Programiranje II

Pokazivači (Pointers)

Varijable

- Naziv varijable možemo posmatrati kao ime određene memorijske lokacije
  - Opseg važenja (engl. scope) varijable određuje kada i kako određena memorijska lokacija može biti ponovo korištena, npr. dvije različite lokalne varijable iz različitih potprograma mogu koristiti istu memorijsku lokaciju u različito vrijeme
  - Tip podatka varijable (npr. int) definiše kako se bitovi u memorijskoj lokaciji interpretiraju i koje operacije se mogu vršiti na toj varijabli
- Pored naziva, svaka varijabla posjeduje svoju adresu

Pokazivači

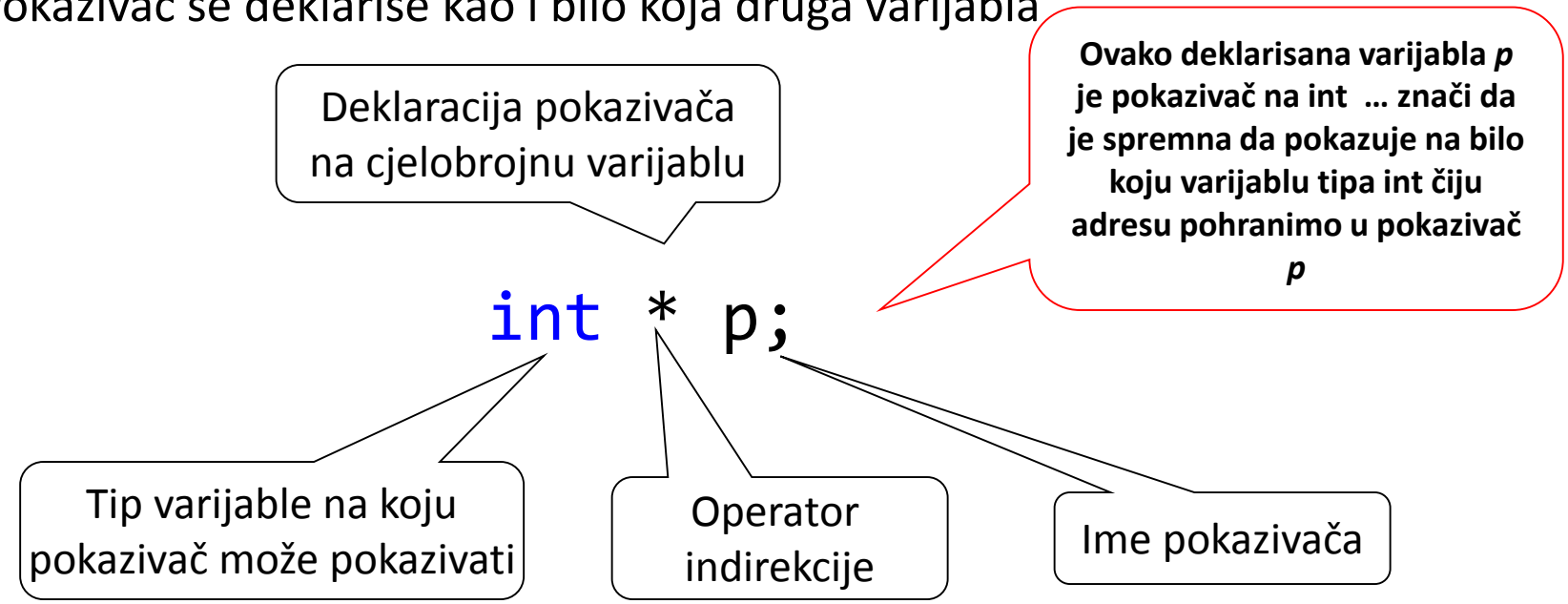
- Pokazivači se smatraju veoma moćanim, fleksibilnim i opasim alatom koji postoji u programskim jezicima C i C++
- Većina drugih programskih jezika (npr., Java, Pascal) programerima ne pružaju mogućnost korištenja pokazivača
- Za sigurno raspolaganje programerskim alatom kakav je pokazivač, jako je bitno razumjeti suštinu pokazivača i načina njihovog funkcionisanja.
- **Pokazivači su varijable koje sadrže (čuvaju kao vrijednost) memorijsku adresu neke druge lokacije**, te se koriste za:
  - Manipulaciju poljima (posebno string-ovima)
  - Vraćanje više rezultata iz funkcije
  - Pristup varijablama koje inače nisu vidljive funkcijama
  - Predavanje adresa drugih funkcija
  - Dinamičku dodjelu i manipulaciju memorijom

Pokazivači i adrese

- Svaka varijabla se nalazi na nekom mjestu u memoriji računara koje je jedinstveno određeno svojom adresom (rednim brojem mjesta)
- Za svaki tip T definiše se tip T* (pokazivač na T)
- Varijabla tipa T* može kao vrijednost imati adresu bilo koje varijable tipa T
- Varijabla tipa T* može kao vrijednost imati posebnu konstantu NULL
- Adresa varijable se dobija korištenjem adresnog operatora (unarni &)
- Operator indirekcije (unarni *) inverzan je adresnom operatoru (dereferencira pokazivač)
- Zabranjeno je dereferenciranje pokazivača čija je vrijednost NULL ili vrijednost nije definisana (jer nije usmjeren na neku varijablu)

Pokazivači

- Pokazivač se deklariše kao i bilo koja druga varijabla



- Tip varijable na koji pokazivač pokazuje koristi kompajler za ispravan pristup memoriji (tipu podatka)
- Adresu neke varijable je moguće dobiti koristeći operator **&**, pa se pokazivač može inicijalizovati na sljedeći način:

```
tip_podatka * ime_pokazivaca = &ime_varijable;
```

Pokazivači - dereferenciranje

- Ukoliko pokazivač ***p*** sadrži (čuva) adresu druge varijable ***x***, onda se toj varijabli može pristupiti na dva načina:
  - Koristeći ime varijable: `x = 42;`
  - "Dereferenciranje" pokazivača: `*p = 42;`
- Navedene dodjele znače potpuno isto
- Drereferencirani pokazivač može zamijeniti ime varijable u svakom iskazu

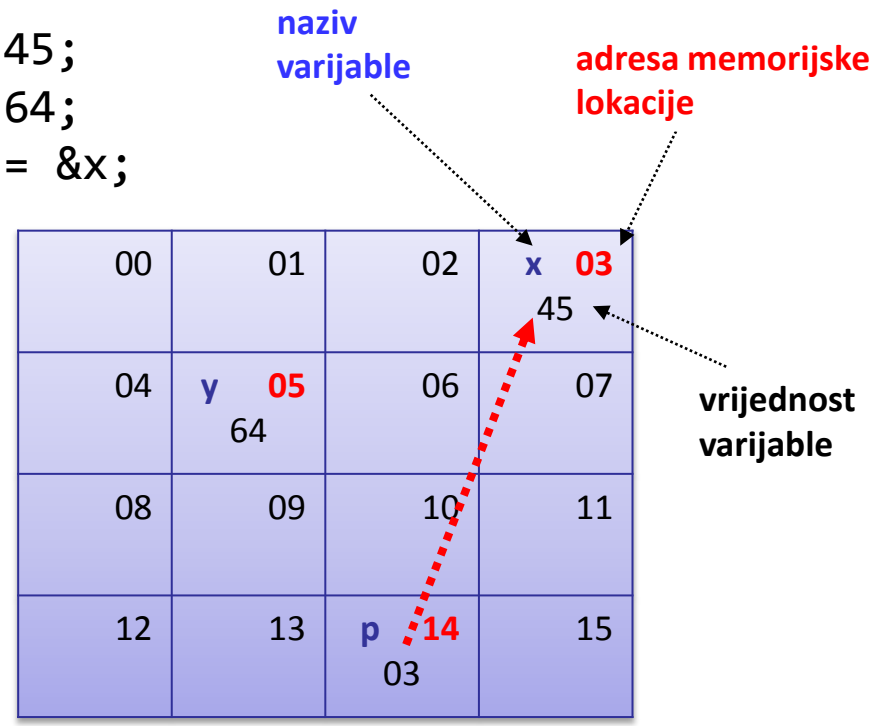
Pokazivači

- U narednim tabelama je prikazan primjer ogranizacije memorijskog prostora u odnosu na naziv, vrijednost i adresu varijable.

Zamislite računar koji bi imao sljedeći memorijski prostor (sistem 2^4)

00	01	02	03
04	05	06	07
08	09	10	11
12	13	14	15

```
int x = 45;
int y = 64;
int * p = &x;
```



- Pokazivač **p** čuva adresu varijable **a** (koja je u ovom slučaju **03**).

Pokazivači

- NULL pokazivač je pokazivač čija je vrijednost nula (0) jer konstanta NULL ima vrijednost 0. NULL pokazivač ne pokazuje ni na jedan objekat odnosno memorijsku lokaciju. Upravo zbog toga je dobra praksa da se na samom početku pokazivači inicijalizuju sa NULL.

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

void main(){
    int x = 45;
    int y = 64;
    int * p = NULL;
    p = &x; //pokazivac p sada pokazuje na varijablu x
    cout<<"x = "<<x<<" &x = "<<&x<<endl;
    cout<<"y = "<<y<<" &y = "<<&y<<endl;
    cout<<"p = "<<p<<" &p = "<<&p<<" *p = "<<*p<<endl;
}
```

Pokazivači

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

void main(){
    int *p = NULL; // deklaracija pokazivača p na int
    int x = 3;  // deklaracija varijable x tipa int
    int y = 4;  // deklaracija varijable y tipa int

    p = &x;         // p pokazuje na x
    x = x + *p;     // udvostručuje x
    p = &y;         // sada p pokazuje na  y
    *p = 42;        // y je dodjeljena vrijednost 42 (x je nedirnut).

    //OBRATITE PAŽNJU NA NAREDNU PETLJU
    p = &x;
    while (*p == x) {
      *p = *p + 1;
    }
    //KOLIKO PUTA SE IZVRŠAVA I ZAŠTO?
}
```

Pokazivači

- Pokušajte predvidjeti izlaz iz narednih primjera.

```
int i = 10, j = 20;
int *p1, *p2;
p1 = &i;
p2 = p1;
*p1 = i;
*p2 = j;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

```
int i = 10, j = 20;
int *p1, *p2;
p1 = &i;
p2 = &j;
j = *p1;
i = *p2;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

```
int i = 10, j = 20;
int *p1, *p2;
p1 = &i;
p2 = &j;
*p1 = i +10;
*p2 = i +10;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

```
int i = 10, j = 20;
int *p1, *p2;
p1 = &i;
*p2 = *p1;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

```
int i = 10, j = 20;
int *p1, *p2;
p1 = &i;
p2 = p1;
j = *p2;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

```
int   i = 10, j = 20;
int  *p1, *p2;
p1 = &i;
p2 = &j;
*p1 = *p2 +10;
*p2 = p1 +10;
//sta ispisuje?
cout<<*p1<<" "<<*p2<<endl;
```

Operacije na pokazivačima

- Pokazivačima može biti dodijeljena vrijednost kao i svim drugim varijablama, ali je dereferenciranje (*) karakteristično samo za pokazivače
- Pokazivači se mogu uspoređivati operatorima `==, !=, <, >` ....
  - Dva pokazivača su "jednaka" ako pokazuju na istu memorijsku lokaciju (varijablu)
  - Pokazivač ***p*** je "manji od" nekog pokazivača ***q*** ako je adresa sadržana u ***p*** manja od adrese sadržane u ***q***

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = &i;
if(p1==p2)
        cout<<"ISTI SU"<<endl;
```

Aritmetika pokazivača

- Pokazivači se mogu koristiti u izrazima sa sabiranjem i oduzimanjem, ali takvi izrazi imaju smisla samo kada pokazivač pokazuje na jednu od lokacija unutar niza!
- Dodavanjem vrijednosti ***k*** pokazivaču ***p*** izračunava se adresa na koju će ***p*** pokazivati.

```
int niz[10] = {0};
int * p = &niz[0];
if(p==&niz[0])
        cout<<"onda je: (p+4)==&niz[4]"<<endl;
```

- Negativni cijeli broj može se, također, dodati...?

Aritmetika pokazivača

- Pokazivači su iste veličine
- Na većini računara pokazivač varijabla je veličine 4 bajta (32 bita)
- Varijabla na koju pokazivač pokazuje može biti različite veličine:
  - char* pokazivač pokazuje na varijablu veličine 1 bajt
  - double* pokazivač pokazuje na varijablu veličine 8 bajta
- U izvršavanju aritmetike pokazivača aktualna adresa sadržana u pokazivač varijabli izračunava se na bazi veličine tipa varijable na koju pokazuje, što se najbolje može uočiti na primjeru sa nizovima

Pokazivači

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

void funkcija(int* p) {
    cout<<"p = "<<p<<endl;
    cout<<"*p = "<<*p<<endl;
    *p = 5;
    cout<<"p = "<<p<<endl;
}

void main() {
    int x = 47;
    cout<<"x = "<<x<<endl;
    cout<<"&x = "<<&x<<endl;
    cout<<"------------------"<<endl;
    funkcija(&x);
    cout<<"------------------"<<endl;
    cout<<"x = "<<x<<endl;
}
```

Pokazivači

- Funkcija ***funkcija***() kao argument prihvata pokazivač, te ga dereferencira pri dodjeli što uzrokuje modifikaciju vanjskog objekta **x**. Mogući izlaz iz programa je sljedeći:

```
x = 47
&x = 001BFB0C
-----------------
p = 001BFB0C
*p = 47
p = 001BFB0C
-----------------
x = 5
```

- Vrijednost sadržana u **p** je ista kao i adresa od **x** – pokazivač **p** zaista pokazuje na **x**.
- Kada je **p** dereferenciran dodjelom vrijednosti 5, vrijednost **x** je također promijenjena na 5 .

Pokazivači

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

void zamijeni(int * x, int * y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
void main(){
    int a, b;
    cout<<"Unesite dva broja: ";
    cin>>a>>b;
    cout<<"a = "<<a<<" b = "<<b<<endl;
    zamijeni(&a, &b);
    cout<<"Nakon zamjene->"<<endl;
    cout<<"a = "<<a<<" b = "<<b<<endl;
}
```

`void` pokazivač

- Pri radu sa pokazivačima često se pominje pravilo da pokazivač mora biti istog tipa kao i varijabla na koju pokazuje

```
int x = 47;
float * p = &x; // ← greska
//error C2440: 'initializing' : cannot convert from
//'int *' to 'float *'
```

- Ipak, postoji pokazivač koji može čuvati adresu bilo kojeg tipa podatka, a on se naziva `void` pokazivač

```
int x = 47;
void * p = &x; // ← dozvoljeno ☺
```

`void` pokazivač

- Uzimajući u obzir činjenicu da teško možemo manipulisati `void` objektima, prije upotrebe lokacije čiju adresu čuva `void` pokazivač potrebno je uraditi cast u stvarni tip podatka koji se nalazi na toj lokaciji

```
int x = 47;
void * p = &x;
//*p = 100;
*((int*)p) = 100;
cout<<"x = "<<x<<endl;
```

`void` pokazivač

- Cast `((int*)p)` preuzima `void*` i naglašava kompajleru da ga tretira kao `int*`, što omogućava dereferenciranje
- Pomenuto se može posmatrati i kao "rupa u sistemu tipova" jer dozvoljava identično tretiranje različitih tipova podataka
- U navedenom primjeru je vrijednost tipa `int` tretirana kao `int`, ali tek nakon cast-a pokazivača `p` u `int*`.
- Umjesto `int`, cast pokazivača `p` smo mogli uraditi u `char*` ili `double*` čime bismo pogrešno tumačili količinu dodijeljene memorije i uzrokovali dodatne probleme
- Zbog svega pomenutog `void` pokazivač se koristi samo u izuzetnim slučajevima
- Ne postoje `void` reference

void pokazivač

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

void povecaj(void* podatak, int velicina){
    switch (velicina){
        case sizeof(char): (*((char*)podatak))++; break;
        case sizeof(int):  (*((int*)podatak))++; break;
    }
}
void main(){
    char slovo = 'a';
    int broj = 20;
    cout<<"broj = "<<broj<<" slovo = "<<slovo<<endl;
    povecaj(&slovo,sizeof(slovo));
    povecaj(&broj,sizeof(broj));
    cout<<"broj = "<<broj<<" slovo = "<<slovo<<endl;
}
```

Pokazivači

- Kakav izlaz se može očekivati iz narednog programa

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

void main() {
    int brojevi[10];
    int * p = &brojevi[0];
    int * prvi_adresa = p;
    int * zadnji_adresa;
    int i = 0;
    while (p != &brojevi[10]) {
        *p = 42 + i;
        p++;
        i++;
    }
    zadnji_adresa = p - 1;
    cout<<*zadnji_adresa<<endl;
}
```

`const` kvalifikator i pokazivači

- `const` kvalifikator
  - Varijabla se ne može promijeniti
  - `const` se koristi ako funkcija ne treba da mijenja varijablu
  - Pokušaj izmjene `const` varijable uzrokuje grešku
- `const` pokazivači
  - Pokazuju na "konstantnu memorijsku lokaciju"
  - Mora biti inicijalizovan pri deklaraciji
  - `int * const p = &x;`
    - konstantni pokazivač na `int`
  - `const int * p = &x;`
    - pokazivač na konstantni `int`
  - `const int * const p = &x;`
    - konstantni pokazivač na konstantni `int`
    - vrijednost **x** može biti promijenjena, ali ne preko ***p**

`const` kvalifikator i pokazivači

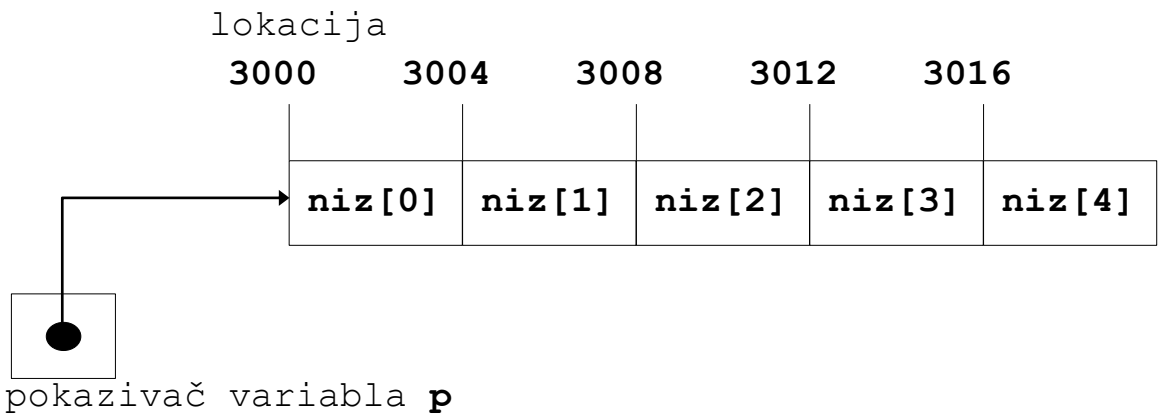
```
#include <iostream>
using namespace std;

void main() {

    int x = 10;
    int y = 20;

    int * const p = &x;

    *p = 33; // dozvoljeno
    //p = &y; // ← greska
    cout<<"x = "<<x<<" y = "<<y<<endl;
}
```

Pokazivač na niz

- Niz od 5 elemenata tipa int

```
int niz[5];
int * p = niz;
```

  - `p` pokazuje na prvi element niza tj. `niz[0]`
  - Na lokaciji 3000 pokazivač `p=3000`
  - `p+=2;` postavlja `p` na `3008` tj. `p` pokazuje na `niz[2]` (inkrementiran sa 2),
    - Rezervišu se 4 byte/int, pa pokazuje na adresu 3008



Pokazivač na niz

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

void main(){
    int niz[10] = {2,4,6,8,10,12,14,16,18,20};
    int *p1 = &niz[2];
    int *p2 = &niz[6];

    cout<<*p1<<endl;
    cout<<*p2<<endl;
    cout<<p2 - p1<<endl;
    cout<<p1 - p2<<endl;
}
```

Pokazivač na niz

```
#include<iostream>
using namespace std;
void main(){
    double niz[5] ={5.2, 5.3, 8, 9.5, 3.9};
    double * p1 = niz ;
    for (int i=0;i<5;i++){
    //realizovati for petlju bez brojaca i
        cout<<*p1<<", ";
        p1++;
    }
    //ispisuje vrijednost drugog elelmenta uvecanu za 3
    cout<<endl<<(niz[1]+3)<<endl;
    double* p2 = niz;
    cout<<*(p2+3)<<endl; //cetvrti element niza
    cout<<p2[3]<<endl;   //cetvrti element niza
    cout<<niz[3]<<endl;  //cetvrti element niza
}
```

KRAJ PREZENTACIJE