

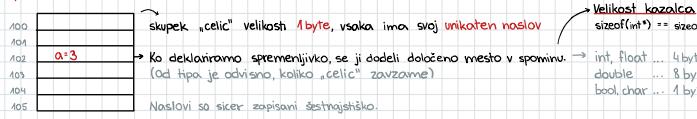
Kazalci

petek, 03. januar 2025 17:27

1. Osnovni koncept

Kazalci (pointers) so spremenljivke, ki hranijo naslov v spominu drugih spremenljivk.

Spomin računalnika:



Zgled:
int x = 20;
int *ptr = &x;
int value = *ptr;

X ... vrednost, ki jo hrani x //20
&x ... naslov x v spominu //412
ptr ... naslov, ki ga hrani ptr //412
*ptr ... vrednost na katero kaže ptr //20

→ Kaj pa če imamo še eno spremenljivko?
int y = 12;
{ *p = y;
spremenili se bo vrednost na naslovu &X (412)
Kazalec p ne bo kazal na naslov y, ker še vedno hrani naslov &x!

→ Velikost kazalca je vedno enaka!
sizeof(int*) == sizeof(double*)

→ Če želimo spremeniti vrednost na naslovu 412:
*ptr = 14;
cout << X; //14
→ Računanje?
int a = 10;
int *p;
p = &a; ← recimo, da je ta vrednost 2002
cout << p+1; //2006 - Zakaj? Tip int zavzame 4 byte
vrednost na tem naslovu pa ni (iz ni dodeljena)

Ponovitev:

a) Prenos po vrednosti: vrednost dejanskega parametra se kopira in s tem ne vpliva na prvotno vrednost (sprememba znatne funkcije ne vpliva na vrednosti zunaj nje)

Primer:

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

```
void increment (int num) {
    num++;
}
```

```
int main () {
    int number = 5;
    increment (number);
    cout << number << endl; //Izpiše 5
    return 0;
}
```

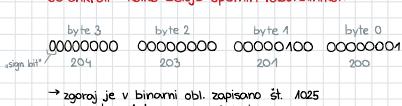
b) Prenos po referenci: vrednost dejanskega parametra, ki se prenese v funkcijo isti naslov v spominu.

Zagonji primer:

```
void increment (int& num) {
    ...
    // Na koncu bi izpisalo povečano vrednost
```

„Dereferenciranje“ - dostop do vrednosti na naslovih v spominu in sprememjanje le-teh

Še enkrat: kako deluje spomin računalnika?



```
int a = 1025;
int *p;
p = &a; //200
```

int *NULL pointer*: ne kaže nikamor („prazen“)
isto: int *p = NULL;
int *p = 0;
int *p = &null;

→ Inicializacija kazalca: če ga inicializiramo na 0 se izognemo napakam

Določilo const: zagotovimo, da se kazalec ali spremenljivka ne spreminja.
↳ PAZI! vsebinsko kazalec = vrednost, na katero kaže

a. kazalec na konstantno spr.: const int *pstevilo = &stevilo;

*stevilo = 5; // NAPAKA

pStevilo =&stevilo2;

lahko spremijnamo kazalec, ne pa vrednosti, na katero kaže

b. konstantni kazalec

int const *pstevilo = &stevilo;

*stevilo = 5;

pStevilo = &stevilo2; // NAPAKA

obratno kot pri a.

Kazalec na kazalec

```
int x = 5
int *p;
*p = 8x;
*p = 6
int **q;
*q = p;
cout << q; //225 (ker je &x = 225)
cout << **q; //6 - dvojno dereferenciranje (glejamo nazaj)
```

2. Kazalci in polja

```
int arr[5]={10,200,3,14,7}
ime polja
arr ... konst. kazalec, ki hrani naslov 1. el. (index 0)
&arr[1] ali arr+1 ... naslov i-tega el. v polju
arr[1] ali *(arr+1) ... vrednost na tistem naslovu
cout << arr; //200 (&arr[0])
5x 4 byte za 5 int elementov
```

Kazalec lahko pristevamo celo število.

↳ kazalec se premakne za toliko mest, kot je velikost tipa

↳ pri poljih lahko vemo, kaj se nahaja na tistem naslovu

celo število

Polja znakov (nizi):

Velikost polja ≥ št. znakov + 1

char C[8]; J A H E Z [0] c[5] *10;

označuje koniec niza.

char string[] = "Srečno novo leto"; //string... naslov 1. elementa
char *string = "Srečno novo leto"; //string... kazalec, ki hrani naslov polja.

Zgled 2:

```
char c1[6] = "Hello"; //polje
char *c2; //kazalec
c2 = c1;
C2[5] = 'A'; //dobjimo 'Hello'
C2[5] je *(C2+1); C2++; //napaka!!!
C2++ je *(C2+1); C2++; //premik na naslednji el.
```

Ne pozabi: polja se v funkcijo privzeto prenosa po referenci.

↳ funkcija prejme naslov (eli polja)

void fun(char C1[]) { void fun(char *C) }

3. Dinamično dodeljevanje pomnilnika

new } omogočata, da sam dolžimo živilo spremenljivke
delete dolžimo lahko tudi, koliko pomnilnika želimo rezervirati

ne glede na doseg

spremenljivka z operanjem new ima živilo, dobo do delete

kazalec = new podatkovniTip

ustvari pomni lokacijo za podatkovniTip → ta naslov nadalje shranimo kot kazalec do takoj ustvarjenem spr. lahko dostopamo samo prek kazalca.

Spomin razdelimo na:

Heap	dynamic memory
Stack	klici funkcij, lokalne spr.
Static/Global	globalne spr.
Code	navigacija

POMEMBNA razlika:
double num = 5.31;
prostor se dodeli med PREVAJANJEM in imenoma, IMENOM "num"

double plnum* = new double (5.31);
prostor se dodeli med IZVANJANJEM in NIMA IMENA - dosegamo preko kazalca.

#include <iostream>
using namespace std;

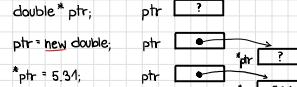
```
int main () {
    double* ptr;
    ptr = new double;
    *ptr = 5.31;
```

delete ptr;

return 0;
}

8ptr ... naslov kazalca
ptr ... vsebinsko kazalca
*ptr ... dereferenciran kazalec

S sliko:



Incializacija hkrati z rezervacijo pomnilnika:
double* ptr;
ptr = new double(5.31);

Ustvarjanje polja: double pArray = new double[size];
delete [] pArray;

→ „Dinamično“ polje
-ma toliko elementov, kot jih potrebujemo pri izvajanju

Ustvarjanje polja: double pArray = new double[size];
delete [] pArray;
~~~~~ prevajalnik si velikost zapomni zato je ne potrebno spet pisati

Funkcija main ()

→ prototip:

```
int main (int argc, char** argv)
```

št. argumentov v  
ukazni vrstici OS, ko  
program zaženemo

Kozalec na polje nizov  
- hrani argumente, kijih  
padavamo ob zagovu

---

```
int main (int argc, char** argv) {
```

cout << argc << endl; // 3

---

```
for (int i = 0; i < argc; i++) {
```

cout << argv[i] << endl; // imedatoteke

}

return 0; // 24

← „Dinamično“ polje  
- ima toliko elementov, kot jih potrebujemo pri izvajanju  
(pri statičnem je št. elementov vnaprej določeno pri prevajanju)

**Splošni kozalci:** `void*`  
↳ kozalec tega tipa lahko služi kot splošni ali univerzalni kozalec  
Uporabljamo, ko imamo v funkciji argumente različnih tipov  
→ `void*` lahko priredimo kozalec na poljuben tip:

int num = 12;  
 int\* pnum = &num;  
 void\* ptr = pnum;

Ko želimo ta kazalec spet uporabiti,  
 ga moramo pretvoriti nazaj v pravi tip

int num2 = \*(int\*)(ptr); ← najprej pretvorba v int, nato deref.

## Reference:

Z **referenco** določimo drugo ime za isto spremenljivko

int stevilo = 21; → ref je drugo ime za stevilo  
int& ref = stevilo; ↗ znak &  
različni imeni za isto pomn. lokacijo z isto vrednostjo

Predstavitev v pomnilniku:  $\text{int} \& = \text{int}^* \text{ const}$

→ omejivje: naslova ne moremo spremenjati  
naslova ref. ni mogoče dobiti

→ compiler sam dereferencira kazalec, ko uporabimo referenco

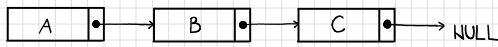
Uporaba: prenos po referenci, krajšanje dolgih imen

**Leva vrednost** (*"left value"*) - entiteta v pomnilniku, ki ima dostopen naslov  
→ spremenljivka na levi str. pireiditev

↳ na splošno v C++ je leva vrednost vse, s čimer lahko inicializiramo referenčno = leva vrednost

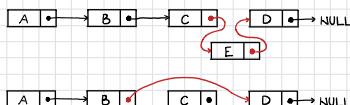
**Prenos kazalcev v funkcijo:** lahko po vrednosti ali referenci  
funkcija (`int*& ptr`) kazalu se pozna tudi ZUNAJ funkcije

#### 4. Dinamično enosmerno povezan seznam



Vsako vozlišče vsebuje podatek, shranjen v seznam in kazalec, ki kaže na naslednji kazalec

Dodajanje in brisanje vozlišč:



↳ Samp meniamo kazalce