6. Pole a ukazatele, textové řetězce

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Pole a ukazatele
  - Pole a ukazatele
  - Vícerozměrná pole
- Část 2 – Textové řetězce
  - Textové řetězce
  - Funkce main
- Část 3 – Zadání 5. domácího úkolu

Část I

Pole a ukazatele

I. Pole a ukazatele

Pole a ukazatele

- Ukazatel ukazuje na vyhrazenou část paměti proměnné

```
int *p;      // ukazatel (adresa)
sizeof(p);   // velikost promenne
```

- Pole je označení souvislého bloku paměti

```
int a[10];   // souvisly blok pameti
sizeof(a);   // 10*sizeof(int)
```

- Obě proměnné odkazují na paměť, kompilátor s nimi však pracuje rozdílně
  - Proměnná typu pole je symbolické jméno pro místo v paměti, kde jsou uloženy hodnoty prvků pole

    Kompilátor nahrazuje jméno přímo paměťovým místem
  - Ukazatel obsahuje adresu, na které je příslušná hodnota

    Nepřímé adresování
- Při předávání pole jako parametru funkce je předáváno pole jako ukazatel.

Pole a ukazatele

- Proměnná typu tříprvkové pole int

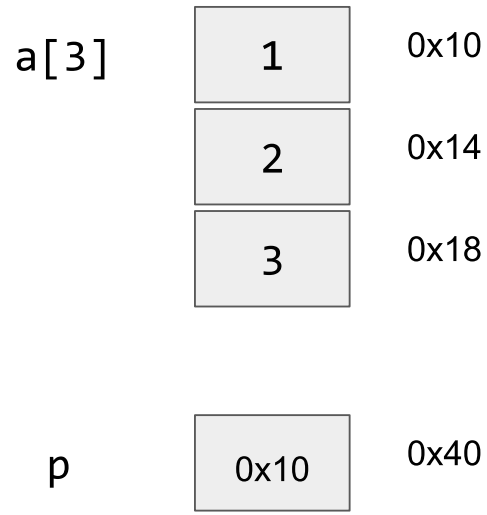
  `a[3] = {1,2,3};`

  Odkazuje na adresu prvního prvku pole

- Proměnná ukazatel na integer

  `int *p = a;`

  Obsahuje adresu prvního prvku pole



- Hodnota `a[0]` přímo reprezentuje hodnotu na adrese `0x10`.

- Hodnota `p` je adresa `0x10`, kde je uložena hodnota 1. prvku pole.
- Přiřazení `p = a` je možné.
- Kompilátor zajistí přiřazení adresy prvního prvku do ukazatele.
- Přístup k 2. prvku lze použít jak `a[1]` tak `p[1]`.
- Oběma přístupy se dostaneme na příslušné prvky pole, způsob je však odlišný

Ukazatele využívají tzv. pointerovou aritmetiku.

Ukazatelová artitmetika

- S ukazateli lze provádět aritmetické operace + a −, tj. přičítat nebo odčítat celé číslo
  - ukazatel = ukazatel stejného typu + (nebo -) a celé číslo (int)
  - Nebo lze používat zkrácený zápis např. ukazatel += 1 a unární operátory např. ukazatel++
- Aritmetické operace jsou užitečné pokud ukazatel odkazuje na více položek daného typu (souvislý blok paměti)
  - Např. pole položek příslušného typu
  - Dynamicky alokovaný souvislý blok paměti
- Přičtením hodnoty celého čísla k ukazateli posouváme hodnotu ukazatele na další prvek, např.

```
int a[10];
int *p = a;
int i = *(p+2); //odkazuje na hodnotu a[2]
```

Pole a ukazatele – příklad

```c
#include<stdio.h>

int main(void)
{
    int *p;          // ukazatel na integer
    int (*ptr)[5]; // ukazatel na pole peti prvku int
    int arr[5];

    p = arr;         // ukazatel na prvni prvek pole
    ptr = &arr;      // ukazatel na cele pole

    printf("sizeof(p) = %lu\n", sizeof(p));
    printf("sizeof(ptr) = %lu\n", sizeof(ptr));
    printf("p = %p, ptr = %p\n", p++, ptr++);
    printf("p = %p, ptr = %p\n", p, ptr);

    return 0;
}
```

lec06/array-pointer.c

Funkce s polem jako argumentem

```
3  void fce(int A[]) {
4    int B[] = { 2, 4, 6 };
5    printf("[A] = %lu, [B] = %lu\n", sizeof(B), sizeof(B));
6    for (int i = 0; i < 3; ++i) {
7      printf("A[%i]=%i B[%i]=%i\n", i, A[i], i, B[i]);
8    }
9  }

13   int array[] = { 1, 2, 3 };
14   fce(array);
```

lec06/array-argument.c

- Po překladu (`gcc -std=c99`) na **amd64**
  - `sizeof(A)` vrátí velikost 8 bajtů (64-bitová adresa)
  - `sizeof(B)` vrátí velikost 12 bajtů (3×4 bajty – `int`)
- Pole se funkcím předává jako ukazatel na první prvek

I. Pole a ukazatele

Vícerozměrná pole

- Pole lze definovat jako vícerozměrná, např. 2D matice

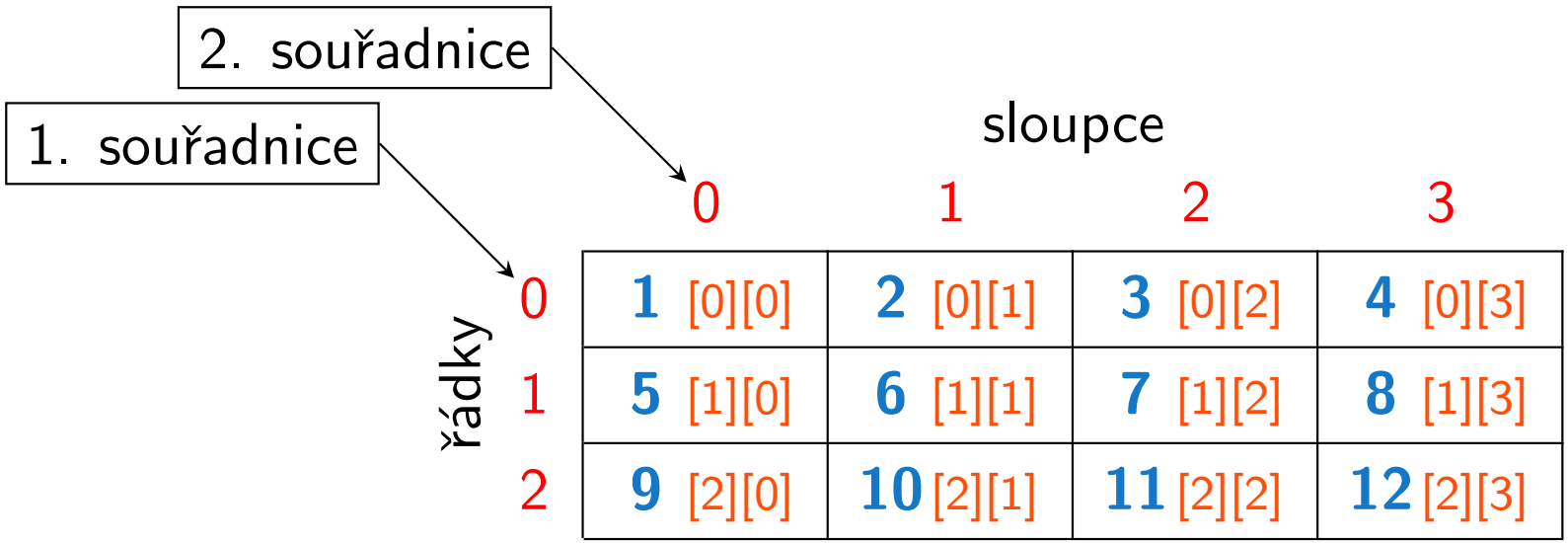
```
4  int m[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
5  printf("[m]: %lu == %lu\n", sizeof(m), 3*3*sizeof(int));
7  for (int r = 0; r < 3; ++r) {
8    for (int c = 0; c < 3; ++c) {
9      printf("%3i", m[r][c]);
10   }
11   printf("\n");
12 }
```

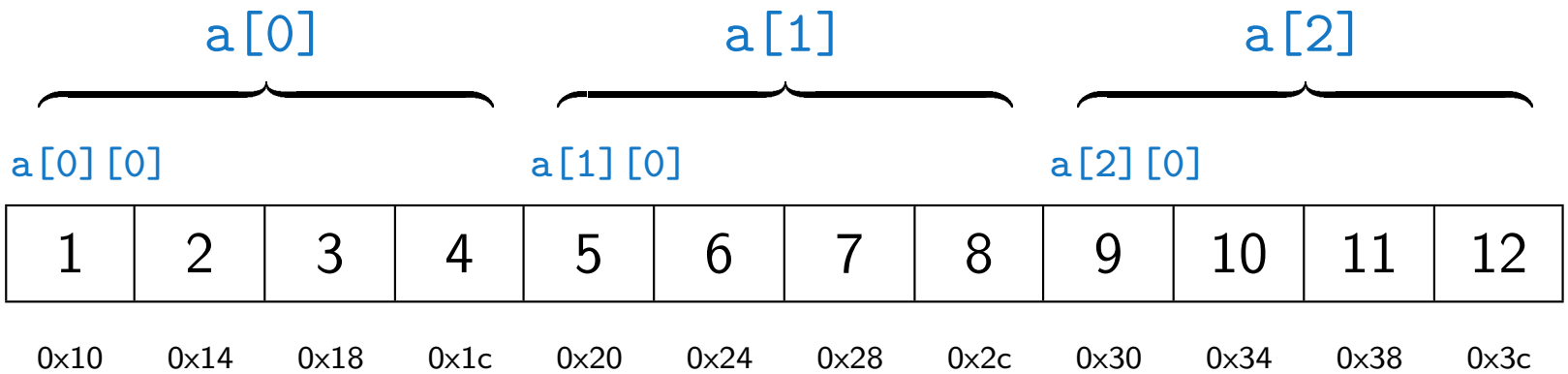
lec06/array-two-dimensions.c

```
[m]: 36 == 36
  1  2  3
  4  5  6
  7  8  9
```

Vícerozměrná pole

```
int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```





Vícerozměrná pole

```
int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

- Každý řádek lze považovat za 1D pole
- `a` je pole tří prvků, každý prvek je pak 1D pole čtyř prvků

`a` – první prvek pole `a`

`a+1` – druhý prvek pole `a`

`a+2` – třetí prvek pole `a`

`*a` – první prvek pole `a[0]`

`*(a+1)` – první prvek pole `a[1]`

`*(a+2)` – první prvek pole `a[2]`

Vnitřní reprezentace vícerozměrných polí

- Vícerozměrné pole je vždy souvislý blok paměti

```
int *pm = (int *)m; // ukazatel na souvislou oblast
printf("m[0][0]=%i m[1][0]=%i\n", m[0][0], m[1][0]);
printf("pm[0]=%i pm[3]=%i\n", pm[0], pm[3]);
```

- Dvourozměrné pole lze také definovat jako ukazatel na ukazatele (pole ukazatelů) na hodnoty konkrétního typu, např.
  - `int **a;` – ukazatel na ukazatele
  - V obecném prípade však takový ukazatel nemusí odkazovat na souvislou oblast, kde jsou alokovány jednotlivé prvky.
  - Při přístupu jako do jednorozměrného pole `int *b = (int *)a;` tedy nelze garantovat přístup do druhého řádku jako v přechozím příkladě.

Vícerozměrná pole jako parametr funkce

- Parametr funkce je ukazatel na pole, např. typu `int`

```
int (*p)[3] = m; // pointer to array of int
printf("Size of p: %lu\n", sizeof(p));
printf("Size of *p: %lu\n", sizeof(*p)); // 3 *
  sizeof(int) = 12
```

- Funkci nelze deklarovat s argumentem typu `[][]` např.

```
int   fce ( int   a [][]) ;
```

  - kompilátor nemůže správně spočítat index, pro přístup na `a[i][j]` se používá adresová aritmetika jinak

    Pro int m[row][col] totiž m[i][j] odpovídá hodnote na adrese *(m + col * i + j)

- Je však možné funkci deklarovat například jako
  - `int g(int a[]);` což odpovídá deklaraci `int g(int *a);`
  - `int fce(int a[][13]);` – je znám počet sloupců
  - nebo `int fce(int a[3][3]);`

Část II

Textové řetězce

II. Textové řetězce

Textové řetězce

Funkce main

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků uzavřená v uvozovkách

  `"Sud kulatý, rys tu pije!\n"`

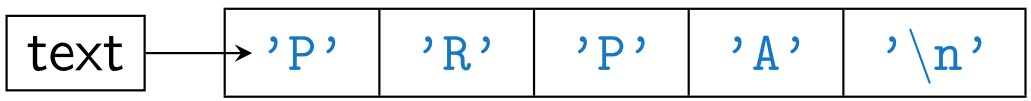
  - Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí

    `"Tu je kára," " ten to ryje!\n"`

    se sloučí do

    `"Tu je kára, ten to ryje!\n"`
- Typ – pole typu char zakončené znakem `'\0'`
  - Pole pro uložení řetězce musí být o jeden prvek větší než délka samotného řetězcového literálu
  - Pokud ukončovací znak chybí, překladač nepozná konec řetězce
  - Při inicializaci literálem se automaticky vytvoří správně velké pole

```
char text[] = "PRPA";
```



Textový řetězec – pole znaků

- Textový řetězec můžeme inicializovat jako pole znaků `char[]`

```
char str[] = "123";
char s[] = {'5', '6', '7'};

printf("sizeof(str): %lu\n", sizeof(str));
printf("sizeof(s)  : %lu\n", sizeof(s));
printf("str: '%s'\n", str);
printf("  s: '%s'\n", s);
```

lec06/string-array.c

```
sizeof(str): 4
sizeof(s)  : 3
str: 123
  s: 567890
```

Pokud není řetězec ukončen, výpis pokračuje, dokud není nalezen znak '\n'

Textový řetězec – ukazatel

- Na textový řetězec lze odkazovat ukazatelem na znak `char*`

```
char a[] = "ahoj";
char *b = "ahoj";
char *c = "ahoj";

printf("sizeof(a): %lu\n", sizeof(a));
printf("sizeof(b): %lu\n", sizeof(b));

printf ("adresa a: %p\n", a);
printf ("adresa b: %p\n", b);
printf ("adresa c: %p\n", c);
```

lec06/string-pointer.c

```
sizeof(a): 5
sizeof(b): 8
 adresa a: 0x7ffe708d1b60
 adresa b: 0x4006d4
 adresa c: 0x4006d4
```

Inicializace textového řetězce – pole a ukazatel

- Inicializace řetězce reprezentovaného polem
  - Alokace správně velkého pole
  - Nakopírování literálu na dané místo v paměti
  - Lze získat informaci o velikosti řetězce
- Inicializace řetězce reprezentovaného ukazatelem
  - Kompilátor umístí literál do zvláštního segmentu paměti
  - Ukazatel je inicializován adresou
  - Pokud v tomto segmentu již existuje stejný literál, ukazatel je inicializován jeho adresou
  - Tento segment paměti je pouze pro čtení, proto nelze měnit libovolně znaky řetězce

    Více o segmentech paměti na příští přednášce
  - Nelze přímo získat informaci o velikosti řetězce

    Operátor sizeof vrací velikost ukazatele, 8 bytů pro 64b OS

Textový řetězec – přiřazení

- Textový řetězec reprezentovaný polem `char[]`

```
char str[5];
str = "prpa"; // nelze zkompilovat
```

  - Podle definice nelze aplikovat operátor přiřazení na datový typ pole
  - Je třeba, aby proměnná na levé straně byla modidikovatelná, čemuž v případě pole není – proměnná reprezentuje adresu již alokovaného pole
  - Přiřazení lze zajistit znak po znaku nebo pomocí funkce `strcpy()`

- Textový řetězec reprezentovaný ukazatelem `*char`

```
char *str;
str = "prpa"; // v pořádku
```

  - Literál je vytvořen v paměti a jeho adresou je inicializován dosud neinicializovaný ukazatel

Načtení textového řetězce funkcí `scanf()`

- Použitím `%s` může dojít k přepisu paměti

```
char str0[4] = "PRP";
char str1[5];
printf("String str0 = '%s'\n", str0);
printf("Enter 4 chars: ");
scanf("%s", str1);
printf("You entered string '%s'\n", str1);
printf("String str0 = '%s'\n", str0);
```

- Načtení maximálně 4 znaků zajistíme řídicím řetězcem `%4s`

```
char str0[4] = "PRP";
char str1[5];
scanf("%4s", str1);
printf("You entered string '%s'\n", str1);
printf("String str0 = '%s'\n", str0);
```

Načtení textového řetězce funkcí `scanf()`

- Načtení celého řádku (včetně bílých znaků)

```
char str[100];
scanf("%[^\n]", str);
```

Místo \n může být libovolný jiný znak

- Omezení vstupních znaků

```
scanf("%[0-9]", str); // podobně [a-z], [0-9a-z]
```

```
1234abc
1234
```

```
scanf("%[123]", str); // podobně [ABC]
```

```
112233126870
11223312
```

Zjištění délky textového řetězce

- Textový řetězec v C je pole (`char[]`) nebo ukazatel (`char*`) odkazující na část paměti, kde je uložena příslušná posloupnost znaků.
- Textový řetězec je zakončen znakem `'\0'`
- Délku textového řetězce lze zjistit sekvenčním procházením znak po znaku až k `'\0'`

```
int delka(char *str) {
  int ret = 0;
  while (str && (*str++) != '\0') {
    ret += 1;
  }
  return ret;
}
...
char *text = "Hello PRP!\n";
printf("%i %lu\n", delka(text), strlen(text));
```

Práce s textovými řetězci

- Základní operace jsou definovány v knihovne `<string.h>`
  - Funkce předpokládající dostatečný rozsah alokovaných polí

```
int strlen (char *s);
char* strcpy (char *dst, char *src);
int strcmp (const char *s1, const char *s2);
strcat ();
```

  - Funkce s explicitním limitem na maximální délku řetězců:

```
char* strncpy(char *dst, char *src, size_t len);
int strncmp (const char *s1, const char *s2, size_t len);
strncat ();
```

- Převod řetězce na číslo – `<stdlib.h>`

  `atoi()`, `atof()` – převod celého a necelého čísla

```
long strtol(const char *nptr, char **endptr, int base);
double strtod(const char *nptr, char **restrict endptr);
```

Porovnávání textových řetězců

- Řetězce nelze porovnávat pomocí relačních operátorů – proměnné reprezentují adresy

```
// např.  str1 = 0x7f42, str2 = 0x654d
void doSomething (char *str1, char *str2) {
  if (str1 > str2) { // porovnává 0x7f42 > 0x654d!
```

- Porovnávání řetězců pomocí funkce `strcmp`

```
int compResult = strcmp (str1, str2);
if (compResult == 0) {
  // equal
} else if (compResult < 0) {
  // str1 comes before str2
} else {
  // str1 comes after str2
}
```

Kopírování textových řetězců

- Řetězce nelze v C kopírovat pomocí operátoru =

```
char str1[] = "hello"; // e.g.  0x7f42
char *str2 = str1; // 0x7f42 stejný řetězec!
str2[0] = 'H';
printf("%s %s", str1, str2); // Hello Hello
```

- Kopírování řetězců pomocí funkce strcpy

```
char str1[] = "hello"; // e.g.  0x7f42
char str2[6];
strcpy (str2, str1);
str2[0] = 'H';
printf ("%s %s", str1, str2); // hello Hello
```

Argumentem strcpy je inicializovaný ukazatel. Tím může být pole, dynamicky alokované pole nebo ukazatel inicializovaný literálem.

Spojování textových řetězců

- Řetězce nelze v C spojovat pomocí operátoru `+`

```
char str1[] = "hello ";
char str2[] = "prpa!";
char *str3 = str1 + str2; // nelze ani zkompilovat!
```

- Spojování řetězců pomocí funkce `strcat`

```
// výsledek se vloží do prvního argumentu strcat
// musí být tedy dostatečně dlouhý
char str1[13] = "hello ";
char str2[] = "prpa!";
// funkce přesune konec str1 na konec výsledku
strcat (str1, str2);
printf ("%s", str1);
```

Části textových řetězců

- Pro získání části textového řetězce lze využít ukazatel

```
char chars[] = "racecar";
char *str1 = chars;
char *str2 = chars + 4;
printf("%s\n", str1); // racecar
printf("%s\n", str2); // car
```

- Pozor na to, že ukazatele ukazují na stejná místa v paměti

```
str2[0] = 'f';
printf("%s\n", str1); // racefar
printf("%s\n", str2); // far
```

Pole textových řetězců

- Pole textových řetězců bude nejméně dvourozměrné

```
char *stringArray[5]; // pole ukazatelů char *s

char *stringArray[] = {
  "my string 1",
  "my string 2",
  "my string 3"
};
printf ("%s\n", stringArray[1]);
```

II. Textové řetězce

Textové řetězce

Funkce main

Funkce `main` a její tvary

- Základní tvar funkce main

```
int main ( int argc , char * argv [] ) {
...  }
```

- Alternativně pak také

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

- Argumenty funkce nejsou nutné

```
int main ( void ) { ... }
```

- Rozšířená funkce o nastavení proměnných prostředí

Unix a MS Windows

```
int main ( int argc , char ** argv , char
 ** envp ) { ... }
```

Přístup k proměnným prostředí funkcí **getenv()** z knihovny **stdlib.h**.

- Rozšířená funkce o specifické parametry Mac OS X

```
int main ( int argc , char ** argv , char
 ** envp , char ** apple ) ;
```

Argumenty funkce `main`

- Základní tvar funkce `main`

```
int main ( int argc , char * argv [])
{
...
}
```

- `argc` – obsahuje počet argumentů programu

  Včetne jména spouštěného programu.

  - Argumenty jsou textové řetězce oddělené mezerou (bílým znakem).
- `argv` – pole ukazatelů na hodnoty typu `char`
  - Pole `argv` má velikost (počet prvku) daný hodnotou `argc`
  - Každý prvek pole `argv[i]` obsahuje adresu, kde je uložen textový řetězec argumentů (tj. typ `char*`)
  - Textový řetězec (argument) je posloupnost znaku (typ `char`) zakončený znakem `'\0'`
  - Alokace paměti pro uložení argumentů (textových retězců) je provedena při spuštení programu

Argumenty programu

- Programu můžeme při spuštění předat argumenty příkazové rádky

```
1  #include <stdio.h>
3  int main(int argc, char *argv[])
4  {
5    printf("Pocet argumentu %i\n", argc);
6    for (int i = 0; i < argc; ++i) {
7      printf("argv[%i] = %s\n", i, argv[i]);
8    }
9    return argc > 0 ? 0 : 1;
10 }
```

- Voláním `return` ve funkci `main()` vracíme z programu návratovou hodnotu, se kterou můžeme dále pracovat

```
$ ./a.out >/dev/null; echo $?
$ 1
$ ./a.out first >/dev/null; echo $?
$ 0
```

Zadání 6. domácího úkolu (HW05)

Téma: Caesarova šifra

- **Motivace:** Práce s polem
- **Cíl:** Dekódování odposlechnuté šifry hrubou silou
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw05`
  - Načtení dvou vstupních textů a tisk dekódované zprávy na výstup
  - Zakódovaný text i (špatně) odposlechnutý text mají stejné délky
  - Nalezení největší shody dekódovaného a odposlechnutého textu na základě hodnoty posunu v Caesarově šifře
- **Termín odevzdání:** **9.11.2019, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

- Pole a ukazatele
- Vícerozměrná pole
- Ukazatelová aritmetika
- Řetězce

- Příště: zásobník, halda, alokace dynamické paměti, ladění

Diskutovaná témata

- Pole a ukazatele
- Vícerozměrná pole
- Ukazatelová aritmetika
- Řetězce

- Příště: zásobník, halda, alokace dynamické paměti, ladění