Algoritmizácia a programovanie: 6. prednáška

Ján Grman

Obsah

- Reťazce ako špeciálne polia
- Ukazovatele – prvý náhľad

Ret’azce

Čo sú to reťazce

- reťazce sú jednorozmerné polia typu **`char`**
  - dĺžka reťazca: ľubovoľná, obmedzená veľkosťou pamäte
  - z celkovej pamäte je aktívna len časť od začiatku poľa do znaku **`'\0'`** $\rightarrow$ ukončovací znak
  - ak nie je reťazec ukončený znakom **`'\0'`**, považuje sa za reťazec celá pamäť až do najbližšieho znaku **`'\0'`**

Definícia a inicializácia reťazca

- Statický reťazec s najviac 6 znakmi:

```
char s[6] = "ahoj";
```

```
char s[] = "abrakadabra";
```

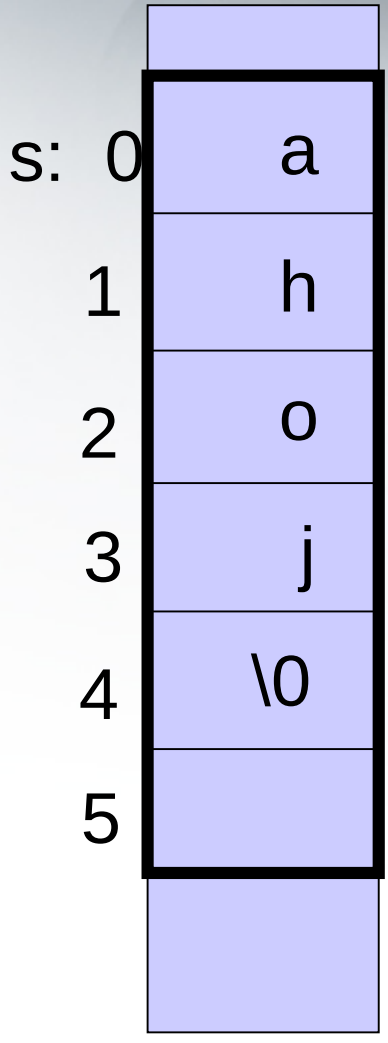
inicializuje sa miesta práve pre daný text

```
char s[15] = "abrakadabra";
```

pridá na koniec `'\0'`

```
char s[10];
s = "ahoj";
```

v C nie je možné takto priradiť statickému reťazcu konštantu



Poznámky k definícii a inicializácii reťazcov

- `"x"` a `'x'`:
  - `"x"` je reťazec s jedným znakom ukončený `'\0'` (2 Byty)
  - `'x'` je jeden znak (1 Byte)

Čítanie reťazca z klávesnice

```
char s[10];
...
scanf("%s", s);
```

sem nepartrí **&**, pretože **s** je adresa

- **scanf()** vynecháva biele znaky a číta po prvý biely znak
- ak je na vstupe **" ahoj Eva!"**, **scanf()** prečíta iba **"ahoj"** a zvyšok zostáva v bufferi klávesnice

Formátované čítanie: príklad

Program vypočíta celkovú sumu peňazí zo súboru, kde jednotlivé sumy sú vždy uvedené znakom **$** a znamienko **+** je pre príjem a **-** pre výdaj

Príklad súboru:

```
+  $10 -$5-   $8
     +$20
```

Formátované čítanie: príklad

```
include <stdio.h>

void main() {
    FILE *f;
    int kolko, suma = 0;
    char akcia[2];

    f = fopen("peniaze.txt",     );
    while (fscanf(f, "%1s", akcia) != EOF) {
        fscanf(f, "$%d", &kolko);
        suma +=  (akcia[0] == '+') ? kolko : (-1*kolko);
    }
    printf("Spolu: %d\n", su    );
    fclose(f);
}
```

namiesto **%1s** nemôže byť **%c**, pretože by prečítal medzeru, nie prvý znak

podobne **$%d**: **$** zabezpečuje, že sa preskočia biele znaky

Výpis reťazca na obrazovku: pomocou statického reťazca

```
printf("%s", s);
```

```
include <stdio.h>
void main()
{
    char str[11];

    printf("Zadaj retazec: ");
      scanf("%10s", str);
    printf("Retazec je: %s \n", str);
}
```

str má 10 znakov, preto je vhodné použiť:

vypísanie načítaného reťazca

Prístup k jednotlivým znakom reťazca

- reťazec = jednorozmerné pole znakov
- pracuje sa s ním ako s jednorozmerným poľom

po znakoch naplní reťazec hviezdičkami

```
char s[10];

for (i = 0; i < 10-1; i++)
    s[i] = '*';
s[10-1] = '\0';
```

dôležité: ukončiť reťazec!

Štandardné funkcie pre prácu s reťazcami

- nie sú súčasťou samotného jazyka C
- sú definované v **`<string.h>`**

Štandardné funkcie pre prácu s reťazcami

```
int strlen(char *s);
```

vracia dĺžku reťazca (bez **\0**)

```
char *strcpy(char *s1, char *s2);
```

kopírovanie reťazca **s2** do **s1**, vracia ukazovateľ na **s1**

```
char *strcat(char *s1, char *s2);
```

pripojí reťazec **s2** k **s1**, vracia ukazovateľ na **s1**

Štandardné funkcie pre prácu s reťazcami

```
char *strchr(char *s, char c);
```

nájdenie znaku **c** v reťazci **s**, vracia prvý výskyt znaku, ak sa v **s** nenachádza, vráti **NULL**

```
int strcmp(char *s1, char *s2);
```

vracia **0**, ak sú reťazce rovnaké, záporné číslo, ak **s1** je menšie, inak kladné číslo

```
char *strstr(char *s1, char *s2);
```

vracia ukazovateľ na prvý výskyt reťazca **s2** v reťazci **s1**, v prípade neúspechu **NULL**

Príklad: prekopírovanie reťazca znakov

```
#include <string.h>
...


char c, s1[10], s2[]="ahoj";
int i=0;

strcpy(s1, s2);
```

Príklad: porovnávanie reťazcov

Časť programu zistí, či sa reťazce rovnajú, prípadne, ktorý je neskôr v abecede.

```
char x[10], y[10];
int r;

if ((r = strcmp(x, y)) == 0)
   printf("Retazce \"%s\" a \"%s\" sa rovnaju\n", x, y);
else if (r < 0)
   printf("Retazec \"%s\" < retazec \"%s\"\n", x, y);
else
   printf("Retazec \"%s\" < retazec \"%s\"\n", y, x);
```

Príklad: spojenie reťazcov

Časť programu spojí názov súboru s príponou.

```
…
char subor[20], nazov[10], pripona[5];

scanf("%s", nazov);
scanf("%s", pripona);

strcpy(subor, nazov);
strcat(subor, ".");
strcat(subor, pripona);
…
```

Práca s obmedzenou časťou reťazca

- podobne ako uvedené funkcie,
- v názve je **n** (zo slova *number*), napr. **strncpy()**:

```
char *strncpy(char *s1, char *s2, int max);
```

kopírovanie najviac max znakov z reťazca **s2** do **s1**,
vracia ukazovateľ na **s1**

Práca s reťazcom naopak

- podobne ako uvedené funkcie,
- v názve je r (zo slova *reverse*), napr. **`strrchr()`**:

```
char *strchr(char *s, char c);
```

nájdenie znaku **`c`** v reťazci **`s`**, vracia posledný výskyt znaku, ak sa v **`s`** nenachádza, vráti **`NULL`**

Prevody reťazcov na čísla

- konvertovanie reťazca číslic na číslo (funkcie definované v **`stdlib.h`**)

```
int atoi(char *s);
```

prekonvertuje reťazec znakov na **`int`**

```
long atol(char *s);
```

prekonvertuje reťazec znakov na **`long`**

```
float atof(char *s);
```

prekonvertuje reťazec znakov na **`float`**

Pri vstupe a výstupe nie je konverzia potrebná (**`scanf()`** a **`printf()`**)

Príklad: prevody reťazcov na čísla

ak je prvý načítaný reťazec **"int"**,
druhý reťazec predstavuje číslo -
prekonvertuje sa na číslo

```
char s1[100], s2[100];
int i;

scanf("%s %s", s1, s2);
if (!strcmp(s1, "int")) {
   i = atoi(s2);
   printf("Nacitalo sa cele cislo: %d\n", i);
}
else {
   printf("Nacital sa retazec znakov: %s\n", s2);
}
```

Formátovaný vstup a výstup z a do reťazca

- použitie výhod formátovaného vstupu a výstupu, ale netlačiť nič na obrazovku ani nenačítavať

```
int sprintf(char *s, char *format, ...);
```

pracuje ako **`fprintf`**, ale zapisuje do reťazca **`s`**

```
int sscanf(char *s, char *format, ...);
```

pracuje ako **`fscanf`**, ale číta z reťazca **`s`**

Formátovaný vstup a výstup z a do reťazca: príklad 1

```
include <stdio.h>
void main() {
    int i;
    char s1[5], s2[10];

    printf("Zadaj 4 hexa cislice: ");

    scanf("%s", s1);
    sscanf(s1, "%x, &i);
    sprintf(s2, "%o", i);
    printf("%s \n", s2);
}
```

načíta 4 hexadec. číslice

číslice načíta do **i**

číslo **i** zapíše ako osmičkové do **s2**

Formátovaný vstup a výstup z a do reťazca: príklad 2

```
void uradnik( char *s ,
double r)
{
  double obsah, obvod;

  obsah = 3.14 * r * r;
  obvod = 2 * 3.14 * r;
  sprintf(s, "Kruh s
    polomerom %f ma
    obsah %f a obvod %f
    \n", r, obsah, obvod);
}
```

```
void sef(void)
{
  char sprava[100];

  uradnik(sprava, 5.0);
  printf("Sprava o
    kruhu je:\n
    %s", s);
}
```

Riadkovo orientovaný vstup a výstup z terminálu

- okrem **`scanf()`**, **`printf()`** aj:

```
char *gets(char *s);
```

číta celý riadok do **`s`**: na koniec nezapíše **`\n`**, ale **`\0`**, vracia ukazovateľ na **`s`**, ak je riadok prázdny dáva do **`s \0`** a vráti **`NULL`**

```
int puts(char *s);
```

vypíše reťazec a odriadkuje (**`\n`**), vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Riadkovo orientovaný vstup a výstup zo súboru

- okrem **`fscanf()`**, **`fprintf()`** aj:

```
char *fgets(char *s, int max, FILE *fr);
```

číta riadok zo súboru do konca riadku ale maximálne **`max`** znakov, načítané zapíše do **`s`** (aj s **`\n`**), vracia ukazovateľ na **`s`**, ak je koniec súboru tak **`NULL`**

```
int fputs(char *s, FILE *fw);
```

do súboru **`fw`** vypíše reťazec **`s`**, neodriadkuje ani neukončuje pomocou **`\0`**, vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Príklad: riadkovo orientovaný vstup a výstup

Program načíta riadok zo súboru a riadok zo štandardného vstupu. Načítané reťazce vypíše na obrazovku - každý do zvlášť riadku. Reťazec znakov vypíše do súboru a na obrazovku a odriadkuje.

```c
#include <stdio.h>
#include <string.h>

int main()
{
   FILE *f;
   char s[100];

   /* citanie */
   if((f = fopen("subor.txt", "r")) == NULL) {
      printf("Subor sa nepodarilo otvorit na citanie.\n");
      return 1;
   }
   printf("%s", fgets(s, 100, f));
   printf("%s\n", gets(s));         /* pridanie \n */

   fclose(f);
```

Príklad: riadkovo orientovaný vstup a výstup

```
    /* zapis */
    if((f = fopen("subor_zapis.txt", "w")) == NULL) {
       printf("Subor sa nepodarilo otvorit na zapis.\n");
       return 1;
    }
    fputs(s, f);
    fputs("\n", f);  /* pridanie \n */
    puts(s);

    fclose(f);

    return 0;
}
```

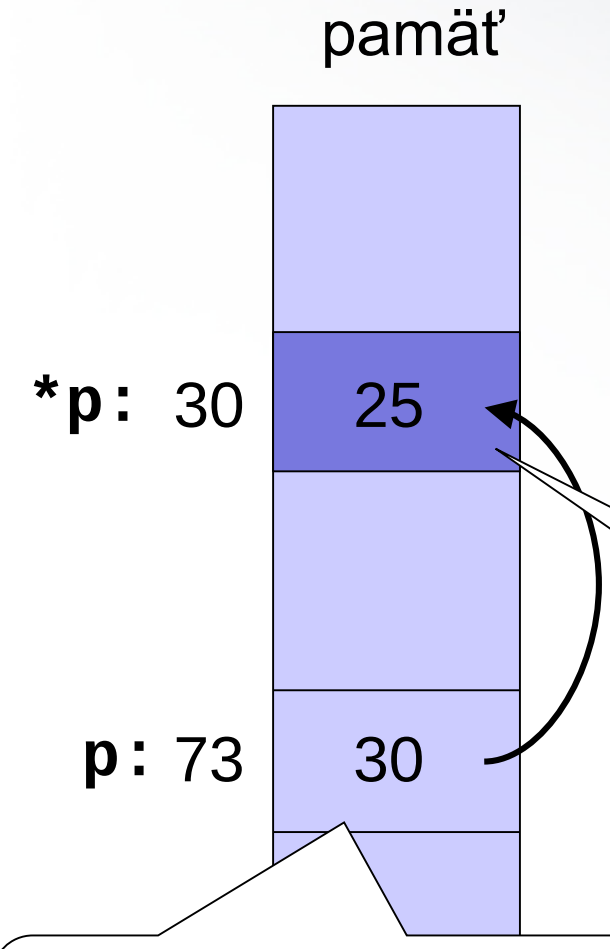
Ukazovatele – prvý náhľad

Čo sú to ukazovatele

= pointery, smerníky

- ukazovateľ
  - je premenná
  - jeho hodnota je adresa v pamäti
- analógia: v texte článku nie je informácia priamo uvedená, ale je tam *odkaz* na nejaký iný článok, kde sa informácia nachádza
- Na čo sú dobré?
  - Keď je až za behu programu jasné, koľko pamäte budeme potrebovať (napr. ako veľké pole)

Príklad ukazovateľa



- ukazovateľ **p**:
  - zapísaný na adrese 73
  - jeho hodnota je 30 a vyjadruje adresu, kde je uložená skutočná hodnota
  - na adrese 30 v pamäti je hodnota 25, ktorá sa použije napr. pri výpočtoch
- **p**: ukazovateľ
- ***p**: hodnota, kam ukazuje

Pamäťové miesto vznikne vytvorením premennej typu ukazovateľ (**p**)

Pamäťové miesto (alebo viac miest – pole) je potrebné v vyhradiť (alokovať)

Ako poznáme ukazovateľ

- ukazovateľ je definovaný pomocou *****
- **`int i`** - „klasická“ celočíselná premenná
- **`int *p_i`** - ukazovateľ na celočíselnú premennú

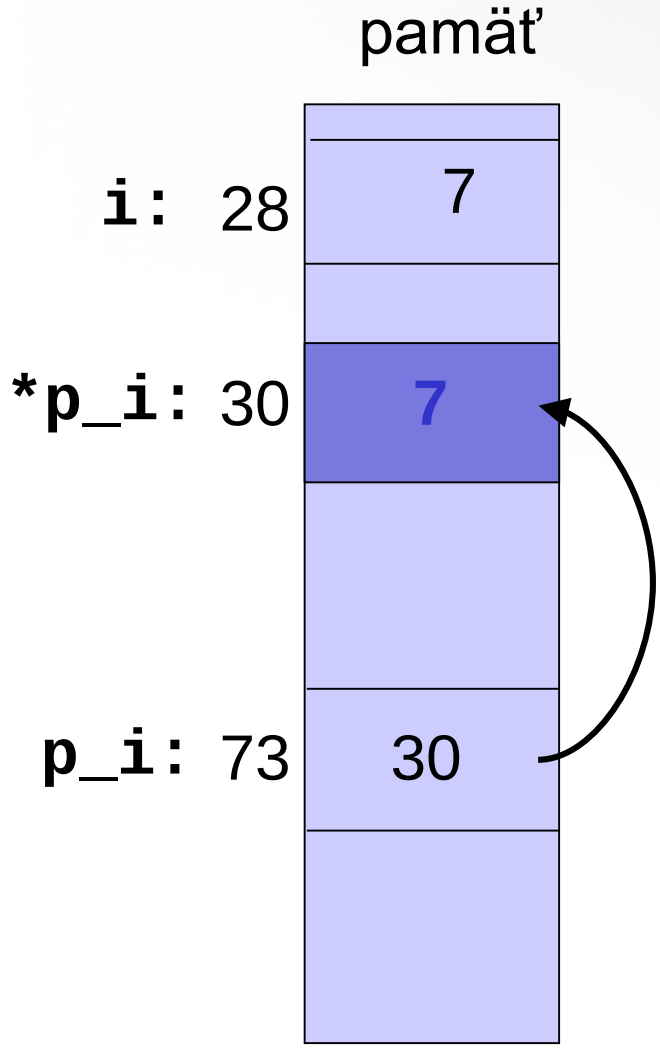
- definícia ukazovateľa:

```
int i;
int *p_i;
```

je ekvivalentné

```
int i,
    *p_i;
```

Čo urobí `*p_i = i;`



- ukazovateľ p_i
  - p_i == 30
  - *p_i == **7**

```
int i, *p_i;
...
*p_i = i;
```

- obsah pamäte, ktorú ukazuje **p_i** sa prepíš notou

treba inicializovať **p_i**,
napr. vyhradiť (alokovať) pamäť
pre **p_i**

niesto v

Ako získame adresu premennej

- pomocou referenčného operátora **&**
- **`int i`** - „klasická“ celočíselná premenná
- **`&i`** – adresa premennej

- definícia ukazovateľa:

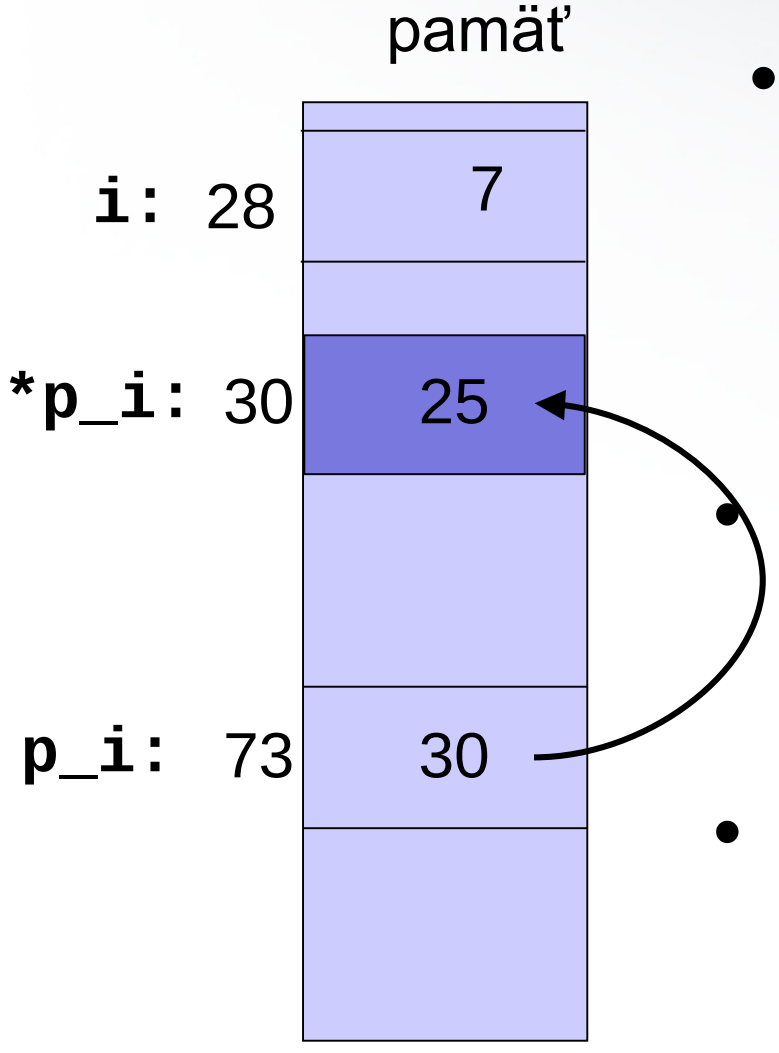
definícia ukazovateľa a súčasne inicializácia

```
int i, *p_i;
p_i = &i;
```

je ekvivalentné

```
int i, *p_i = &i;
```

Čo urobí `p_i = &i;`



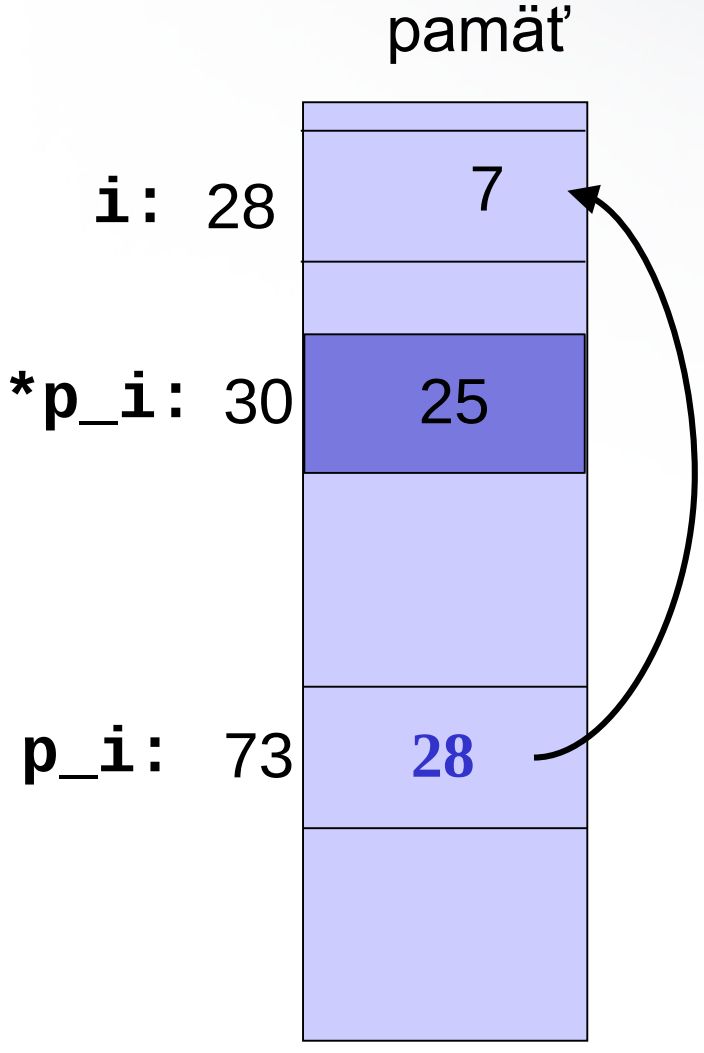
- ukazovateľ p_i
  - p_i == 30
  - *p_i == 25

```
int i, *p_i;

p_i = &i;
```

- hodnota **`p_i`** (adresa, kam **`p_i`** ukazuje) sa prepíše adresou premennej **`i`**
- hodnota **`*p_i`** je tá istá ako hodnota **`i`**

Čo urobí `p_i = &i;`



- ukazovateľ p_i
  - p_i == **28**
  - *p_i == **7**

```
int i, *p_i;

p_i = &i;
```

- hodnota **`p_i`** (adresa, kam **`p_i`** ukazuje) sa prepíše adresou premennej **`i`**
- hodnota **`*p_i`** je tá istá ako hodnota **`i`**

Príklady

`p_i = &i;`	- správne
`p_i = &(i + 3);`	- chyba: **`(i + 3)`** nie je premenná
`p_i = &15;`	- chyba: konštanta nemá adresu
`p_i = 15;`	- chyba: priraďovanie absolútnej adresy
`i = p_i;`	- chyba: priraďovanie adresy
`i = & p_i;`	- chyba: priraďovanie adresy
`*p_i = 4;`	- správne, ak **`p_i`** bol inicializovaný

Keď ukazovateľ neukazuje nikam

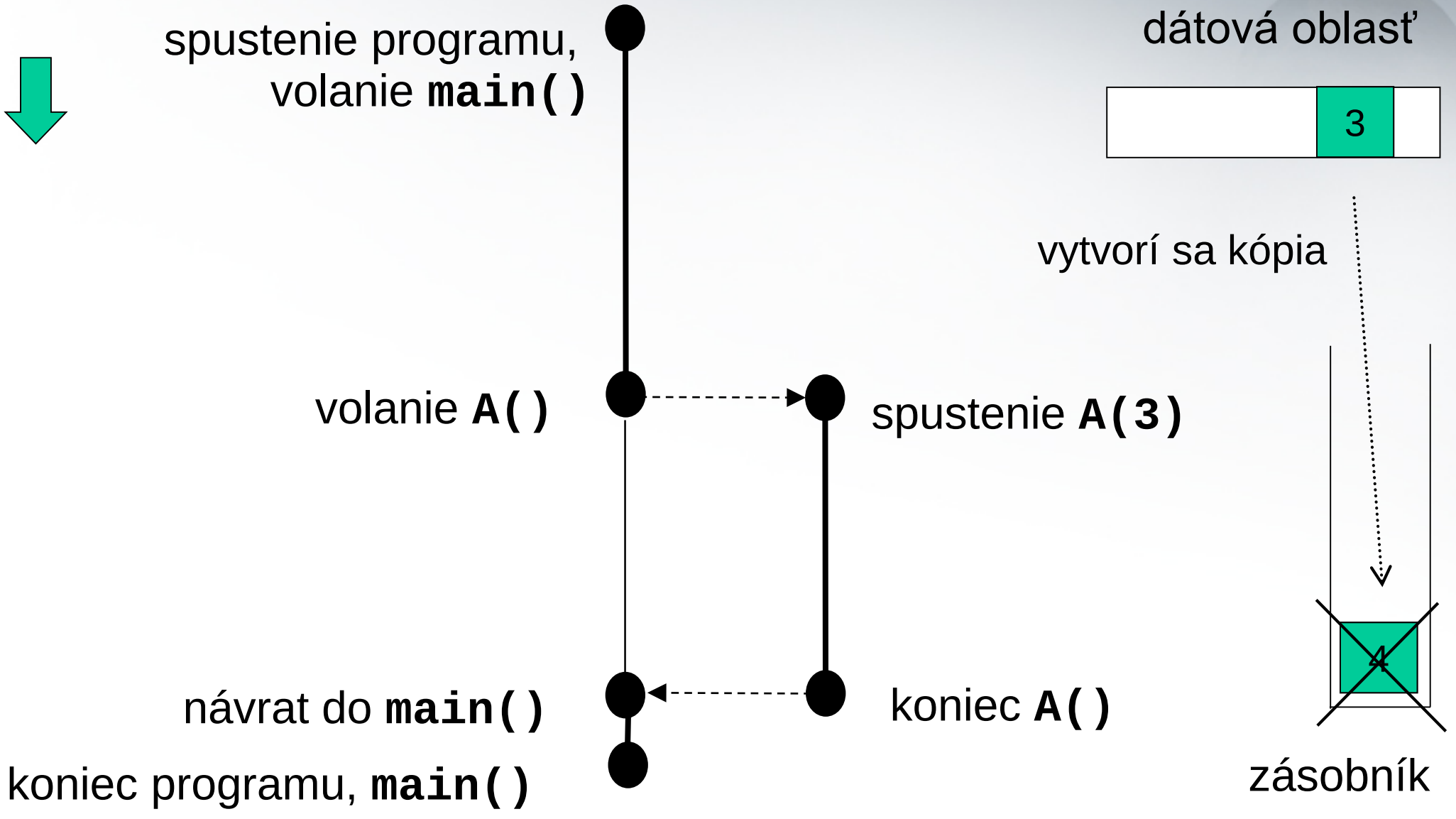
- Nulový ukazovateľ: **`NULL`**
- **`NULL`** - symbolická konštanta definovaná v **`stdio.h`**:
  - **`#define NULL 0`**
  - **`#define NULL ((void *) 0)`**
- Je možné priradiť ho ukazovateľom na ľubovoľný typ

```
if (p_i == NULL)
    ...
```

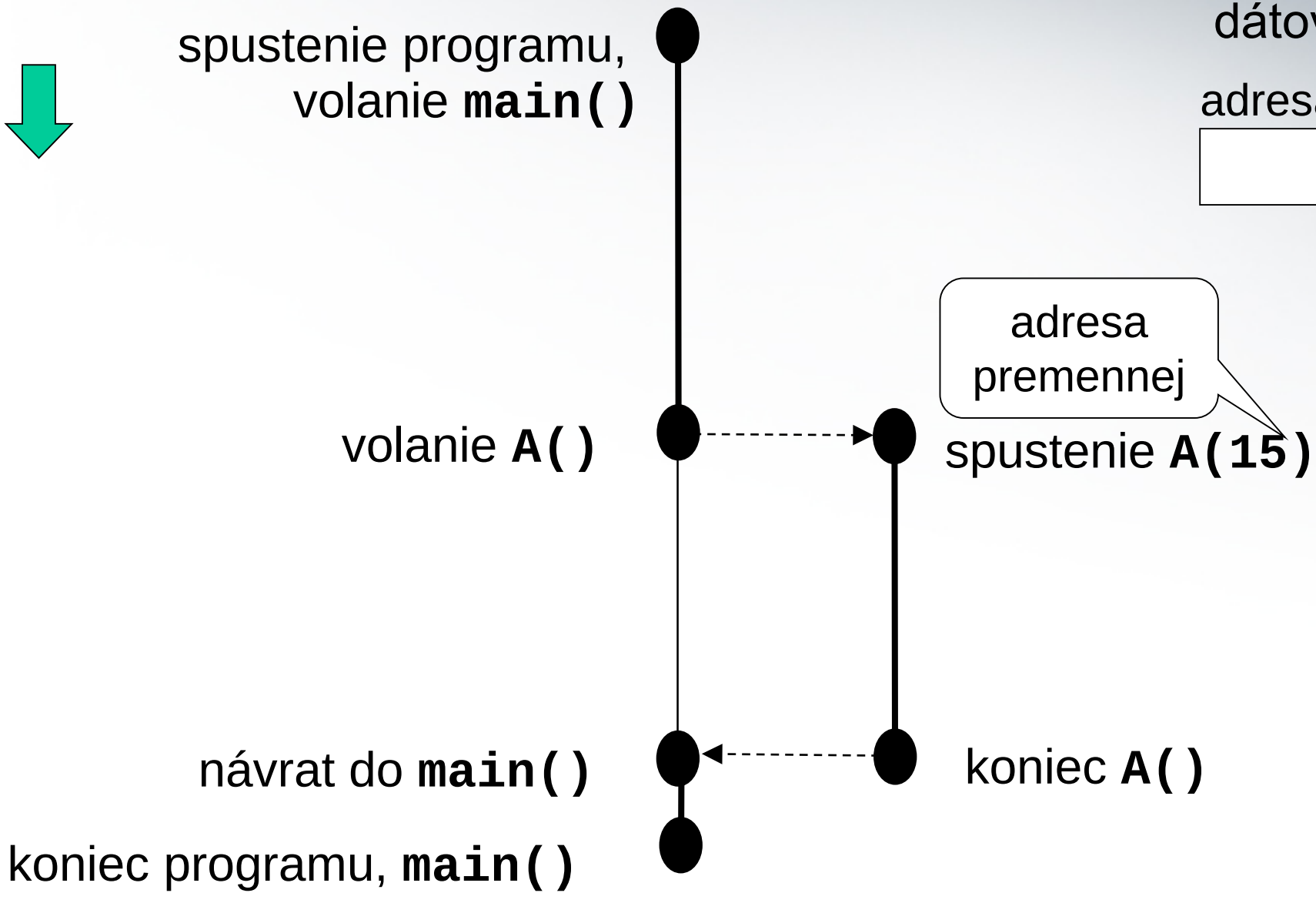
Opakovanie: volanie odkazom

- V C nie je volanie odkazom, funkcie sa volajú len hodnotou
- Vo funkcii vzniká kópia argumentu funkcie (lokálna premenná), ktorá zaniká s ukončením funkcie
- Preto sa funkcia nevolá s premennou, ktorú chceme meniť, ale s jej adresou

Parametre funkcií - volanie hodnotou: `int A(int i)`



Parametre funkcií - volanie odkazom: `int A(int *i)`



```c
#include <stdio.h>

#define PI 3.14
#define na_druhu(i) ((i) * (i))

void kruh(int r, float *o, float *s)
{
   *o = 2 * PI * r;
   *s = PI * na_druhu(r);
}

int main()
{
   int polomer;
   float obvod, obsah;

   printf("Zadaj polomer kruhu: ");
   scanf("%d", &polomer);

   kruh(polomer, &obvod, &obsah);
   printf("obvod: %.2f, obsah: %.2f\n", obvod, obsah);
   return 0;
}
```

Funkcia vypočíta obvod a obsah kruhu vo funkcii **`kruh()`**. Volanie odkazom.

Príklad funkcie: výmena premenných

```
void vymen(int *p_x, int *p_y)
{
    int pom;

    pom = *p_x;
    *p_x = *p_y;
    *p_y = pom;
}
```

i: 7

j: 5

- volanie funkcie: **`vymen(&i, &j)`**

Príklad funkcie: výmena premenných

- volanie funkcie: `vymen(&i, &j)`

```
vymen(i, j);
```

chyba: vymieňa obsah adries, daných obsahom **i**, **j**: vymieňa hodnoty na adresách 5 a 7

```
vymen(*i, *j);
```

chyba: vymieňa adresy adries z obsahu **i**, **j**: z adries 5 a 7 sa zoberú hodnoty a tie sa použijú ako adresy

Prideľovanie pamäte

- pomocou funkcie definovanej v **`stdlib.h`** (niekedy v **`alloc.h`**):

počet Bytov

```
void *malloc(unsigned int)
```

Adresa prvého prideleného prvku - je vhodné pretypovať. Ak nie je v pamäti dosť miesta, vráti NULL.

Testovanie pridelenia pamäte

- kontrola, či **`malloc()`** pridelil pamäť:

```
int n, *p_i;
printf("Zadajte velkost pola.\n");
scanf("%d", &n);

if((p_i = (int *) malloc(n*sizof(int))) == NULL) {
    printf("Nepodarilo sa pridelit pamat\n");
    exit;
}
```

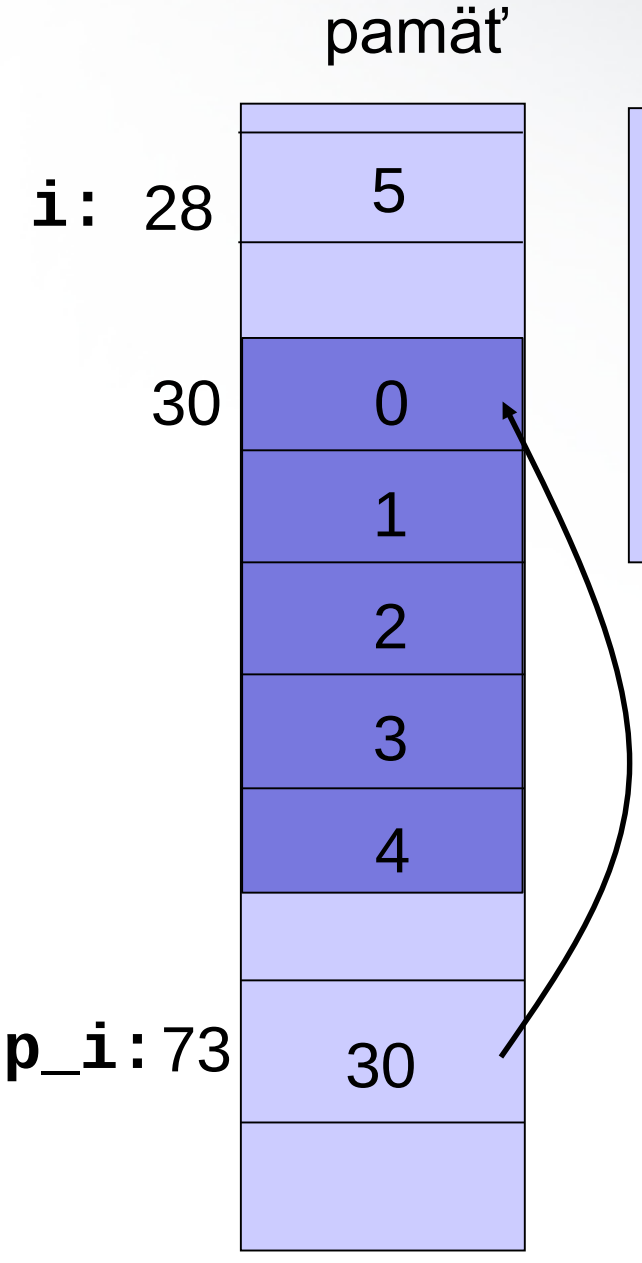
Dynamické polia

- Pomocou **`malloc()`** je možné prideliť blok pamäte – dynamické pole
- Prístup k prvkom dynamického poľa:
  - Ukazovateľov (dozviete sa na budúci semester)
  - Indexov – rovnako ako u statického poľa

```
int n, *p_i;

scanf("%d", &n);
if((p_i = (int *) malloc(n*sizeof(int))) == NULL) {
    printf("Nepodarilo sa pridelit pamat\n");
    exit;
}
for(i=0; i<n; i++)
   p_i[i] = i+1;
```

Dynamické polia




```
int i=5, *p_i;

p_i = (int *) malloc(n*sizof(int)));
for(i=0; i<n; i++)
   p_i[i] = i;
```

- hodnota **p_i** (adresa, kam **p_i** ukazuje) sa prepíše adresou premennej **i**
- hodnota ***p_i** je tá istá ako hodnota **i**

Uvoľňovanie pamäte

- nepotrebnú pamäť je vhodné ihneď vrátiť operačnému systému
- pomocou funkcie: `void free(void *)`

príklad:

```
char *p_c;

p_c = (char *) malloc(1000 * sizeof(char));
...
free(p_c);
p_c = NULL;
```

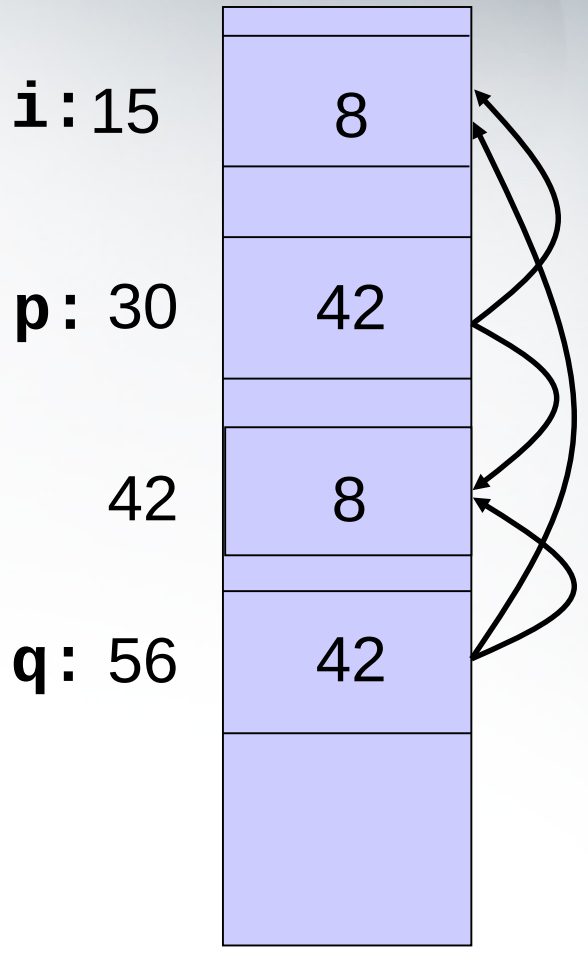
Dynamické prideľovanie a uvoľňovanie pamäte

- prideľovanie pamäte za chodu programu
  - v zásobníku (stack) - riadi operačný systém
  - **v hromade (heap) - riadi programátor**
- životnosť dynamických dát:
  - od alokovania po uvoľnenie pamäte

Ukazovatele príklad

```
int i, *p, *q;

i = 5;
p = &i;
*p = 6;
q = &i;
*q = 7;
p = (int *) malloc(sizeof(int));
*p = 8;
*q = *p;
 q = p;
```



Ukazovatele – čo treba vedieť

- Využitie ukazovateľov na vrátenie hodnoty z funkcie prostredníctvom argumentov
- Porozumenie funkciám na prácu s reťazcami (ktoré budú nasledovať)