača koji je oslobođen (dealociran). Nepredviđena vrijednost ispisa.

Ponavljanje - Primjer

- Je li ispravno napisan zadani kod:

```
int *pok = 0;
cout<<*pok<<endl;
```

- Dereferenciranje null-pokazivača uzrokuje trenutni prekid programa.

Ponavljanje - Primjer

- Je li ispravno napisan zadani kod:

```
int a;
int *pok=&a;
delete pok;
```

- Oslobađanje pokazivača na memoriju koja nije dinamički alocirana (npr. pokazivač na statički podatak) uzrokuje trenutni prekid programa.

Ponavljanje - Primjer

- Je li ispravno napisan zadani kod:

```
int *pok=new int (10);
delete pok;
delete pok;
```

- Oslobađanje pokazivača koji je već oslobođen uzrokuje trenutni prekid programa.

Moguće greške

- Moguće greške kod dinamičke alokacije memorije:
  - Zanemarivanje oslobađanja dinamičke memorije (nedostupan objekt, curenje memorije).
  - Pristupi elementima polja izvan zadanih granica.
    - Česta greška je pokušaj pristupanja s pomakom 1

    zbog pogrešnog algoritma indeksiranja.
    - Potrebno je paziti i kod korištenja aritmetike pokazivača u radu s poljima.

Litertura

- B. Motik, J. Šribar: Demistificirani C++
  - 2. ili 3. izdanje
  - Poglavlje 4. Polja, pokazivači, reference
    - 4.2. pokazivači ,...4.6.,...
  - Poglavlje 5. Funkcije
    - 5.4. lista argumenata
      - 5.4.3. pokazivač i referenca kao argument
      - 5.4.4. promjena pokazivača unutar funkcije
    - 5.5. pokazivači i reference kao povratne vrijednosti