Kazalci

1

- spremenljivke, kot smo jih uporabljali sedaj, predstavljajo konkretno vrednost
- v primeru razrednih tipov, predstavljajo konkreten primerek danega razreda – objekt
- ime spremenljivke nam predstavlja vrednost/objekt

```
int x = 10;
CLok mojLok(10, false, "oranžen");
```

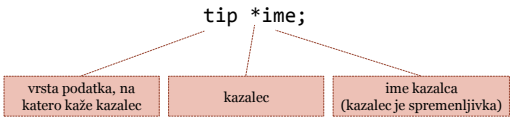
- kazalec je spremenljivka, ki vsebuje naslov pomnilniške lokacije
  - torej vrednost v kazalcu je **naslov**
  - ta naslov nam običajno pove, kje najdemo vrednost
  - pravimo, da kazalec kaže na lokacijo, kjer najdemo podatek/vrednost

Kazalci - deklaracija

2

- kazalec deklariramo na naslednji način

```
tip *ime;
```



- tip kazalca pove, kakšno vrsto podatka najdemo na lokaciji, na katero kaže kazalec
- pravimo tudi, da kazalec kaže na tip
- deklaracija kazalca ne inicializira kazalca
  - torej kazalec kaže na neko **nedefinirano** lokacijo

Kazalci - primeri

3

```
int *p;   // kazalec na pomnilniško lokacijo
          // tam najdemo celo število
int *q, x; // pozor, samo q je kazalec
           // x je običajna celošt. spr.
int *a, *b; // a in b sta kazalca na celo št.
```

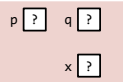
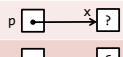
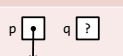
Kazalci – osnovne operacije

4

- poseben kazalec NULL (lahko vrednost `0`)
  - ne kaže nikamor
  - definiran v knjižnici *cstdlib*
- operator & (referenca)
  - z njim dobimo naslov nekega objekta v pomnilniku
- operator * (dereferenca)
  - zapisan pred kazalcem nam s kazalcem vred predstavlja vrednost na katero kaže kazalec
- operator new
  - sami rezerviramo pomnilniški prostor
- operator delete
  - sami sprostimo pomnilniški prostor

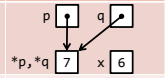
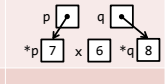
Kazalci – osnovne operacije: primer 1

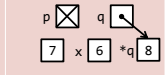
5

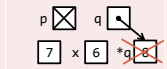
C++ koda	Kako si narišemo?	Opis
`int *p, *q;` `int x;`		Za spremenljivke p, q in x se rezervira pomnilnik ob prevajanju/zagonu programa. Takim spremenljivkam pravimo statično ustvarjene spremenljivke.
`p = &x;`		p kaže na spremenljivko x (p ima shranjen naslov od x)
`*p = 6;`		Na lokacijo, kamor laže p, shrani vrednost na desni strani prirejanja.
`p = new int;` `*p = 7;`		Rezerviramo pomnilnik za novo celo število (dinamično ustvarjena spremenljivka). Na ta novi pomnilnik shrani vrednost 7. Paziti moramo na ujemanje tipov!

Kazalci – osnovne operacije: primer 1

6

C++ koda	Kako si narišemo?	Opis
`q = p;`		Prirejanje vrednosti kazalcu pomeni, da povemo, kam ta kazalec kaže. V tem primeru kaže q na isto lokacijo kot p.
`q = new int;` `*q = 8;`		Rezerviramo pomnilnik za novo celo število. Na ta novi pomnilnik shrani vrednost8.

`p = NULL;`
`// ali`
`p = 0;`
		Kazalec `0` oz. `NULL` pove, da p ne kaže na nobeno pomnilniško lokacijo. Do lokacije, kjer je shranjeno št. 7, ne moremo več dostopati.

`delete q;`		Sprosti pomnilnik, na katerega kaže kazalec q. Torej q kaže na lokacijo, do katere nimamo več dostopa.

Kazalci – osnovne operacije: primer 2

7

C++ koda	Kako si narišemo?	Opis
`int *p, *q;`	p ? q ?	Deklariramo kazalca na cela števila. Ne kažeta na obstoječi objekt.
`p = new int;`	p ? q ?	p kaže na novo rezerviran prostor za celo število (na tisi lokaciji še ni znane vrednosti)
`*p = 1;`	p 1 q ?	Na lokacijo, kamor laže p, shrani vrednost na desni strani prirejanja, torej 1.
`q = new int;` `*q = 2;`	p 1 q 2	Rezerviramo pomnilnik za novo celo število. Na ta novi pomnilnik shrani vrednost 2.
`cout << *p;`		Izpiši vrednost na lokaciji, na katero kaže p, torej 1.

Kazalci – osnovne operacije: primer 2

8

C++ koda	Kako si narišemo?	Opis
`*p = *p + *q;`	p 3 q 2	Na lokacijo, kamor kaže p, shrani vsoto trenutne vrednosti na tej lokaciji in vrednosti na lokaciji, kamor kaže q.
`cout << *p;`		Izpiše 3.
`p = q;`	p 3 q 2	V kazalec p shrani isti naslov, ki je shranjen v kazalcu q. Pravimo: p naj kaže na isto mesto kot q. Mesto, kjer je shranjena 3 smo izgubili!
`cout << *p;`		Izpiše 2.

`*p = 7;`
`// ali`
`*q = 7;`
	p 3 q 7	Na mesto, kamor kaže p oz. q, shrani vrednost 7. Oba kažeta na isto lokacijo, torej je vseeno, katerega uporabimo.

Kazalci – osnovne operacije: primer 2

9

C++ koda	Kako si narišemo?	Opis
`p = new int;`	p ? 3 q 2	Rezervirajmo nov prostor za eno celo število.
`delete p;` `p = NULL;`	p ~~?~~ 3 q 2	Sprosti pomnilnik, na katerega kaže p. Torej je spet na voljo. Povej, da p ne kaže nikamor. To je **pravilni način** sproščanja pomnilnika. Če imamo lokacijo, na katero na kaže več noben drugi kazalec, jo sprostimo, preden spremenimo kazalec!
`q = NULL;`	p ~~?~~ 3 q 2	To je **napačen** način! Izgubili smo lokacijo, na kateri je shranjeno število 2, lokacija ni več na voljo.

Kazalci na objekte

10

- kazalec lahko kaže na poljubni tip, torej tudi razred
- nov objekt razrednega tipa CRazred ustvarimo tako:

```
CRazred *x = new CRazred();
```

podatkovni tip je CRazred

gre kazalec, ime kazalca je x

ustvari nov objekt tega tipa, kličemo **poljubni konstruktor** tega razreda

- samo deklaracija kazalca NE USTVARI objekta:

```
CRazred *x;
```

nimamo objekta! kazalec kaže na neznano lokacijo

Dostop do komponent objekta preko kazalca

11

- dostop do razrednih spremenljivk in funkcij objekta:

```
(*x).imeSpremenljivke
(*x).imeFunkcije(parametri)
```

ali

```
x->imeSpremenljivke
x->imeFunkcije(parametri)
```

- mi bomo uporabljali operator `->`
  - sestavljen iz dveh znakov
  - med njima ne sme biti presledka

Brisanje objektov

12

- če objekt ustvarimo dinamično, moramo sami tudi zbrisati objekt iz pomnilnika
  - ustvarimo dinamično pomeni uporabo kazalca in operatorja new
- objekt, na katerega kaže kazalec *ime*, zbrišemo tako:

```
delete ime;
```

- ko brišemo objekt iz pomnilnika, se samodejno kliče destruktor razreda

Kazalec this

13

- v vsakem razredu je definiran tudi kazalec *this*
- kazalec `this` kaže na objekt, nad katerim izvajamo metodo
- kazalec `this` se dejansko prenese kot argument metode
- uporablja ga lahko tudi programer

PRIMER: razred CTocka2D (tocka2d.h)

14

```
class CTocka2D {
public:
  CTocka2D();
  CTocka2D(double x, double y);
  ~CTocka2D();
  double vrniX() { return x; }
  double vrniY() { return y; }
  void doloci(double x, double y);
  void translacija(double dx, double dy);
  void izpis();

private:
  double x;
  double y;
};
```

kratke metode lahko implementiramo kar v razredu

PRIMER: razred CTocka2D (tocka2d.cpp)

15

```
CTocka2D::CTocka2D() {
  x = 0;
  y = 0;
}

CTocka2D::CTocka2D(double x, double y) {
  doloci(x, y);
}

CTocka2D::~CTocka2D() {
}

void CTocka2D::izpis() {
  cout << "(" << x << ", " << y << ") ";
}
```

PRIMER: razred CTocka2D (tocka2d.cpp)

16

```
void CTocka2D::doloci (double x, double y) {
  this->x = x;
  this->y = y;
}

void CTocka2D::translacija (double dx, double dy) {
  x += dx;
  y += dy;
}
```

razredne spremenljivke lahko uporabimo preko kazalca **this**

spremenljivki x in y sta lokalni spremenljivki, definirani v parametrih, in razredni spremenljivki

prednost imajo lokalne spremenljivke

ker ni lokalne spremenljivke s tem imenom, ni dvoumnosti glede imen spremenljivk, zato lahko kazalec *this* izpustimo

PRIMER: razred CTocka2D (main.cpp)

17

```
#include "tocka2d.h"

int main() {
  CTocka2D *tockaA, *tockaB;
  tockaA = new CTocka2D();
  tockaB = new CTocka2D(-0.5, 2.13);

  tockaA->izpis(); //  ne znamo: cout << tockaA;
  tockaB->izpis();

  delete tockaA;
  delete tockaB;

  return 0;
}
```

kazalca na točko (objekt še ne obstaja)

ustvari novo točko (privzeti konstruktor)

ustvari novo točko (inicializacijski konstruktor)

operator << ni definiran za razred CTocka

sami smo ustvarili objekte, zato jih moramo sami zbrisati s pomnilnika

PRIMER: razred CPremica (premica.h)

18

```
#include "tocka2d.h"

class CPremica {
public:
  CPremica();
  CPremica(double a, double b, double c);
  ~CPremica();

  bool jeVzporedna(CPremica* p);
  CTocka2D* vrniPresecisce(CPremica* p);

private:
  // premico si predstavimo v implicitni obliki
  // a*x + b*y + c = 0
  double a, b, c;
};
```

kazalce lahko uporabimo tudi kot parametre funkcij ali kot rezultat funkcije

zakaj je slednje smiselno?

PRIMER: razred CPremica (premica.cpp)

19

```
CPremica::CPremica() {
  a = 0; b = 1; c = 0; // premica y=0
}

CPremica::CPremica(double a, double b, double c) {
  if (a==0 && b==0) { // če premica ne obstaja
    cout << "Podatki ne predstavljajo premice.";
    exit(-1);
  }
  else {
    this->a = a;
    this->b = b;
    this->c = c;
  }
}

CPremica::~CPremica() { }
```

Objekt mora vedno predstavljati veljaven primerek razreda. Če ne, zaključimo program z napako.

exit(i) ... zaključi program z vrednostjo i, definirana je v knjižnici *cstdlib*

PRIMER: razred CPremica (premica.cpp)

20

```
/*
   Ali je premica, nad katero izvajamo metodo, vzporedna s
   premico, na katero kaže kazalec p?
*/
bool CPremica::jeVzporedna(CPremica* p) {
  if (p==NULL) // če p ne kaže nikamor, ni premice
    return false;
  if (this->b==0 && p->b==0) // obe premici sta oblike x = m
    return true;
  else { // lahko izračunamo smerni koeficient
    double k1 = - (this->a/this->b); // ali a/b
    double k2 = - (p->a/p->b);
    if (k1 == k2)
      return true;
  }
  return false;
}
```

PRIMER: razred CPremica (premica.cpp)

21

```
/*
   Vrne točko, ki je presečišče premice, nad katero izvajamo metodo,
   in premice, na katero kaže kazalec p. Če točka obstaja, sicer
   vrnemo kazalec NULL, tako vemo, da take točke ni.
*/
CTocka2D* CPremica::vrniPresecisce(CPremica* p) {
  if (p==NULL) // če p ne kaže nikamor, ni premice, ni presečišča
    return NULL;
  if (this->jeVzporedna(p)==true) // premici sta vzporedni
    return NULL;
  else { // lahko izračunamo presečišče
    // dodajte kodo, ki izračuna presečišče premic
    // pazite na možne posebne primere!!!
    // double x = ...;
    // double y = ...;
    CTocka2D *presecisce = new CTocka2D(x,y);
    return presecisce;
  }
}
```

NALOGA: Dopolni funkcijo tako, da bo pravilno izračunala presečišče dveh premic!

PRIMER: razred CTocka2D (main.cpp)

22

```
#include "tocka2d.h"
#include "premica.h"
...

int main() {
  CTocka2D *tockaA = NULL;
  CPremica *p, *q;
  p = new CPremica(-1, 1, 0); // y = x
  q = new CPremica(-2, 1, -1); // y = 2x+1
  tockaA = p->vrniPresecisce(q);
  if (tockaA != NULL) // če obstaja presečišče
    tockaA->izpis();
  else
    cout << "Premici se ne sekata." << endl;

  delete tockaA, p, q;
  return 0;
}
```