Procedurálne programovanie

slido.com
2558824
PrPr – P4

Ján Zelenka

Ústav Informatiky

Slovenská akadémia vied

Obsah prednášky

slido slido.com
2558824
PrPr – P4

1. **Opakovanie**
2. **Ukazovateľ**
3. **Alokácia pamäte – statická alokácia**

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*

Opakovanie

slido.com
2558824
PrPr – P4

Funkcie

Argumenty: lokálne premenné, predávajú si hodnotou

Návratová hodnota: výstup funkcie (za kľúčovým slovom ***return***)

```
návratový_typ meno_funkcie(argumenty)
{
    telo_funkcie;
    return(hodnota_typu_zhodného_s_návratový_typ);
}
```

Kľúčové slovo ***void***

žiadne argumenty

```
int f(void);
```

bez návratovej hodnoty (procedúra)

```
void f(int a, int b);
```

Funkcie

slido slido.com
2558824
PrPr – P4

Prenesenie hodnoty mimo funkcie:

- návratová hodnota
- pomocou smerníkov (neskôr)
- globálna premenná (nepoužívať)

Vracanie viacerých hodnôt:

- globálne premenné
- štruktúrovaný typ (struct, smerník)
- modifikácia vstupných parametrov (smerník)

Prehľad základných funkcií na prácu so súbormi

slido slido.com
2558824
PrPr – P4

- `fopen()` - **otvorenie súboru** v rôznych režimoch (čítanie, zápis, textový binárny...)
- `fclose()` - **zatvorenie súboru** otvoreného fopen()
- `fseek()` - **nastavenie** ukazovateľa **na pozíciu** čítania, zápisu v otvorenom súbore
- `ftell()` - **zistenie pozície** ukazovateľa čítania, zápisu v otvorenom súbore

Prehľad základných funkcií na prácu so súbormi

slido.com
2558824
PrPr – P4

Textový súbor:

- `getc()` - čítanie **znaku** z otvoreného súboru
- `putc()` - zápis **znaku** do otvoreného súboru
- `fgets()` - čítanie **celého riadku** z otvoreného súboru
- `fputs()` - zápis **ret'azca** do otvoreného súboru
- `fprintf()` - analógia printf() a sprintf(), ale s výstupom do súboru -> pracujeme s **číslami, slovami...**

Binárny súbor:

- `fread()` - čítanie **bloku** z otvoreného súboru
- `fwrite()` - zápis **bloku** do otvoreného súboru

Ukazovatele

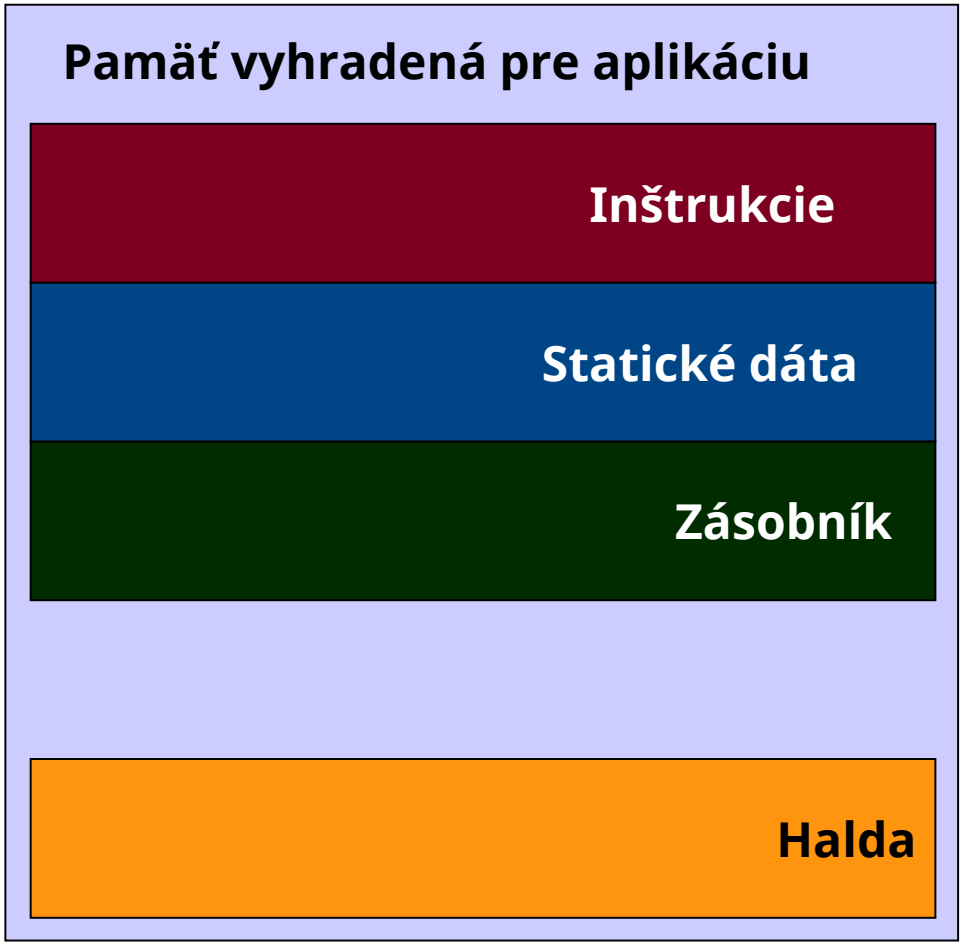
slido.com
2558824
PrPr – P4

Pamäť

- pamäť obsahuje veľké množstvo buniek s pevnou veľkosťou
  - adresované zvyčajne na úrovni 8 bitov (1 bajt)
- v pamäti sú umiestnené entity (premenné, reťazce…)
- entita zaberá viacej ako jednu bunku
  - napr. premenná typu int zaberá 4 bajty
  - adresa entity v pamäti je prvá bunka kde je entita umiestnená
- adresy sú uvádzané v hexadecimálnej sústave (s predponou 0x)

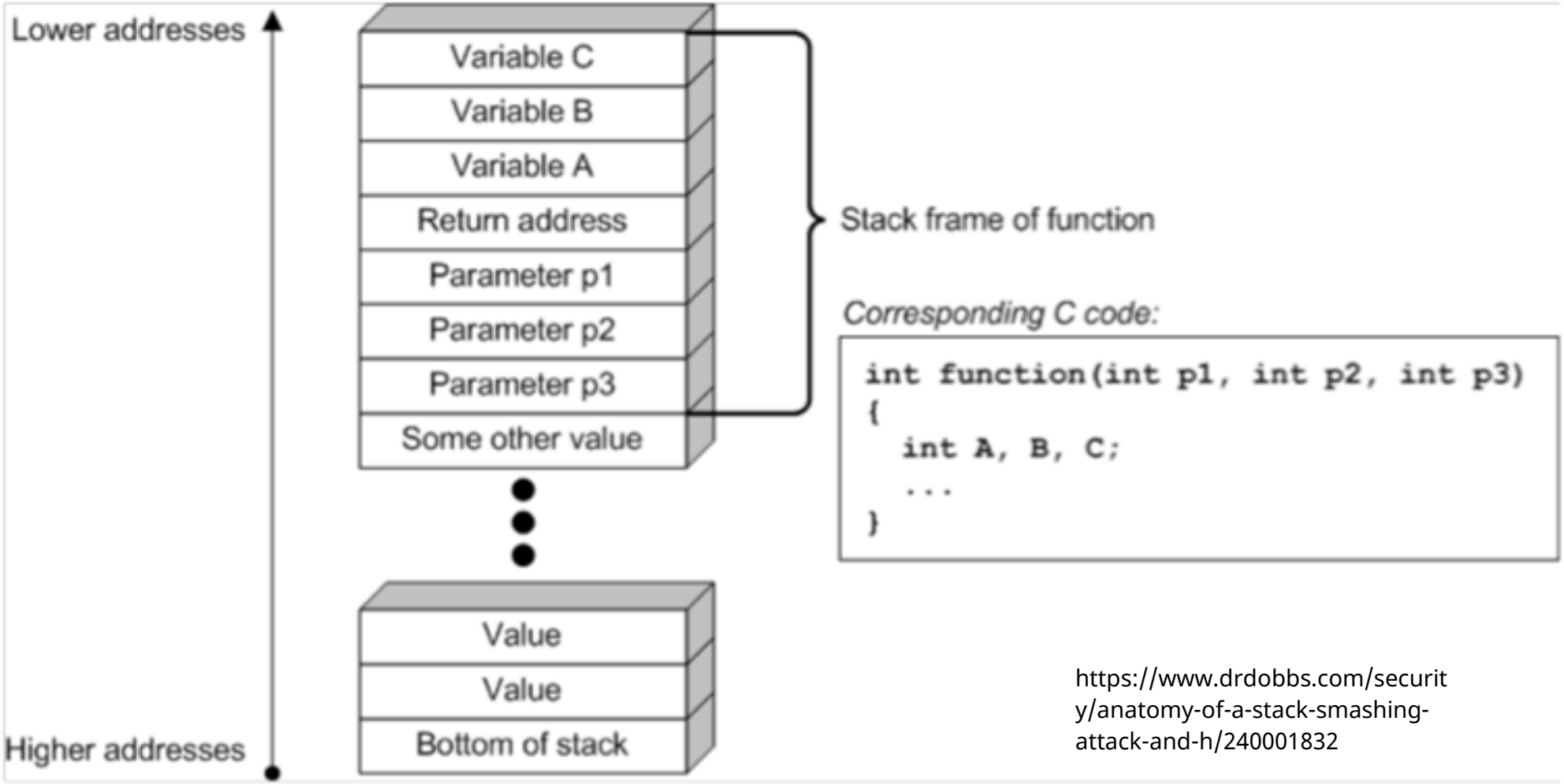
Organizácia pamäte

- Inštrukcie
  - Asemblerovský kód aplikácie
- Statické dáta (static)
  - Globálne premenné
- Zásobník (stack)
  - Volania funkcií
    (iné pre každú funkciu)
  - Lokálne premenné
- Halda, hromada (heap)
  - Dynamicky alokované premenné
    (malloc, free)
  - Iné pri každom spustení



Zásobník

slido slido.com
2558824
PrPr – P4



Corresponding C code:

```
int function(int p1, int p2, int p3)
{
  int A, B, C;
  ...
}
```

https://www.drdobbs.com/security/anatomy-of-a-stack-smashing-attack-and-h/240001832

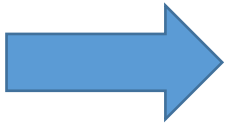
Adresa premennej
Referenčný operátor &

slido slido.com
2558824
PrPr – P4

- Operátor `&` vracia adresu svojho argumentu
  - miesto v pamäti, kde je argument uložený
- Adresa (jeho výstup) sa priraďuje do ukazovateľa

```
int i = -10;
int *p_i = &i;
```

`int i = -10;`
(i je meno miesta v pamäti s adresou X, kde je uložená hodnota -10)



Obsah pamäťovej bunky	**Adresa** pamäťovej bunky
-10	X

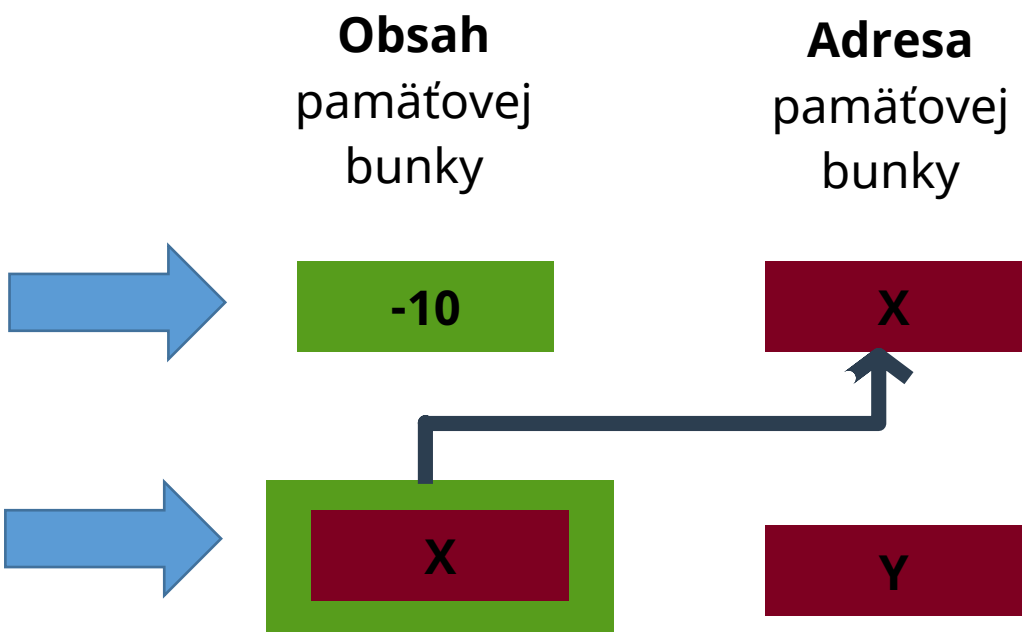
Adresa premennej
Referenčný operátor &

slido slido.com
2558824
PrPr – P4

- Operátor `&` vracia adresu svojho argumentu
  - miesto v pamäti, kde je argument uložený
- Adresa (jeho výstup) sa priraďuje do ukazovateľa

```
int i = -10;
int *p_i = &i;
```

`int i = -10;`
(i je meno miesta v pamäti s adresou X, kde je uložená hodnota -10)
`int *p_i = &i;`
(p_i je meno miesta v pamäti s adresou Y, kde je uložená adresa X)



Premenná VS ukazovateľ

slido slido.com
2558824
PrPr – P4

Štandardná premenná

- pomenovanie pamäťové miesto, ktoré má svoju adresu X

Ukazovateľ

- všetky ukazovatele majú rovnakú veľkosť (sú to pamäťové miesta na uchovanie adresy - dátový typ ukazovateľa je adresa)
- jeho typ musí zodpovedať typu premennej, na ktorú ukazuje
  - ako sa majú bity na adrese Y interpretovať?
- ak ukazovateľ ukazuje na určitú adresu, vieme že na tejto adrese začína a ďalej spojito pokračuje cez toľko bajtov, koľko je potrebných na uloženie premennej príslušného typu.

Definícia ukazovateľa

- špecifikujeme typ ukazovateľa
- pred menom premennej je *

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

je ekvivalentné

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

iba premenná **a** je ukazovateľ

```
int *a, b;
```

- Pozor na neinicializovaný ukazovateľ
  - Ukazuje na náhodné miesto v pamäti
  - Pred prvým použitím je potrebná jeho inicializácia

Inicializácia ukazovateľa

slido.com
2558824
PrPr – P4

Priradíme premennej p adresu premennej i.

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

Ukazovateľ p_i ukazuje na premennú i.

Ukazovateľ, ktorý nikam neukazuje

- Nulový ukazovateľ: **`NULL`** neobsahuje žiadnu platnú adresu – neukazuje na žiadne konkrétne miesto v pamäti, s ktorým chceme pracovať.
- **`NULL`** - symbolická konštanta definovaná v **`stdio.h`**:
  - `#define NULL 0`
  - `#define NULL ((void *) 0)`
- Je možné priradiť ho ukazovateľom na ľubovoľný typ

```
int *p;
p = NULL;
```

```
if (p == NULL)
...
```

Údaje na adrese
Dereferenčný operátor *

slido slido.com
2558824
PrPr – P4

- Umožňuje pracovať s obsahom pamäťového miesta na ktoré ukazuje ukazovateľ
  - Pozor nie vlastnú hodnotu ukazovateľa – to je adresa
  - Ukazovateľ môže ukazovať na hodnotu, ktorá je interpretovaná ako adresa -> **`int**`**

```
int i = 7;
int *p = &i;
*p = 10;
```

- obsah pamäte, na ktorú ukazuje **`p`** sa zmení na 10
- **`p`** ukazuje na to isté miesto v pamäti

```
p = 10;
```

- **CHYBA**: meníme miesto, na ktoré ukazuje **`p`**

Ukazovateľ na ukazovateľ

```
int a, *p_a, **p_p;

p_a = &a;
p_p = &p_a;

int b = *p_a;
int c = **p_p;

 *p_p = p_a;
**p_p = 10;
 *p_p = &a;
```

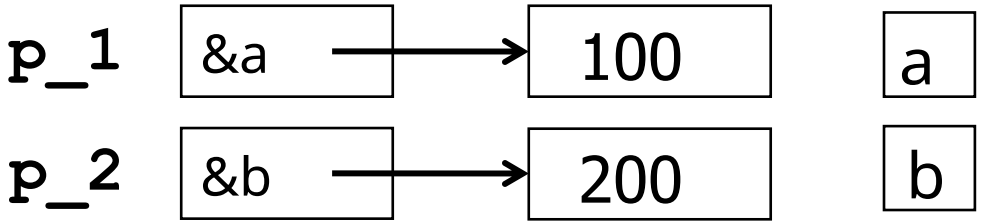
a == *p_a == **p_p
&a == p_a == *p_p
&p_a == p_p

Základné operácie s ukazovateľmi

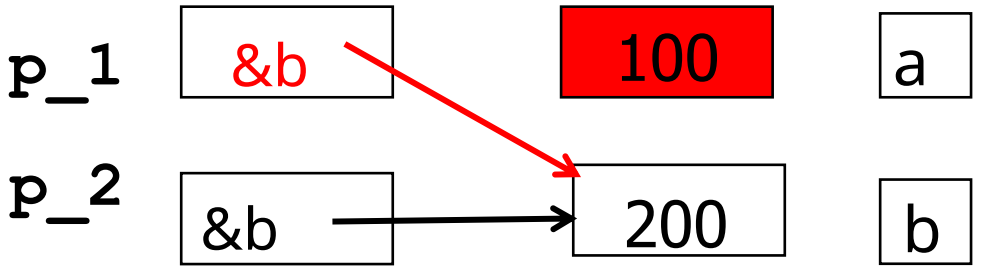
```
int *p_1, *p_2;
...
p_1 = p_2;
```

```
int a = 100, b = 200;
p_1 = &a;
p_2 = &b;
```

pred príkazom `p_1 = p_2;`



po príkaze `p_1 = p_2;`



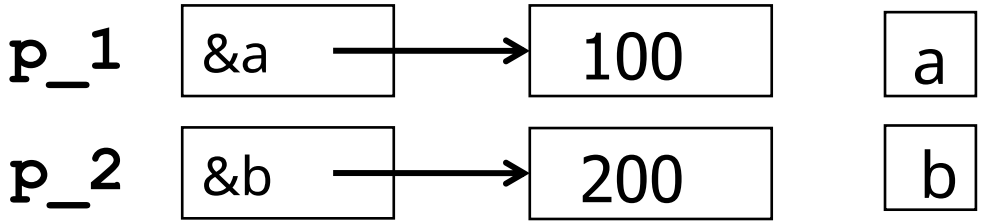
p_1 ukazuje na rovnakú premennú ako p_2

Základné operácie s ukazovateľmi

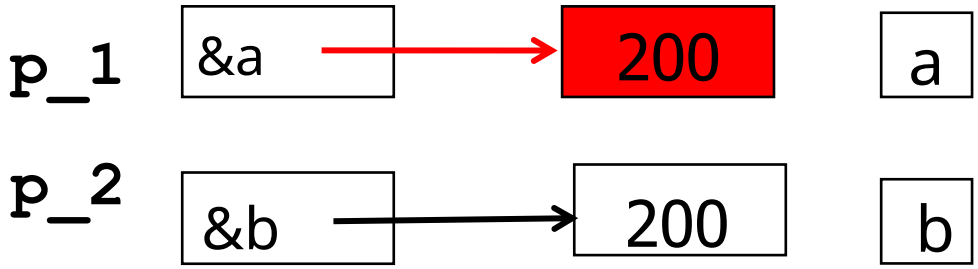
```
int *p_1, *p_2;
...
*p_1 = *p_2;
```

```
int a = 100, b = 200;
p_1 = &a;
p_2 = &b;
```

pred príkazom `*p_1 = *p_2;`



po príkaze `*p_1 = *p_2;`



Nemení sa hodnota ukazovateľa,
ale hodnota premennej na ktorú `p_1` ukazuje.

Základné operácie s ukazovateľmi

slido slido.com
2558824
PrPr – P4

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

```
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

Výpis adresy

Špecifikácia formátu (v **`printf()`**): %p

```
int i, *p_i;

p_i = &i;
printf("Adresa i: %p, hodnota p_i:
%p\n", &i, p_i);
```

Segmentation fault

slido slido.com
2558824
PrPr – P4

Majme ukazovateľ na celočíselnú premennú

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

Priraďme mu adresu, ktoré nie je prístupná programu

```
p_i = 10;
```

Pri pokuse o prístup mimo vyhradenú pamäť

```
*p_i = 7;
```

Segmentation fault

Viacej ukazovateľov na rovnaké miesto v pamäti

```
int i = 1;
int *p_1 = &i;
int *p_2 = &i;
int *p_3 = &i;
```

Všetky ukazovatele môžu meniť dané miesto v pamäti

```
*p_3 = 5;
*p_1 = 62;
*p_2 = 39;
```

Pozor na vzájomné prepisovanie si hodnôt

Konverzia ukazovateľov

- Vyhnúť sa jej!
- Ak sa nedá vyhnúť – explicitne pretypovávať

```
int *p_i;
char *p_c;

p_c = p_i;
p_c = (char *)p_i;
```

nevhodné

vhodnejšie

Ukazovateľ typu `void`

slido slido.com
2558824
PrPr – P4

- V čase definície ukazovateľa nie je zrejmé na aký typ premennej bude ukazovať.
- Použijeme generický ukazovateľ

- Môže ukazovať na ľubovoľný typ

```
void *p;
```

- Pred konkrétnym použitím generického ukazovateľa je potrebné pretypovanie na typ konkrétnej premennej, na ktorú bude ukazovať.

Ukazovateľ typu `void`

slido slido.com
2558824
PrPr – P4

Pri priraďovaní je potrebné uviesť typ

```
int i;
float f;
void *p_void = &i;

*(int *) p_void = 2;
p_void = &f;
*(float *) p_void = 3.5;
```

`p_void` ukazuje na `i`

nastavenie `i` na 2

`p_void` ukazuje na `f`

nastavenie `f` na 3.5

Parametre funkcií
Odovzdanie hodnotou

slido slido.com
2558824
PrPr – P4

Odovzdávanie hodnotou

```
void foo(int X){
    X = 3;
    // v X je 3
}
int main(){
    int variable = 0;
    foo(variable);
    //v X je znova 0
    return 0;
}
```

- Hodnota sa nakopíruje do novej lokálnej premennej
- Po zániku lokálnej premennej sa zmena nepropaguje mimo tela funkcie

Parameter je ukazovateľ

```
void foo(int *P){
    *P = 3;
}
int main(){
    int X = 0;
    foo(&X);
    // v X je 3
    return 0;
}
```

- Riešením je predávať (ako hodnotu) adresu premennej
- Modifikuje sa hodnota na tejto adrese, namiesto lokálnej premennej
- Takto sa odovzdávajú premenné, ktoré sa majú meniť

Parametre funkcií
Odovzdanie odkazom

slido slido.com
2558824
PrPr – P4

- predávanie parametrov odkazom v čistom C neexistuje (podporuje ho C++)
  - volanie odkazom by umožnilo meniť parametre v rámci funkcie tak, aby zostali aj po jej skončení
  - rieši sa pomocou ukazovateľov
  - ukazovateľ určuje, na ktorom mieste vo volajúcej funkcii sa má premenná zmeniť (nemení sa ukazovateľ - adresa)

Ukazovateľ na funkciu

slido slido.com
2558824
PrPr – P4

- Udalosťami riadené programovanie (Event-driven)
  - napr. zaregistrovanie funkcie ako (callback) reakcie pri nejakej udalosti (napr. antivírové programy)
- Ukazovateľ na funkciu obsahuje adresu umiestnenia kódu funkcie
  - Namiesto hodnoty je na adrese kód funkcie
- Ukazovateľ na funkciu môže byť parameter inej funkcie
  - Predáme miesto, kde je zápis toho, čo sa má vykonať

```
void foo(int i,int(*p_f)(float, float));
```

Ukazovateľ na funkciu

slido slido.com
2558824
PrPr – P4

```
#include <stdio.h>
void vypis(int a)
{
    printf("Zadana hodnota je %d\n", a);
}

int main()
{
    void (*p_vypis)(int) = &vypis;

    (*p_vypis)(10);

    return 0;
}
```

```
void (*p_vypis)(int);
p_vypis = &vypis;
```

`p_vypis` je ukazovateľ na funkciu vypis()

Zavolanie funkcie pomocou ukazovateľa

- Meno funkcie sa dá použiť na získanie jej adresy
- operátor & nie je povinný
- vždy bez zátvoriek

- Ukazovateľ na funkciu sa dá zavolať ako funkcia
- Operátor * nie je povinný

Ukazovateľ na funkciu

Dôležitá je signatúra funkcie

- Typ a počet argumentov, typ návratovej hodnoty
- Meno funkcie nie je dôležité
- Do ukazovateľa na funkciu s danou signatúrou môžeme priradiť adresy všetkých funkcií s rovnakou signatúrou

```
int Scitaj (int a, int b){…}
int Vynasob (int a, int b){…}
int main (void) {
    int (*p_f) (int, int) = &Scitaj;
    p_f(1,3);
    p_f = &Vynasob;
    p_f(7,2);
    return 0;
}
```

Príklady definícií

slido slido.com
2558824
PrPr – P4

```
int i;
```
- **i** je typu **int**

```
float *y;
```
- **y** je ukazovateľ na typ **float**

```
double *z();
```
- **z** je funkcia vracajúca ukazovateľ na **double**

```
int (*v)();
```
- ukazovateľ na funkciu vracajúcu **int**

```
int *(*v)();
```
- ukazovateľ na funkciu vracajúcu ukazovateľ na **int**

Príklad - v1

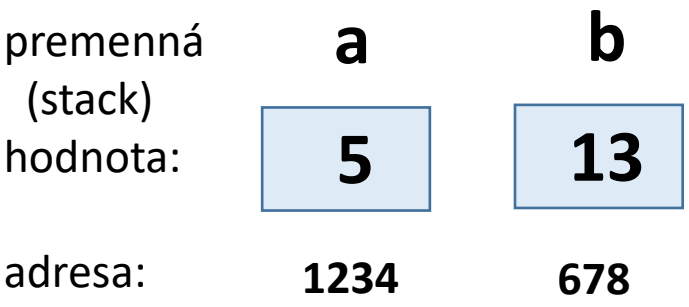
slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

Príklad - v1

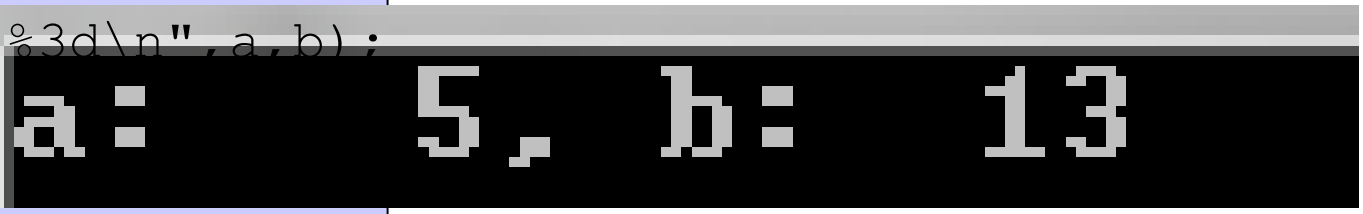
slido slido.com
2558824
PrPr – P4

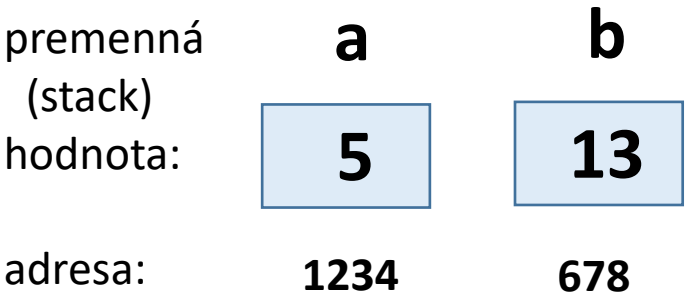
```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```



Príklad - v1

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

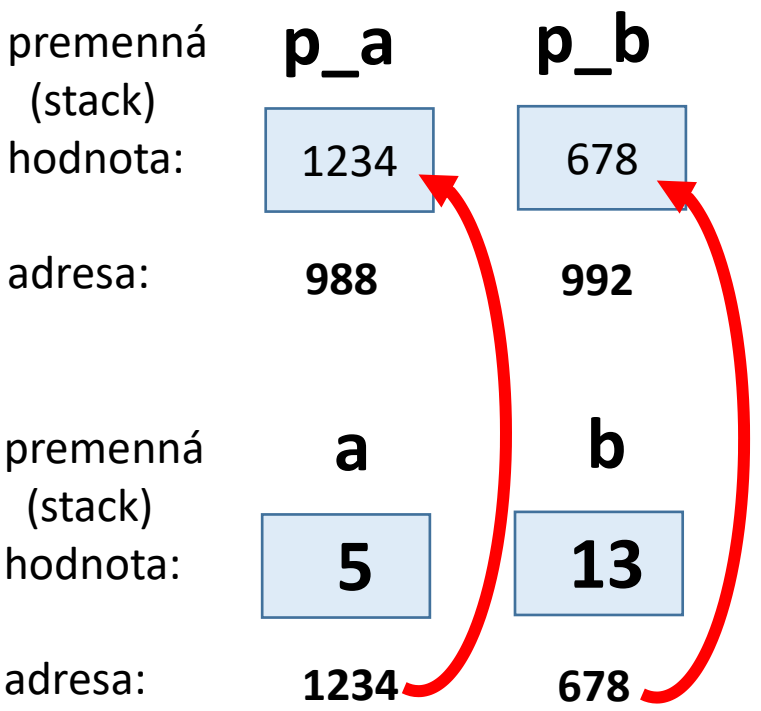




Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```



Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

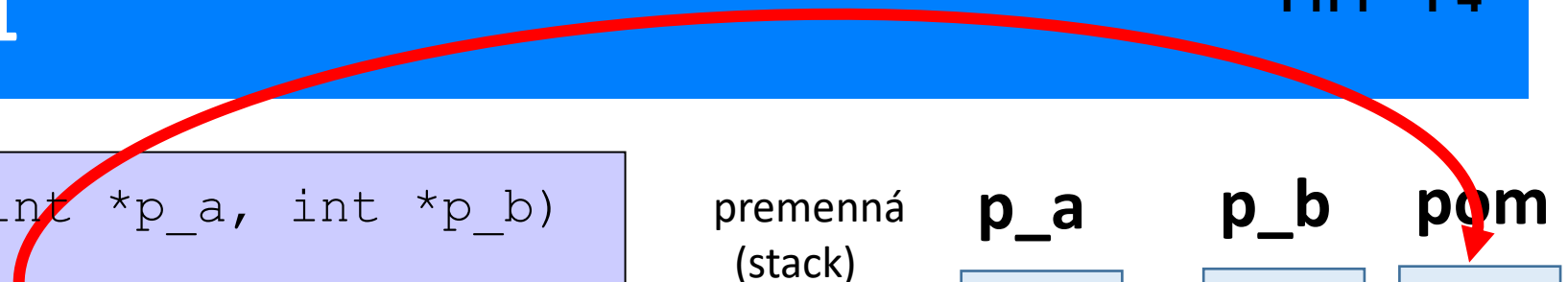
premenná (stack)	p_a	p_b	pom
hodnota:	1234	678	54324
adresa:	988	992	9676

premenná (stack)	a	b
hodnota:	5	13
adresa:	1234	678

Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom = 0;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```



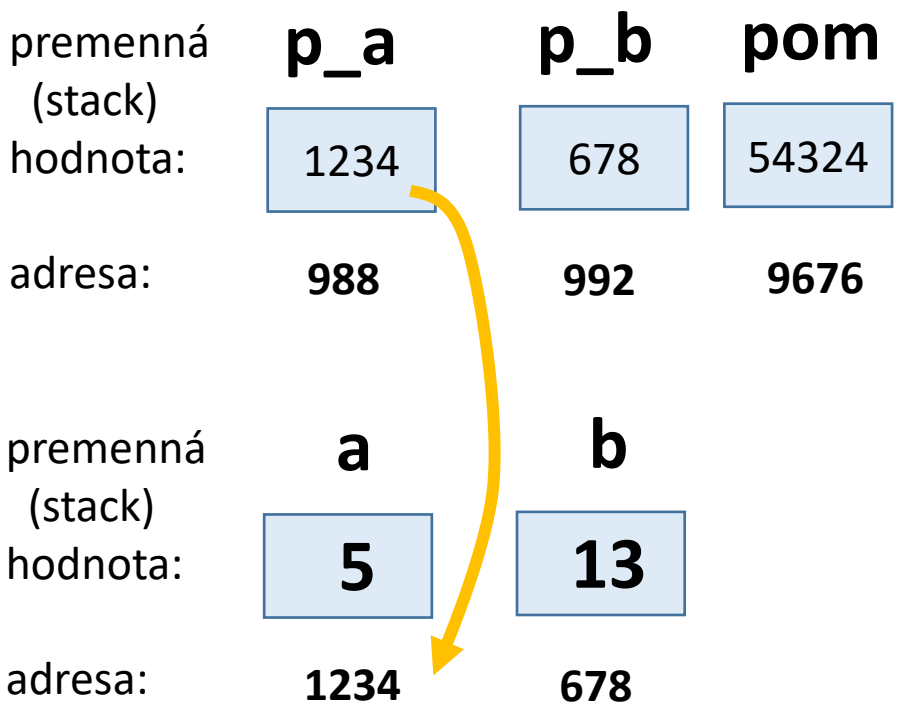
premenná (stack)	p_a	p_b	pom
hodnota:	1234	678	0
adresa:	988	992	9676

premenná (stack)	a	b
hodnota:	5	13
adresa:	1234	678

Príklad - v1

slido slido.com
2558824
PrPr – P4

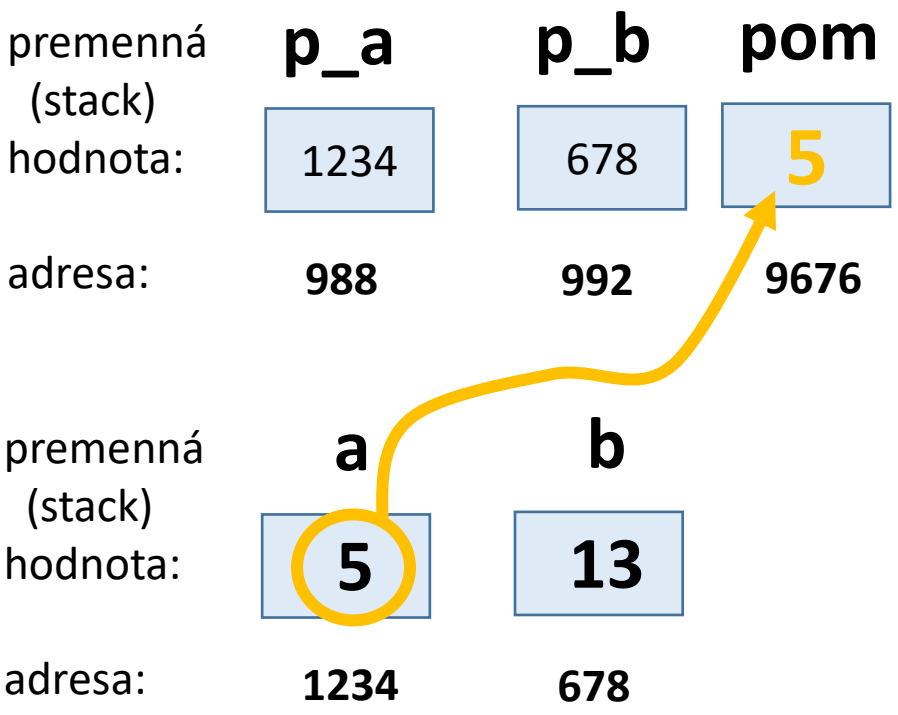
```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```



Príklad - v1

slido slido.com
2558824
PrPr – P4

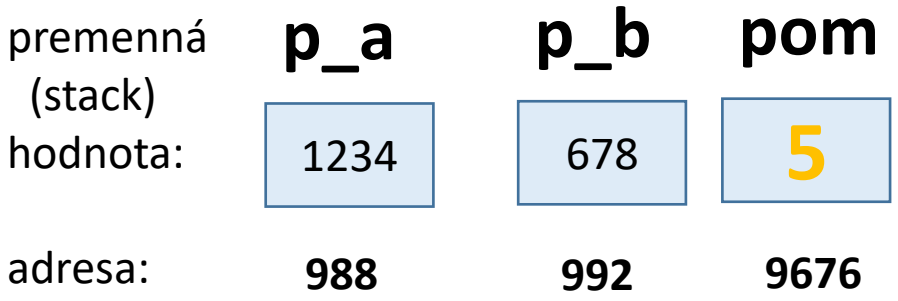
```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

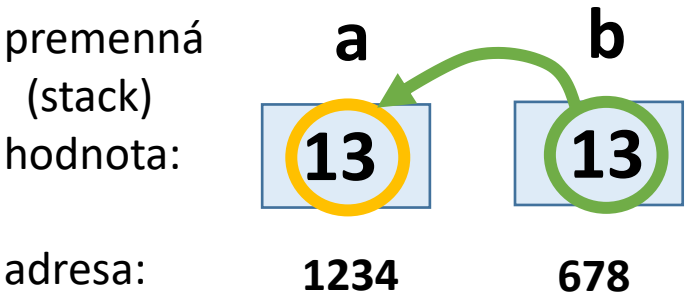


Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

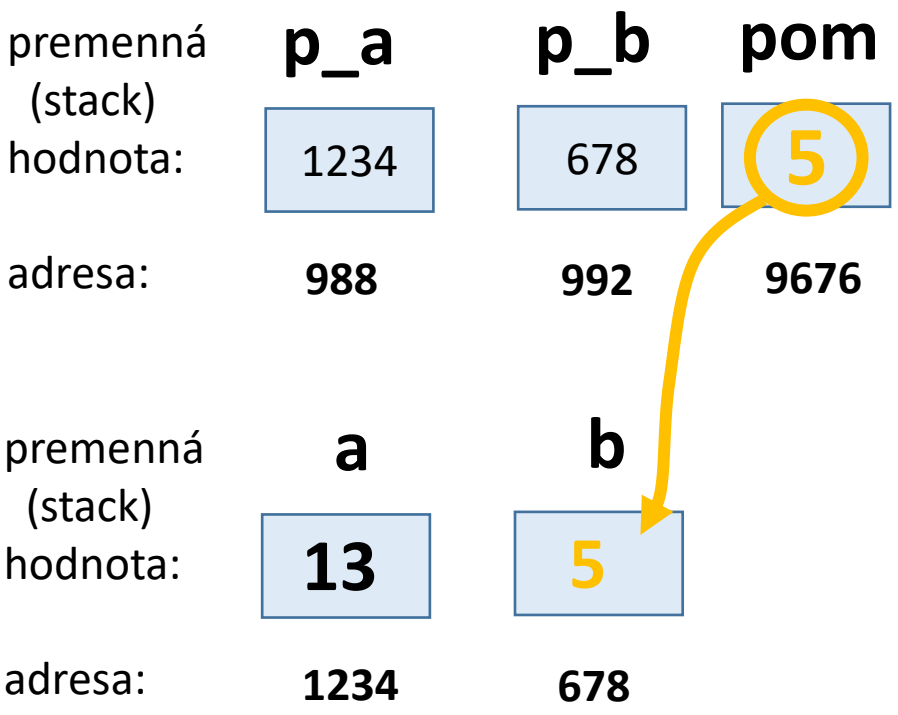




Príklad - v1

slido slido.com
2558824
PrPr – P4

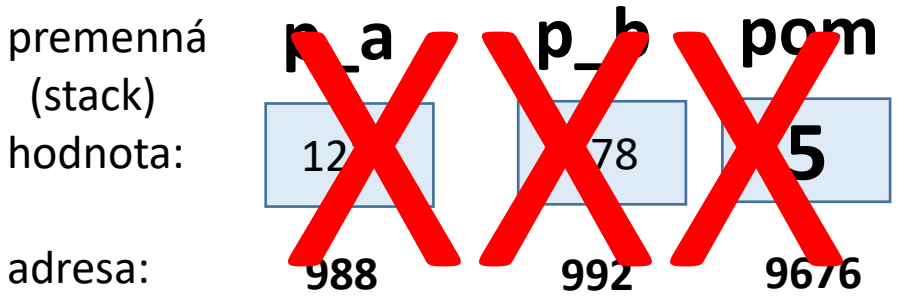
```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

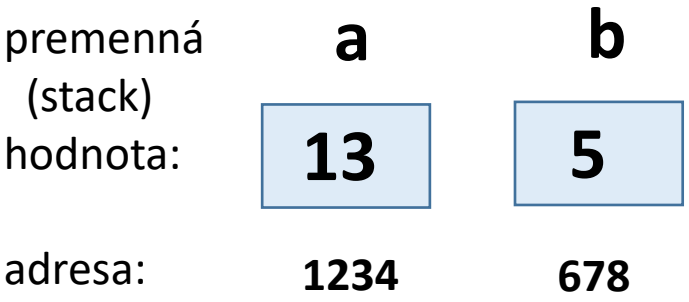


Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

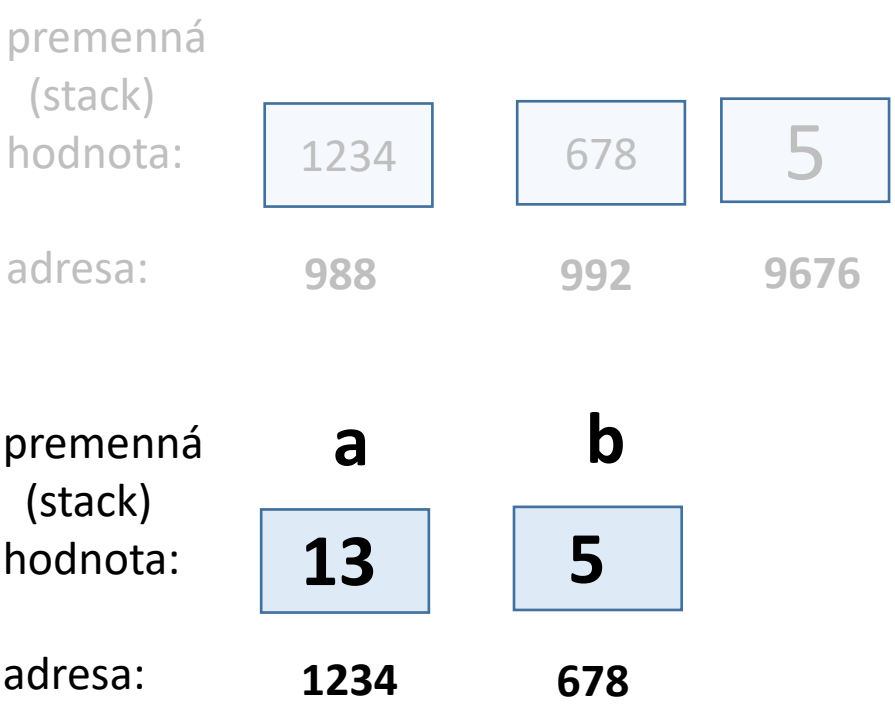




Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

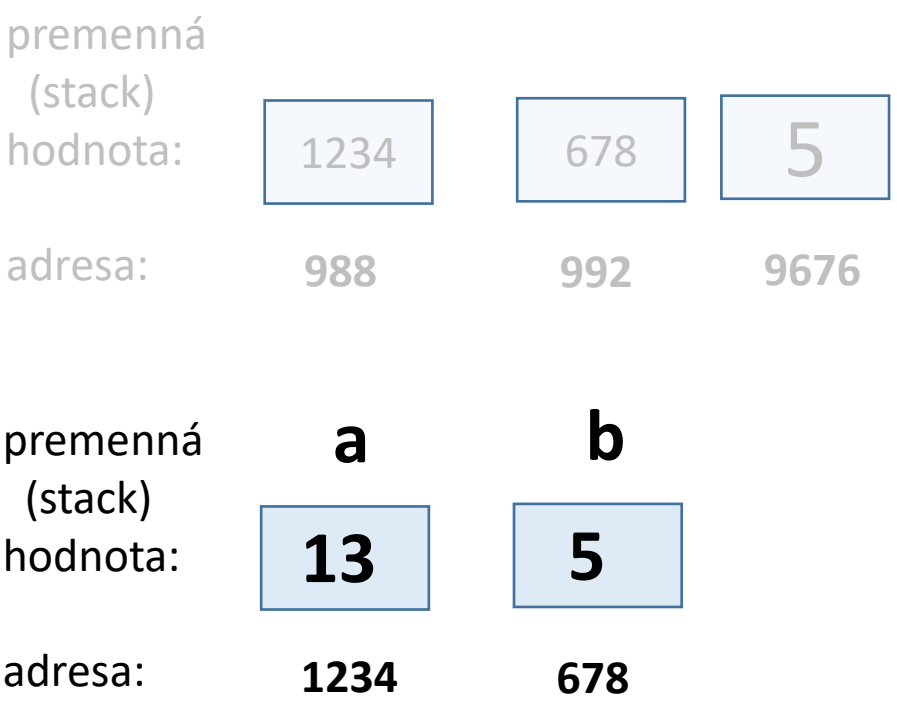



```
a:   5, b:  13
a:  13, b:   5
```

Príklad - v1

slido slido.com
2558824
PrPr – P4

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(&a, &b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

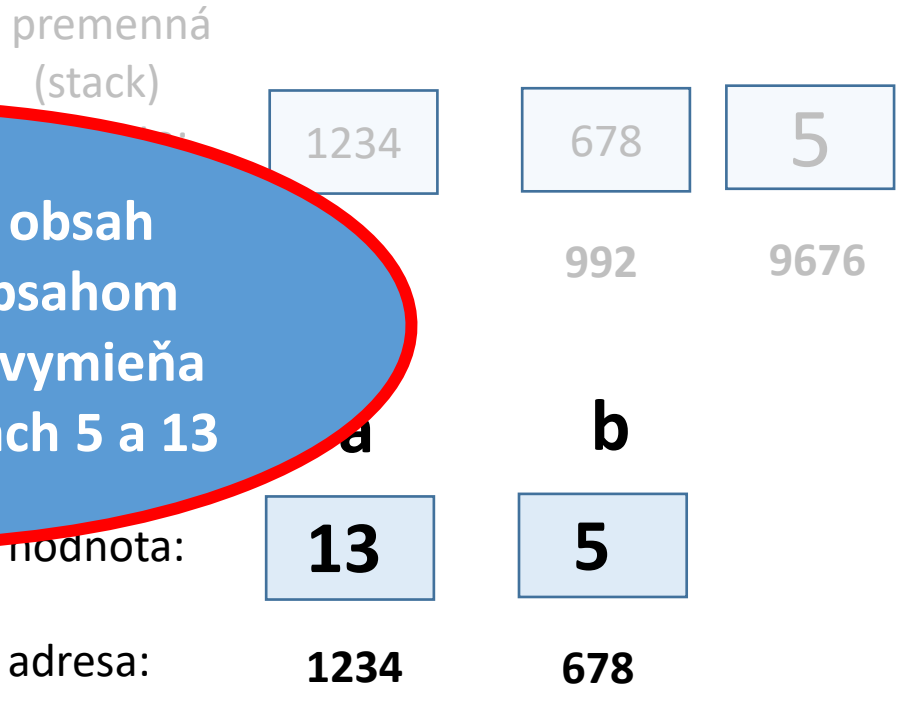



```
a:   5, b:  13
a:  13, b:   5
```

Príklad - v1

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(a, b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

chyba: vymieňa obsah adries, daných obsahom premenných a, b: vymieňa hodnoty na adresách 5 a 13

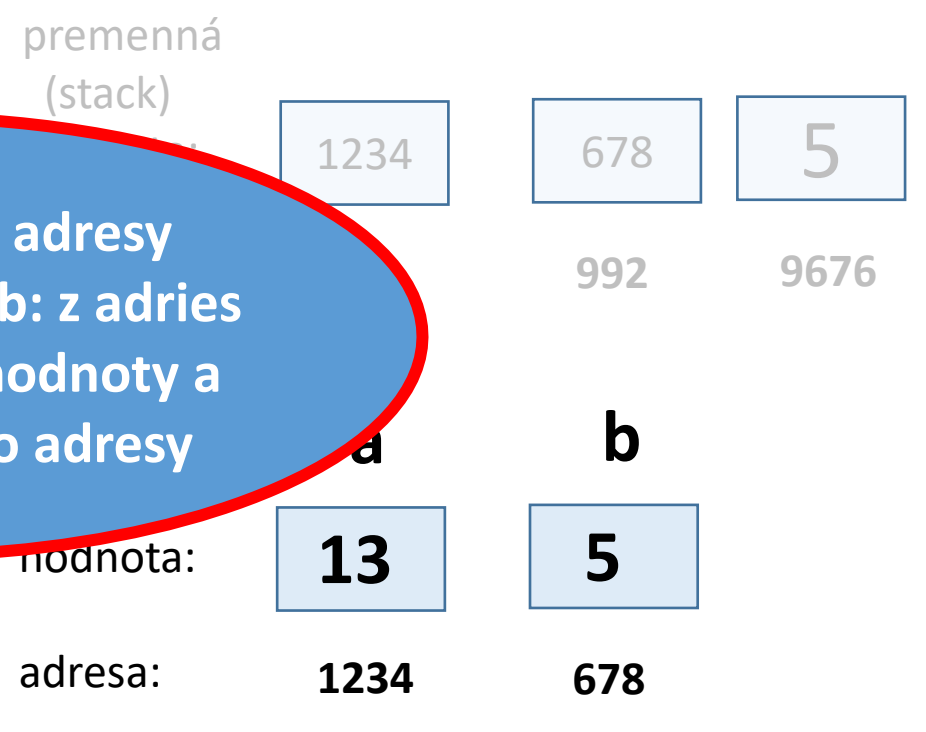



```
a:   5, b:  13
a:  13, b:   5
```

Príklad - v1

```
void vymen(int *p_a, int *p_b)
{
 int pom;
 pom = *p_a;
 *p_a = *p_b;
 *p_b = pom;
}

int main (void) {
 int a=5, b=13;
 printf("a: %3d, b: %3d\n",a,b);
 vymen(*a, *b);
 printf("a: %3d, b: %3d\n",a,b);
 return 0;
}
```

chyba: vymieňa adresy
adries z obsahu a, b: z adries
5 a 13 sa zoberú hodnoty a
tie sa použijú ako adresy




```
a:   5, b:  13
a:  13, b:   5
```

Príklad – v2

```
#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 * r* 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;
}
```

Príklad – v3

slido slido.com
2558824
PrPr – P4

```
void vymen1( char**p_x,  char**p_y)
{
   char *p;
   p = *p_x;
   *p_x = *p_y;
   *p_y = p;
}
```

```
char   c = ' a ',   *p_c  =  &c,
       d = ' b ',   *p_d =  &d;

vymen1(  &p_c,  &p_d);
```

Príklad – v3

```
void vymen2( void **p_x,  void **p_y)
{
   void *p;
   p = *p_x;
   *p_x = *p_y;
   *p_y = p;
}
```

```
char   c = ' a ',   *p_c  =  &c,
       d = ' b ',   *p_d =  &d;

vymen2( (void **)&p_c, (void **)&p_d);
```

Príklad – v3

```
void vymen2( void **p_x,  void **p_y)
{
   void *p;
   p = *p_x;
   *p_x = *p_y;
   *p_y = p;
}
```

```
char   c = ' a ',   *p_c  =  &c,
       d = ' b ',   *p_d =  &d;
int    e = 1,       *p_e = &e,
       f = 13,      *p_f = &f;

vymen2( (void **)&p_c, (void **)&p_d);
vymen2( (void **)&p_e, (void **)&p_f);
```

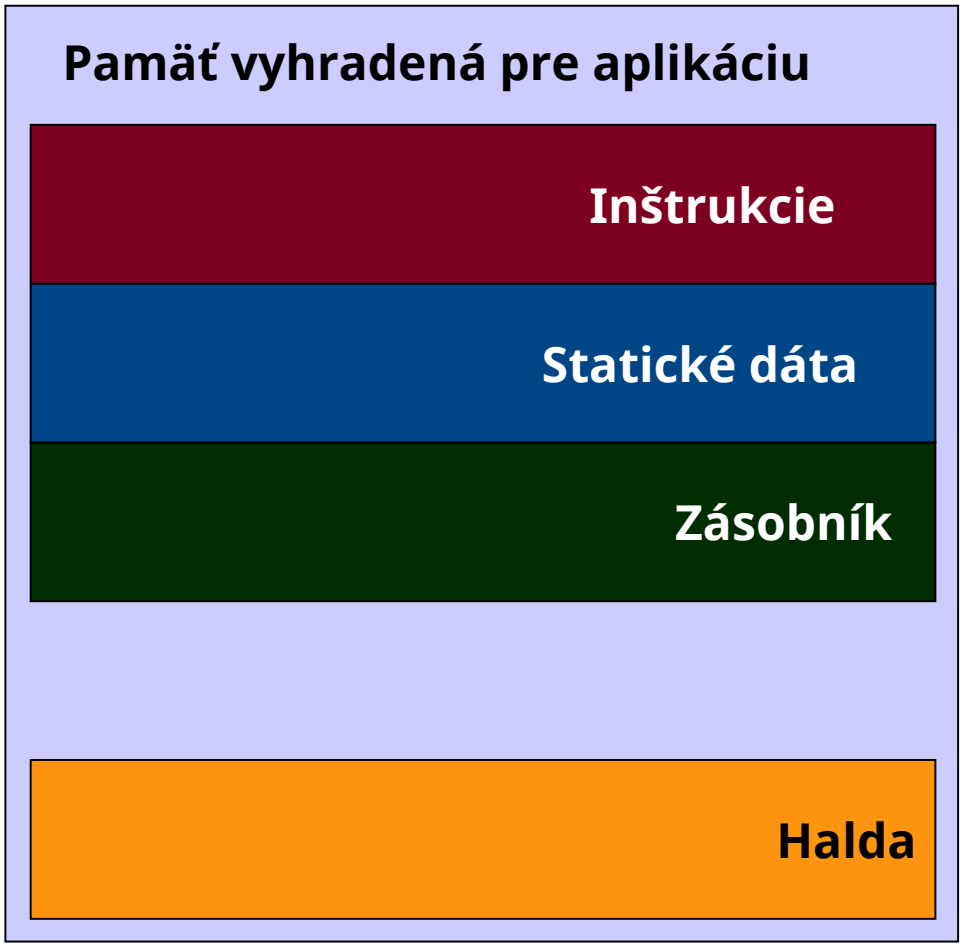
Alokácia pamäte

slido.com
2558824
PrPr – P4

Organizácia pamäte

slido slido.com
2558824
PrPr – P4

- Inštrukcie
  - Asemblerovský kód aplikácie
- Statické dáta (static)
  - Globálne premenné
- Zásobník (stack)
  - Volania funkcií
    (iné pre každú funkciu)
  - Lokálne premenné
- Halda, hromada (heap)
  - Dynamicky alokované premenné
    (malloc, free)
  - Iné pri každom spustení



Halda

slido slido.com
2558824
PrPr – P4

https://stack

Alokácia pamäte

slido slido.com
2558824
PrPr – P4

- vyhradenie pamäťového priestoru
- **Statická alokácia**
  - trvalé alokovanie miesta v dátovej oblasti
  - životnosť: od spustenia po koniec programu
  - riadi operačný systém
- **Dynamická alokácia**
  - pamäťové nároky vznikajú a zanikajú počas behu programu
  - **Životnosť**: od alokovania po uvoľnenie pamäte!
  - riadi programátor

Statická alokácia

slido.com
2558824
PrPr – P4

Statická alokácia pamäte

slido slido.com
2558824
PrPr – P4

- prekladač pozná vopred pamäťové nároky
  - napr. dve premenné typu **`double`** a jednu premennú typu **`char`**
- prekladač sám určí požiadavky pre všetky definované premenné a pri spustení programu sa pre ne alokuje miesto
- počas vykonávania programu sa nemanipuluje s touto pamäťou

Statická alokácia pamäte

slido slido.com
2558824
PrPr – P4

- vymedzuje miesto v dátovej oblasti
- globálne premenné - statické
- nie vždy to stačí
  - napr. rekurzia alebo do pamäte potrebujeme načítať obsah súboru
  - použiť dynamickú alokáciu, alebo vymedzenie pamäte v zásobníku

Vymedzenie pamäte v zásobníku

slido slido.com
2558824
PrPr – P4

- zaisťuje kompilátor pri volaní funkcie
- väčšina lokálnych premenných definovaných vo funkciách
- existencia týchto premenných začína pri vstupe do funkcie a končí pri výstupe z funkcie
- ak chceme prenášať hodnotu premennej medzi jednotlivými volaniami funkcie - nemôže byť premenná alokovaná v zásobníku

Statické jednorozmerné pole

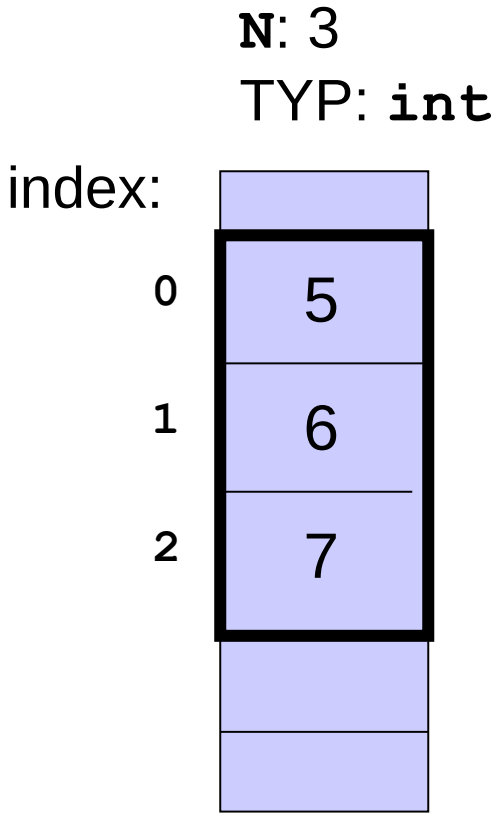
slido slido.com
2558824
PrPr – P4

Základy práce s poliami

- pole je štruktúra zložená z niekoľkých prvkov rovnakého typu (blok prvkov)

```
TYP x[N];
```

statická definícia poľa



- pole obsahuje **N** prvkov
- dolná hranica je vždy **0**
  - $\Rightarrow$ horná hranica je **N-1**
- číslo **N** musí byť známe v čase prekladu
- hodnoty nie sú inicializované na 0
- hranica pola nie je kontrolovaná

Príklady definícií statického poľa

definícia konštanty (inak sa nejedná o statické pole!)

```
#define N 10

int x[N], y[N+1], z[N*2];
```

x má 10 prvkov poľa, od indexu 0 po index 9
y má 11 prvkov poľa, od indexu 0 po index 10
z má 20 prvkov poľa, od indexu 0 po index 19

Prístup k prvkom poľa

slido.com
2558824
PrPr – P4

```
#define N 10

...
   int x[N], i;

   x[0] = 1;

   for (i = 0; i < N; i++)
      x[i] = i+1;

   for (i = 0; i < N; i++)
      printf("x[%d]: %d\n", i, x[i]);
```

priradenie hodnoty do prvého prvku poľa

v cykle priradenie hodnoty postupne všetkým prvkom poľa

výpis prvkov poľa

Prístup k prvkom poľa

slido slido.com
2558824
PrPr – P4

Kompilátor nekontroluje rozsah hodnôt (range-checking) t.j. či index je mimo rozsahu poľa

```
x[10] = 22;
```

- program sa skompiluje, ale hodnota 22 sa zapíše na zlé miesto v pamäti
- prepísanie obsahu iných premenných
- prepísanie časti kódu

Inicializácia poľa


```
int A[3] = { 1, 2, 3 };

int B[4]; //spravne
B[4]={ 1, 2, 3, 4 }; //nespravne
B[0]=1; B[1]=2; B[2]=3; B[3]=4; //spravne

double C[5]={5.1, 6.9};
double C[5]={0};
```

pole tu nedefinujeme

meníme prvky existujúceho pola

jednoduchá inicializácia všetkých prvkov na 0

inicializuje C[0]=5.1, C[1]=6.9 a C[2]..C[4]=0

Porovnávanie poľa

- nie je možné vykonať pomocou operátora ==
- neporovná sa obsah poľa, ale adresa
- meno poľa bez [] vracia adresu na začiatok poľa (väčšinou prvý prvok – závisí od kompilátora)
- treba vykonať prvok po prvku

```
for (int i = 0; i < N; i++) {
      if(A[i] != B[i]) {
          return 0; // false
      }
  }
return 1; // true
```

Kopírovanie poľa

- nie je možné vykonať pomocou operátora =
- neskompiluje sa
- treba vykonať prvok po prvku

```
for (int i = 0; i < N; i++) {
      B[i] = A[i];
  }
```

Statické pole – histogram písmen

slido slido.com
2558824
PrPr – P4

```
#include <stdio.h>
#include <stdlib.h>
 #define N ('Z' - 'A' + 1)  /* 90 – 65 + 1*/

int main()
{
    int I, hist[N]; char  slovo[100];

   scanf("%s", slovo);      /* nacitanie slova */
   for (i = 0; i < N; i++)      /* inicializacia hist */
         hist[i] = 0;             /* naplnenie hist */

  i = 0;
 while ( (i < 100)  &&  (slovo[i] != '\0' ) )
 {
    hist[toupper(slovo[i]) - 'A']++;
    i++;
 }
```

```
   for (i = 0; i < N; i++)      /* vypis hist */
       if ( hist[i] != 0 )
           printf("%c: %d\n", i+'A', hist[i] );

   return(0);
}
```

Statické pole – pole ako parameter funkcie


```
int maximum( int pole[],  int n )
{
  int i,  max = pole[0];
  for (i = 1; i < n; i++)
      if (pole[i] > max)
            max = pole[i];
   return max;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#define N 15

int main()
{
    int i;  int cisla[N];

    for (i = 0; i < N; i++)
        scanf(" %d", & csila[i])     /* naplnenie pola*/

   printf("Maximum je: %d", maximum(cisla, N) ) ;
   return(0);
}
```

Statické pole – pole ako parameter funkcie

```
int maximum( int pole[],  int n )
{
   int i,  max = pole[0];
   for (i = 1; i < n; i++)
       if (pole[i] > max)
              max = pole[i];
    return max;
}
```

parameter sa dá vo funkcii meniť (lebo sa vytvorila jeho lokálna kópia (nezáleží na tom, či ide o statické alebo dynamické pole)

int maximum(int pole[15])

```
#include <stdio.h>
#include <stdlib.h>
#define N 15

int main()
{
    int i;  int cisla[N];

    for (i = 0; i < N; i++)
          scanf(" %d", & csila[i])     /* naplnenie pol
   printf("Maximum je: %d", maximum(cisla, N) ) ;
   return(0);
}
```

Dynamická alokácia

slido.com
2558824
PrPr – P4

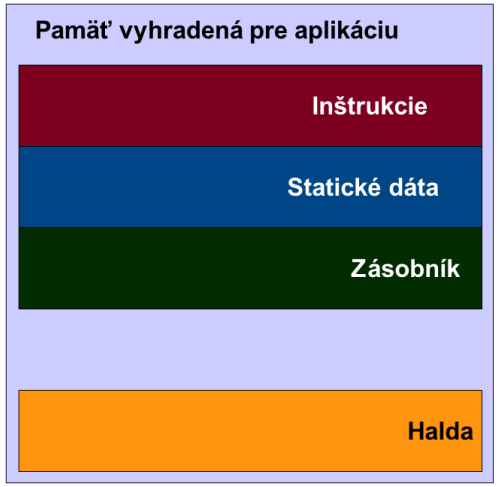
slido slido.com
2558824
PrPr – P4

Dynamická alokácia

- vymedzenie pamäte v halde (heap)
- za behu programu dynamicky prideliť (alokovať) oblasť pamäte určitej veľkosti
- pristupuje sa do nej prostredníctvom ukazovateľov

Organizácia pamäte

- Inštrukcie
  - Asemblerovský kód aplikácie
- Statické dáta (static)
  - Globálne premenné
- Zásobník (stack)
  - Volania funkcií (iné pre každú funkciu)
  - Lokálne premenné
- Halda, hromada (heap)
  - Dynamicky alokované premenné (malloc, free)
  - Iné pri každom spustení



Statická a dynamická aplokácia

slido slido.com
2558824
PrPr – P4

Statická alokácia vymedzuje pamäť na zásobníku

- Automaticky sa uvoľní po dokončení bloku kódu
  - Koniec funkcie, cyklu…
- Výrazne rýchlejšia
  - Na zásobníku nevzniká fragmentácia, ľahké uvoľnenie
  - Vrchol zásobníka je v cache
- Krátkodobé premenné

Dynamická alokácia vymedzuje pamäť na halde

- Existuje do explicitného uvolnenia (alebo do skončenia programu)
- Dlhodobé premenné

Funkcie pre dynamickú alokáciu pamäte

slido slido.com
2558824
PrPr – P4

```
void *malloc(size_t n)
```

- Alokuje pamäť veľkosti n bytov
- Jeden súvislý blok (ako pole)
- Na halde
- Pamäť **nie je** inicializovaná (pozor na „neporiadok“)
- Ak je alokácia neúspešná, vráti NULL

```
#include <stdlib.h>
```

```
void *calloc(size_t n,size_t size_item)
```

- Ako malloc
- Alokuje pamäť veľkosti n * size_item
- Inicializuje pamäť na 0

Prideľovanie pamäte

slido slido.com
2558824
PrPr – P4

počet bytov

```
void *malloc(size_t n)
```

Adresa prvého prideleného prvku - je vhodné pretypovať (generický smerník typu void, ktorý môže ukazovať na ľubovoľný typ premennej). Ak nie je v pamäti dosť miesta, vráti NULL.

Testovanie pridelenia pamäte

slido slido.com
2558824
PrPr – P4

Kontrola, či **`malloc()`** pridelil pamäť:

Dynamický priestor pre 5 celých čísiel

```
int * p_i;

if((p_i = (int *) malloc(5 *sizeof(int))) == NULL)
{
    printf("Nepodarilo sa pridelit pamat\n");
    exit;
}
```

Uvoľnenie alokovanej pamäte

slido slido.com
2558824
PrPr – P4

- Uvoľní alokovanú pamäť na halde
- Funguje na ukazovateľ vrátený z malloc(), calloc() alebo realloc()
  - *nie na hocijaký ukazovateľ (napr. z ukazovateľovej aritmetiky)*

- Prístup k pamäti po zavolaní free()
  - Pamäť môže byť pridelená inej premennej aj keď na ňu ukazovateľ stále ukazuje
  - Use-after-free (exploits)

```
void free(void *)
```

- Free() nemaže obsah pamäti
  - Citlivé údaje je potrebné pre uvoľnením zmazať
- Je vhodné nastaviť uvoľnený ukazovateľ na NULL
  - Opakované uvoľnenie je v poriadku, ak je argument NULL
  - Prístup na adresu je síce nevalidný, ale neprepisujú sa žiadne dáta

Uvoľnenie alokovanej pamäte

- nepotrebnú pamäť je vhodné ihneď vrátiť operačnému systému

```
char *p_c;

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

Zmena veľkosti alokovanej pamäte

slido slido.com
2558824
PrPr – P4

```
void *realloc(void *ptr,size_t size);
```

- prt == NULL, ako malloc()
- prt != NULL, zmení veľkosť akolovaného pamäťového bloku na hodnotu size
- size == 0, závisí od kompilátora (očakávame free(), ale nemusí to tak vždy byť)
- Obsah pôvodného pamäťového bloku je zachovaný
  - Ak je nová veľkosť väčšia ako pôvodná
  - Inak sa pamäťový blok skráti
- Pri zväčšení je dodatočná pamäť neinicializovaná
  - Tak isto ako pri malloc()

Rýchle nastavenie pamäte

slido slido.com
2558824
PrPr – P4

```
void *memset(void *ptr,int value,size_t num);
```

- Nastaví pamäť na zadanú hodnotu (0/-1)
- Výrazne rýchlejšie ako inicializácia cez cyklus
- Pracuje na úrovni bytov
  - Často sa používa spolu so sizeof()

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

```
int array[10];
memset(array, 0, 10*sizeof(int));
```

Dynamická alokácia pamäte

slido slido.com
2558824
PrPr – P4

1. Alokovať potrebný počet bytov
   - Zvyčajne ako `počet_prvkov*sizeof(typ_prvku)`
2. Uložiť adresu pamäte do ukazovateľa daného typu
   - `int *array = (int *) malloc(5 *sizeof(int)));`
3. Uvolniť alokovanú pamäť
   - `free(array);`

Operátor sizeof

- zistí veľkosť dátového typu alebo objektu v bytoch
- vyhodnotí sa v čase prekladu (nezdržuje beh)

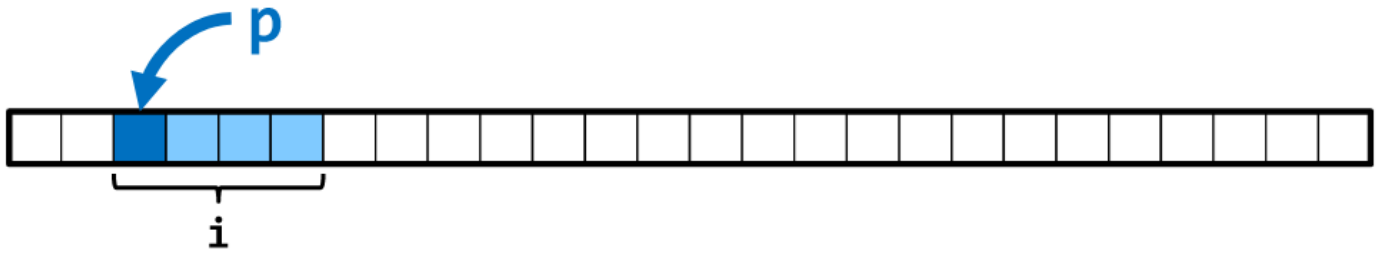
```
int i, *p_i;
i = sizeof(*p_i);
```

počet bytov potrebných na uloženie typu **`int`** – využíva sa často

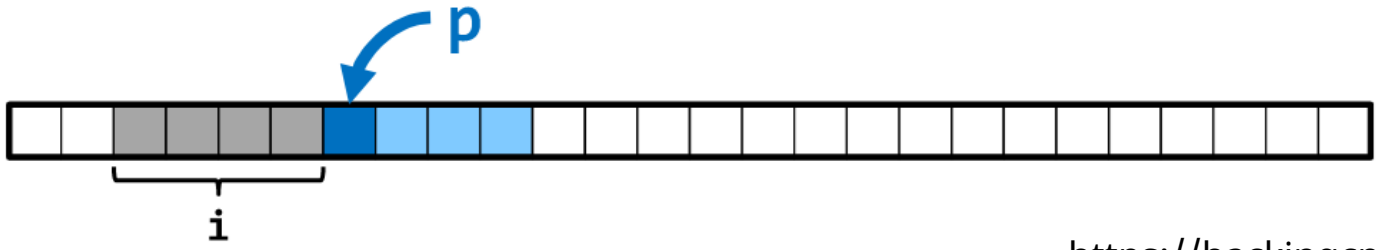
Ukazateľová aritmetika

- `sizeof(int) == 4 byte`

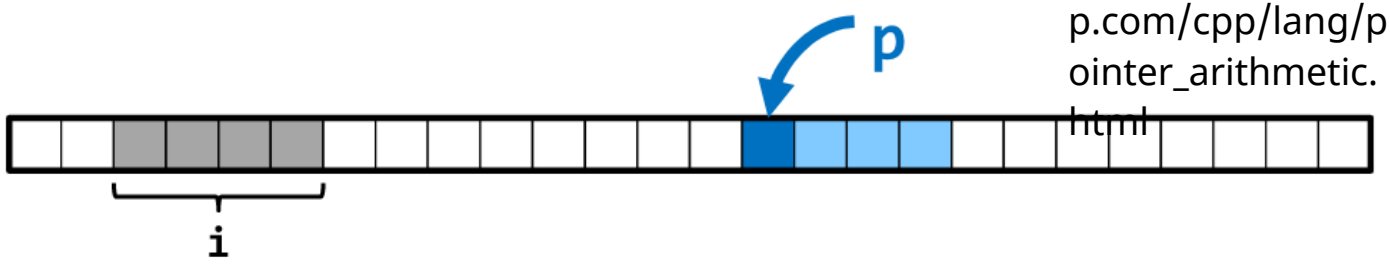
```
int i = 5;
int* p = &i;
```




```
p = &i + 1;
```




```
p += 2;
```



https://hackingcpp.com/cpp/lang/pointer_arithmetic.html

Ukazateľová aritmetika

Vieme:

```
sizeof(char)   == 1
sizeof(int)    == 4
sizeof(double) == 8
```

```
  char c, *p_c=&c; //&c 10
   int i, *p_i=&i; //&i 20
double f, *p_f =&f;//&f 30
```

Potom platí:

```
p_c + 1 == 11
p_i + 1 == 24
p_f + 1 == 38
```

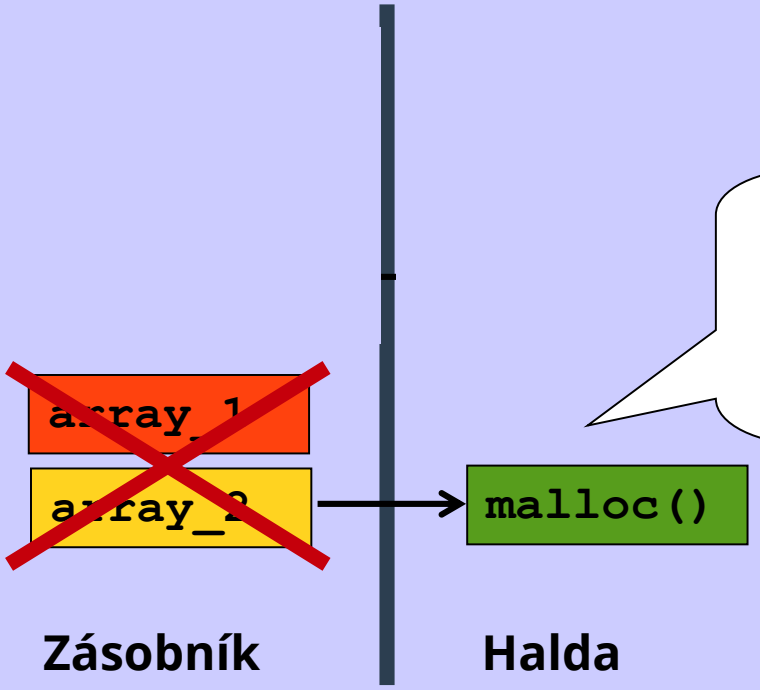
Porovnávanie ukazovateľov s konštantou NULL

- bez explicitného pretypovávania
- **p = NULL**
  - neukazuje na žiadne zmysluplné miesto v pamäti

```
int n, *p;
...
if (n >= 0)
    p = alokuj(n);
else
   p = NULL;
...
if (p != NULL)
  ...
```

Statická a dynamická alokácia pamäte

```
void procedura()
{
    int array_1[10];
    int *array_2 = (int *) malloc (10*sizeof(int));
    ...
}

int main(void)
{
    ...
    procedura();
    ...
    procedura();
    ...
    return 0;
}
```



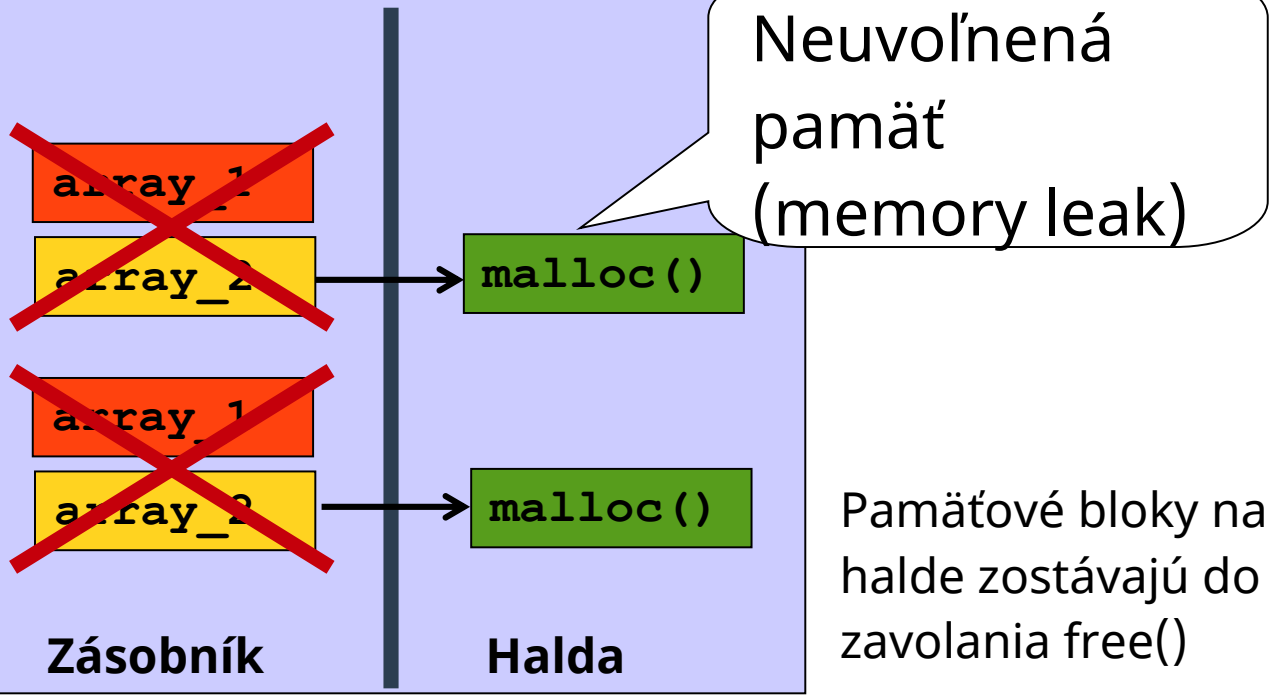
Neuvoľnená pamäť (memory leak)

Pamäťové bloky na halde zostávajú do zavolania free()

Statická a dynamická alokácia pamäte

slido slido.com
2558824
PrPr – P4

```
void procedura()
{
    int array_1[10];
    int *array_2 = (int *) malloc (10*sizeof(int));
    ...
}

int main(void)
{
    ...
    procedura();
    ...
    procedura();
    ...
    return 0;
}
```



Neuvoľnená pamäť (memory leak)

Pamäťové bloky na halde zostávajú do zavolania free()

Neuvoľnená pamäť

slido slido.com
2558824
PrPr – P4

- Dynamicky alokovaná pamäť sa musí uvoľniť
  - Explicitne programátorom
  - C nemá garbage collector
- Valgrind – nástroj na detekciu neuvoľnenej pamäti
  - **`valgrind -v –leak-check=full`** `./testovaný_program`
  - Eclipse Valgrind plugin

http://www.valgrind.org/

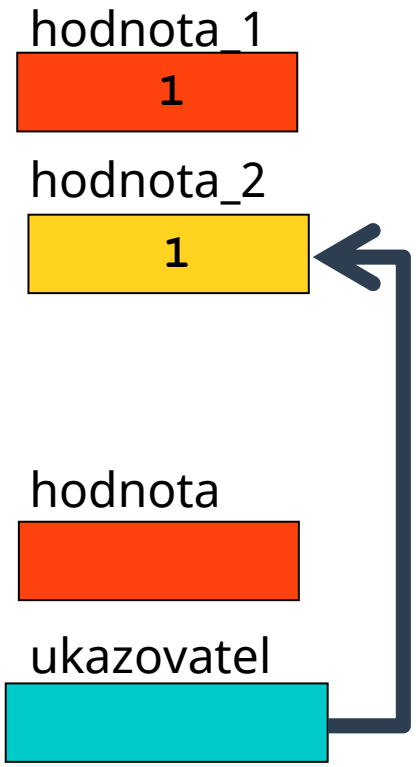
- Microsoft Visual Studio
  - Automaticky zobrazuje neuvoľnenú pamäť v debug režime
  - `_CrtDumpMemoryLeaks();` #include <crtdbg.h>

Parameter funkcie: Hodnota vs ukazovateľ

slido slido.com # 2558824 PrPr – P4

Lokálne premenné funkcie zaniknú (hodnota, ukazovateľ), ale zápis do hodnota_2 zostane

```
void predanieHodnoty(int hodnota, int *ukazovatel)
{
    hodnota = 5;
    *ukazovatel = 7;
}

int main()
{
    int hodnota_1 = 1;
    int hodnota_2  = 1;
    predanieHodnoty(hodnota_1, &hodnota_2);
    return 0;
}
```



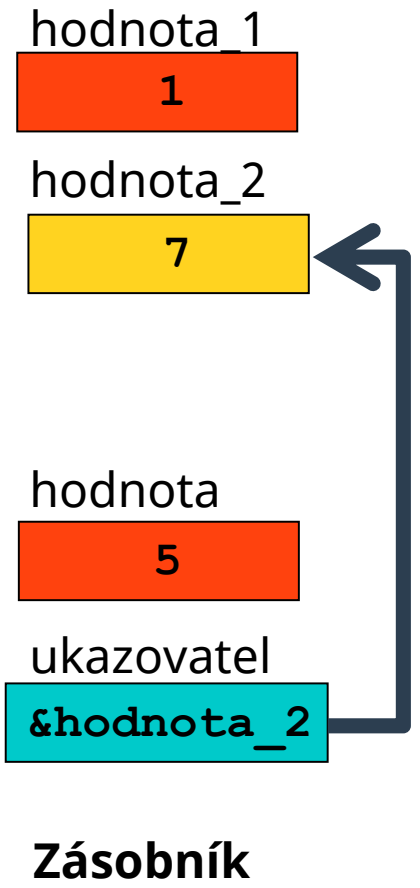
Zásobník

Parameter funkcie: Hodnota vs ukazovateľ

slido slido.com # 2558824 PrPr – P4

Lokálne premenné funkcie zaniknú (hodnota, ukazovateľ), ale zápis do hodnota_2 zostane

```
void predanieHodnoty(int hodnota, int *ukazovatel)
{
    hodnota = 5;
    *ukazovatel = 7;
}

int main()
{
    int hodnota_1 = 1;
    int hodnota_2  = 1;
    predanieHodnoty(hodnota_1, &hodnota_2);
    return 0;
}
```

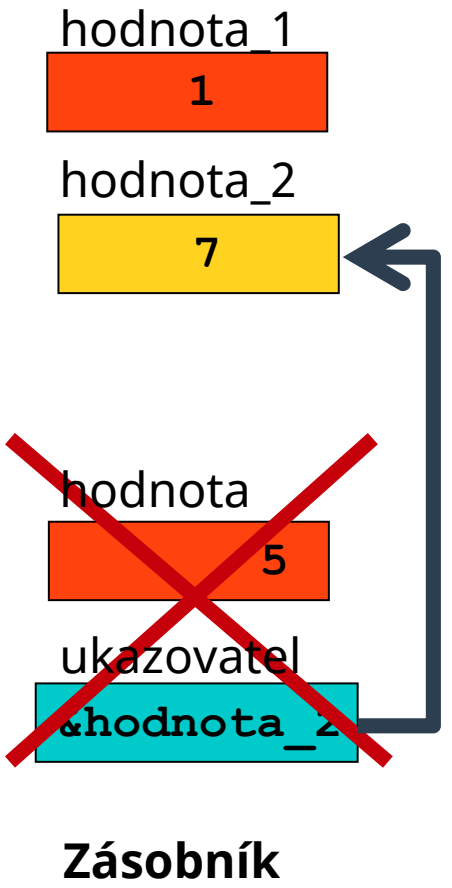


Parameter funkcie: Hodnota vs ukazovateľ

slido slido.com # 2558824 PrPr – P4

Lokálne premenné funkcie zaniknú (hodnota, ukazovateľ), ale zápis do hodnota_2 zostane

```
void predanieHodnoty(int hodnota, int *ukazovatel)
{
    hodnota = 5;
    *ukazovatel = 7;
}

int main()
{
    int hodnota_1 = 1;
    int hodnota_2  = 1;
    predanieHodnoty(hodnota_1, &hodnota_2);
    return 0;
}
```



Zásobník

Parameter funkcie: Hodnota vs ukazovateľ

slido slido.com # 2558824 PrPr – P4

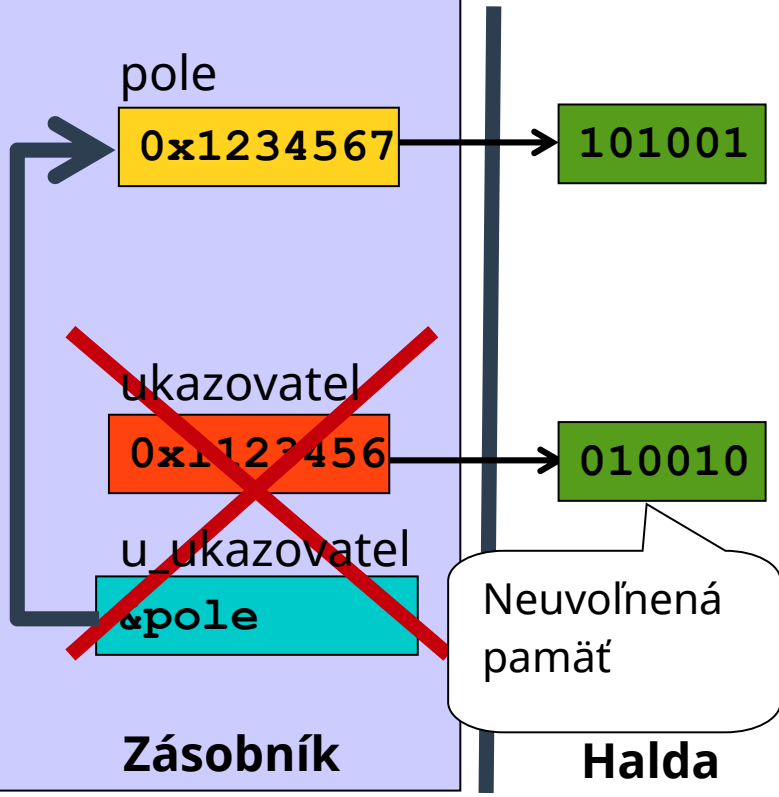
Lokálne premenné funkcie zaniknú (hodnota, ukazovateľ), ale zápis do hodnota_2 zostane

```
void predanieHodnoty(int hodnota, int *ukazovatel)
{
    hodnota = 5;
    *ukazovatel = 7;
}

int main()
{
    int hodnota_1 = 1;
    int hodnota_2  = 1;
        int *p_h2 = &hodnota_2;
    predanieHodnoty(hodnota_1, p_h2);
    return 0;
}
```

Predanie dynamického poľa funkciou

Ukazovateľ na ukazovateľ

```
void alokujPole(int *ukazovatel, int **u_ukazovatel)
{
    // adresa sa stratí, neuvolnena pamat
    ukazovatel = malloc(10*sizeof(int));
    // warning: integer from pointer
    *ukazovatel = malloc(10 *sizeof(int));
    // OK
    *u_ukazovatel = malloc(10*sizeof(int));
}

int main()
{
    int *pole = NULL;
    alokujPole(pole, &pole);
    free(pole);
    return 0;
}
```



Predanie dynamického poľa funkciou

Ukazovateľ na ukazovateľ

```
void alokujPole(int *ukazovatel, int **u_ukazovatel)
{
    // adresa sa stratí, neuvolnena pamat
    ukazovatel = malloc(10*sizeof(int));
    // warning: integer from pointer
    *ukazovatel = malloc(10 *sizeof(int));
    // OK
    *u_ukazovatel = malloc(10*sizeof(int));
}

int main()
{
    int *pole = NULL;
    int **p_pole = &pole;
    alokujPole(pole, p_pole);
    free(pole);
    return 0;
}
```

Polia a ukazovatele

Ak sú definované `int x[1],*p_x;`

- `x` je statický ukazovateľ, nemôže byť zmenený, ale
- `*x` je obsah staticky alokovanej pamäti.

  t.j. `*x = 2;` je to isté ako `x[0] = 2;`
  - Smerník `p_x` nie je inicializovaný
  - priradenie `*p_x = 2;` je staticky správne, ale dynamicky chybné (chceme meniť obsah na neznámej adrese, ktorá nebola alokovaná)
  - `p_x = x;` – ukazujú na rovnakú adresu v pamäti (dynamický smerník nasmerujeme na začiatok statického poľa – ok.)
  - `x = p_x;` – chybné priradenie, x je konštantný smerník jeho hodnotu nemôžeme meniť (chceme zmeniť začiatok statického poľa na adresu kde ukazuje smerník – statické pole nemôžeme v pamäti premiestňovať)

Polia a ukazovatele - operátor []

- p[n] == hodnota na adrese ukazovateľa + n

```
#define N 10

...
   int x[N], i;

   for (i = 0; i < N; i++)
       x[i] = i+1;

   for (i = 0; i < N; i++)
      printf("x[%d]: %d\n", i, x[i]);

   for (i = 0; i < N; i++)
      printf("x[%d]: %d\n", i, *(x+i));
```

```
X[0] = 1
X[1] = 2
X[2] = 3
X[3] = 4
X[4] = 5
X[5] = 6
X[6] = 7
X[7] = 8
X[8] = 9
X[9] = 10
X[0] = 1
X[1] = 2
X[2] = 3
X[3] = 4
X[4] = 5
X[5] = 6
X[6] = 7
X[7] = 8
X[8] = 9
X[9] = 10
```

Polia a ukazovatele

```
int *p;
p = (int *) malloc(4 * sizeof(int));
```

`*p` je smerník na blok pamäti alokovanej pomocou funkcie `malloc()`

`p` je dynamické pole, ktoré vzniká v čase behu programu

Platí:

- `p[0] == *p`
- `p[1] == *(p + 1)`
- `p[2] == *(p + 2)`
- `p[3] == *(p + 3)`

Rozdiel medzi statickými a dynamickými poliami je v definícii a v spôsobe prideľovania pamäte

Dynamická alokácia

- Pri uvoľnení nastavte premennú späť na NULL
  - Opakované uvoľnenie nie je problém
  - Správnosť ukazovateľa sa nedá testovať, NULL áno
- Dynamicky alokovanú pamäť priraďujeme do ukazovateľa
  - `sizeof(ukazovatel)` vracia veľkosť ukazovateľa, nie poľa
- Dynamická alokácia (a jej uvoľnenie) nerobí nič s pamäťou
  - Neinicializovaná pamäť
  - Zabudnuté dáta (heslá...)

Dynamická a statická alokácia

slido slido.com
2558824
PrPr – P4

Pamäť alokovaná v dobe prekladu s pevnou dĺžkou v statickej časti

- Konštanty, reťazce, konštantné pole
- `const int n = 10;` (neskôr)
- Dĺžka známa v dobe prekladu
- Alokované v statickej sekcii programu (nachádzajú sa v nespustenom programe)

Pamäť alokovaná za behu na zásobníku, dĺžka známa v dobe prekladu

- Lokálne premenná, lokálne pole
- `int pole[10];`
- Pamäť je alokovaná a uvoľnená automaticky

Pamäť alokovaná za behu na halde, dĺžka nie je známa v dobe prekladu

- Alokácia a uvoľnenie explicitne pomocou funkcií malloc a free
- Neuvoľnená pamäť
- `int *pole=malloc(velkost*sizeof(int)); free(pole);`

Ďakujem vám za pozornosť!

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*