# Ukazatele, pole a řetězce

Petr Šaloun

katedra informatiky FEI VŠB-TU Ostrava

31. října 2011

# Ukazatele přehled

Téma souvisí se vstupem a výstupem (znakovým, formátovaným, textovým i binárním) a s dynamickými datovými strukturami.
Ukazatel reprezentuje adresu objektu.
Ukazatel obsahuje informaci o datovém typu na adrese.

```
int x, y, *px, *p2x;

px = &x;          /* px nyní ukazuje na x */
*px = 5;          /* jako x = 5;          */
y = *px + 1;      /*      y = x + 1;      */
*px += 1;         /*      x += 1;         */
(*px)++;          /*      x++; závorky nutné */
p2x = px;         /* p2x i px ukazují na x */
*p2x = *p2x + y; /*           x = x + y;  */
```

# Ukazatele a modifikátor **const**

Překladače ISO/ANSI C přísně kontrolují typovost hodnoty i konstantnost ukazatele.

```
int i;
int *pi;      /* pi je neinicializovaný uk. na int */
int * const cp = &i;  /* konstantní ukazatel na int*/
const int ci = 7;     /* celočíselná konstanta  */
const int *pci; /* neinicializovaný ukaz. na int */
const int * const cpc = &ci; /* konst. uk. na konst.*/
```

# Pole

Pole je kolekce prvků stejného typu, mají stejný identifikátor.
Přístup k prvkům pole:
– identifikátor a index;
– (dereferencovaný) ukazatel.
Pole – spojitá oblast operační paměti, první prvek na nejnižší adrese, poslední na nejvyšší adrese.
Pole nemusí uvádět dimenzi.
Pole je v C výhradně jednorozměrné.
Prvkem pole může být pole → použití vícerozměrných polí.

```
typ jméno[rozsah];
```

`typ` určuje typ prvků pole, tj. *bázový typ*,

`jméno` představuje identifikátor pole,

`rozsah` – počet prvků pole, tj. celočíselný konstantní výraz (vyčíslitelný za překladu), indexy 0 až `rozsah-1`.

# definice proměnných typu pole

```
const int N = 10;
int a[N];

a[0], a[1], ..., a[N-1]
```

počet obsazených byte = N * sizeof(int)
počet obsazených byte = sizeof(a)
inicializace pole současně s jeho definicí:

```
                                b[0]  b[1]   b[2]    b[3]  b[4]
static double b[5] = {1.2,  3.4,  -1.2,  123.0,  4.0};
```

Inicializačních hodnot může být méně, než prvků pole, (zbývající) prvky pole zůstanou neinicializovány.

```
int c[] = { 3, 4, 5};
```

Identifikátor pole id je konstantní ukazatel na první prvek pole, tedy id[0].

# Aritmetika ukazatelů

ukazatel ukazuje na hodnotu nějakého typu
aritmetika ukazatelů využívá adresu i velikost položky
*porovnání*
*sčítání* – ukazatel na + celé číslo (kladné i záporné) = ukazatel ukazující o příslušný počet prvků výše, respektive níže
*odčítání* – ukazatel na − ukazatel na = počet položek mezi adresami

```
int i, *pi, a[100];
```

`a[0]` je `&a[0]` ⇔ `a` ⇔ `a + 0`
`a + i` ⇔ `&a[i]`
`*(a+i)` ⇔ `a[i]`

```
pi = a;
pi = a + 49;
(pi - a == 49);
```

`a[i]` ⇔ `*(a+i)` ⇔ `pi[i]` ⇔ `*(pi+i)`.
lze `++pi`, `pi++`
nelze `++a`, `a++`

# Aritmetika ukazatelů – kopírování pole I – cpyarry.c

```
int main()
{int pole1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9},
     pole2[N], dim1;
 dim1 = sizeof(pole1) / sizeof(int);

 print_array(pole1, dim1);
 copy_array1(pole1, pole2, N);
 print_array(pole2, N);
 copy_array2(pole1 + 3, pole2, N);
 print_array(pole2, N);
 return 0;
}
/* vystup:
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6
4       5       6       7       8       9
*/
```

# Aritmetika ukazatelů – kopírování pole II – cpyarry.c

```
const int N = 6;
#include <stdio.h>

void copy_array1(int *a, int *b, int n)
/* a - vstupni pole, b - vystupni pole, n - prvku */
{register int i = 0;
 for (; i < n; i++)     b[i] = a[i];}

void copy_array2(int *a, int *b, int n)
/* a - vstupni pole, b - vystupni pole, n - prvku */
{while (n--> 0)      *b++ = *a++;}

void print_array(int *p, int n)
{puts("");
 while (n--> 0)   printf("\t%d", *p++);
 puts("");
}
```

# Řetězec

(jednorozměrné) pole znaků ukončené znakem `'\0'` – zarážkou.

```
char s[SIZE];
```

- proměnná typu řetězec délky `SIZE`, jednotlivé znaky přístupné pomocí indexů `s[0]` až `s[SIZE-1]`.
- konstantní ukazatel na znak, tj. na první prvek pole `s`, `s[0]`.

`"abc"` řetězec, délka $3+1$ znak (čtyři bajty paměti), konstantní pole čtyř znaků;

`"a"` řetězcová konstanta, délku $1+1$ znak;

`'a'` znaková konstanta (neplést s řetězcem!).

```
char pozdrav[] = "hello";
char pozdrav[] = {'h','e','l','l','o','\0'};

char *ps = "retezec";
"ahoj světe" + 5    odpovídá    "světe".
```

# Vztah mezi řetězcem a polem znaků – str_ptr .c

```c
#include <stdio.h>

int main()
{
 char text[] = "world", *new1 = text, *new2 = text;
 printf("%s\t%s\t%s\n", text, new1, new2);
 new1 = "hello";
 printf("%s\t%s\t%s\n", text, new1, new2);
 printf("%s\n", "hello world" + 2);
 return 0;
}
/*
world    world    world
world    hello    world
llo world
*/
```

# Kopírování řetězců I – str_cpy .c

```
int main() {
 char s1[] = "prvni (1.) retezec",
      s2[] = "druhy (2.) retezec",
      d1[SIZE], d2[SIZE],
      *ps = s2;

 strcpy1(d1, s1);
 strcpy2(d2, s2 + 6);
 ps = s2 + 8;
 printf("s1[]:%s\nd1[]:%s\n", s1, d1);
 printf("s2[]:%s\nd2[]:%s\n *ps:%s\n", s2, d2, ps);
 return 0;
}
/* s1[]:prvni (1.) retezec  s2[]:druhy (2.) retezec
   d1[]:prvni (1.) retezec        d2[]:(2.) retezec
                                  *ps:.) retezec*/
```

# Kopírování řetězců II – str_cpy .c

```c
#include <stdio.h>

const int SIZE = 80;

void strcpy1(char *d, char *s) {
 while ((*d++ = *s++) != 0);
}

void strcpy2(char d[], char s[]) {
 int i = 0;
 while ((d[i] = s[i]) != 0)
    i++;
}
```

# Řetězcové funkce C – `string.h` – I

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

lexikograficky porovnává řetězce, vrací hodnoty

| | | | | |
|---|---|---|---|---|
| $< 0$ | je–li | s1 | $<$ | s2 |
| $= 0$ | | s1 | $=$ | s2 |
| $> 0$ | | s1 | $>$ | s2 |

```
int strncmp(const char *s1, const char *s2,unsigned int n);
```

jako předchozí s tím, že porovnává nejvýše `n` znaků;

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

vrátí počet významných znaků řetězce (bez zarážky);

```
char *strcpy(char *dest, const char *src);
```

nakopíruje `src` do `dest`;

```
char *strncpy(char *dest, const char *src, unsigned int n);
```

jako předchozí, ale nejvýše `n` znaků (je–li jich právě `n`, nepřidá zarážku);

# Řetězcové funkce C – string.h – II

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

s2 přikopíruje za s1;

```
char *strncat(char *s1, const char *s2, unsigned int n);
```

jako předchozí, ale nejvýše n znaků, n se týká délky s2, ne s1;

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

vyhledá první výskyt (zleva) znaku c v řetězci s;

```
char *strrchr(const char *s, int c);
```

vyhledá první výskyt (zprava) znaku c v řetězci s;

```
char *strstr(const char *str, const char *substr);
```

vyhledá první výskyt (zleva) podřetězce substr v řetězci str.

# Vícerozměrná pole (matice)

Jazyk C jednorozměrné pole, prvky libovolného typu *righrarrow* (jednorozměrná pole).
Definice matice (dvourozměrného pole) může vypadat takto:

```
type jmeno[5][7];
```

`typ` určuje datový typ položek pole,

`jmeno` představuje identifikátor pole,

`5][7]` určuje rozsah jednotlivých vektorů na pět řádků a sedm sloupců.

„poslední index se mění nejrychleji" – umístění vícerozměrného pole v paměti

# Inicializace a výpis matice typu $2 \times 3$ – in_mat.c

```c
#include <stdio.h>

int main(void)
{
  int i, j;
  float matice[2][3] = {{1.1, 1.2, 1.3}, {2.1, 2.2, 2.3
  for (i=0; i < 2; i++)
  {
      for (j = 0; j < 3; j++)
         printf("%10.2f", matice[i][j]);
      putchar('\n');
    }
  return 0;
}
```

# 2D pole/matice – číslo dne v roce – K&R – yearday.c I

```
static int day_tab[2][13] =
 {{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

int day_of_year(int year, int month, int day) {
 int i, leap;
 leap = year % 4 == 0 && year % 100 != 0 || year % 400
 for (i = 1; i < month; i++)
   day += day_tab[leap][i];
 return day;
}
```

# 2D pole/matice – číslo dne v roce – K&R – yearday.c II

```
int month_day(int year,int yearday,int *pmonth,int *pd
 int i, leap;
 leap = year % 4 == 0 && year % 100 != 0 || year % 400
 for (i = 1; yearday > day_tab[leap][i]; i++)
   yearday -= day_tab[leap][i];
 *pmonth = i;
 *pday = yearday;
 return i;
}
int main() {
 int year = 1993, month = 11, day = 12, yearday, m, d;
 yearday = day_of_year(year, month, day);
 month_day(year, yearday, &m, &d);
 return 0;
}
```

# Ukazatele na funkce

```
typ *jmeno();
```

funkce vracející ukazatel na typ.

```
typ (*jmeno)();
```

ukazatel na funkci bez argumentů vrací hodnotu zvoleného datového typu
použití ukazatelů na funkci – knihovní funkce `qsort()`, prototyp v `stdlib.h`.

```
void qsort(void *base, size_t nelem, size_t width,
           int (*fcmp)(const void *, const void *));
```

`base` začátek pole, které chceme setřídit;

`nelem` počet prvků, které chceme setřídit (rozsah pole);

`width` počet bajtů, který zabírá jeden prvek;

`fcmp` ukazatel na funkci, provádějící porovnání, ta má jako argumenty dva konstantní ukazatele na právě porovnávané prvky (nutno přetypovat).

# Použití knihovní funkce qsort() – fn_qsrt .c – I

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
const int POCET = 1000000;
const int RND_START = 1234;

int float_sort (const float *a, const float *b) {
 return (*a - *b);      /* <0, ==0, >0  */
}
void test (float *p, unsigned int pocet) {
 int chyba = 0;
 for (; !chyba && --pocet > 0; p++)
   if (*p > *(p+1))
     chyba=1;
 puts ((chyba) ? "\npole neni setrideno\n"
               : "\npole je setrideno\n");
}
```

# Použití knihovní funkce qsort() – fn_qsrt .c – I

```
void vypln(float *p, int pocet) {
 srand(RND_START);
 while (pocet--> 0)
    *p++ = (float) rand();
}
int main (void) {
 static float pole [POCET];
 clock_t start, end;
 vypln (pole, POCET);
 start = clock();
 qsort(pole, POCET, sizeof(*pole),
   (int (*) (const void *, const void *)) float_sort);
 end = clock ();
 printf("Trideni qsort trvalo %fs",(end-start)/CLK_TCK
 test (pole, POCET); getc(stdin);
 return 0;
}
```

# Ukazatele na funkce a jejich použití – ptr_fn01.c – I

```
#include <stdio.h>
#include <process.h>

void fn_1(void); void fn_2(void); void fn_3(void);

typedef void (*menu_fcn) (void);
menu_fcn command[3] = {fn_1, fn_2, fn_3};
void fn_1(void)  {
 puts("funkce cislo 1");
}
void fn_2(void) {
 puts("funkce cislo 2");
}
void fn_3(void) {
 puts("funkce cislo 3\nKONEC");
 exit(0);
}
```

# Ukazatele na funkce a jejich použití – ptr_fn01.c – II

```
void menu(void) {
 int choice;
 do {puts("1\tpolozka\n2\tpolozka\n3\tukonceni\n");
     putchar('>');
     scanf("%d", &choice);
     if (choice >= 1 && choice <= 3)
       command[choice - 1]();
   } while (1);
}

int main(void) {
 menu();
 return 0;
}
```

# Argumenty příkazového řádku

Nutné rozšířit definici fce `main()` o dva argumenty:

```
int main(int argc, char *argv[]);
```

argc První argument, typicky `int argc`, udává *počet argumentů* příkazové řádky (jeden = jméno programu, dva = jméno programu + jeden argument, ...).

argv Druhý argument, typicky `char *argv[]`, představuje *hodnoty argumentů* příkazového řádku. Jeho typ je pochopitelně pole ukazatelů na řetězce, neboť jimi argumenty příkazového řádku skutečně jsou.

# Výpis argumentů příkazového řádku – cmd_ln04.c

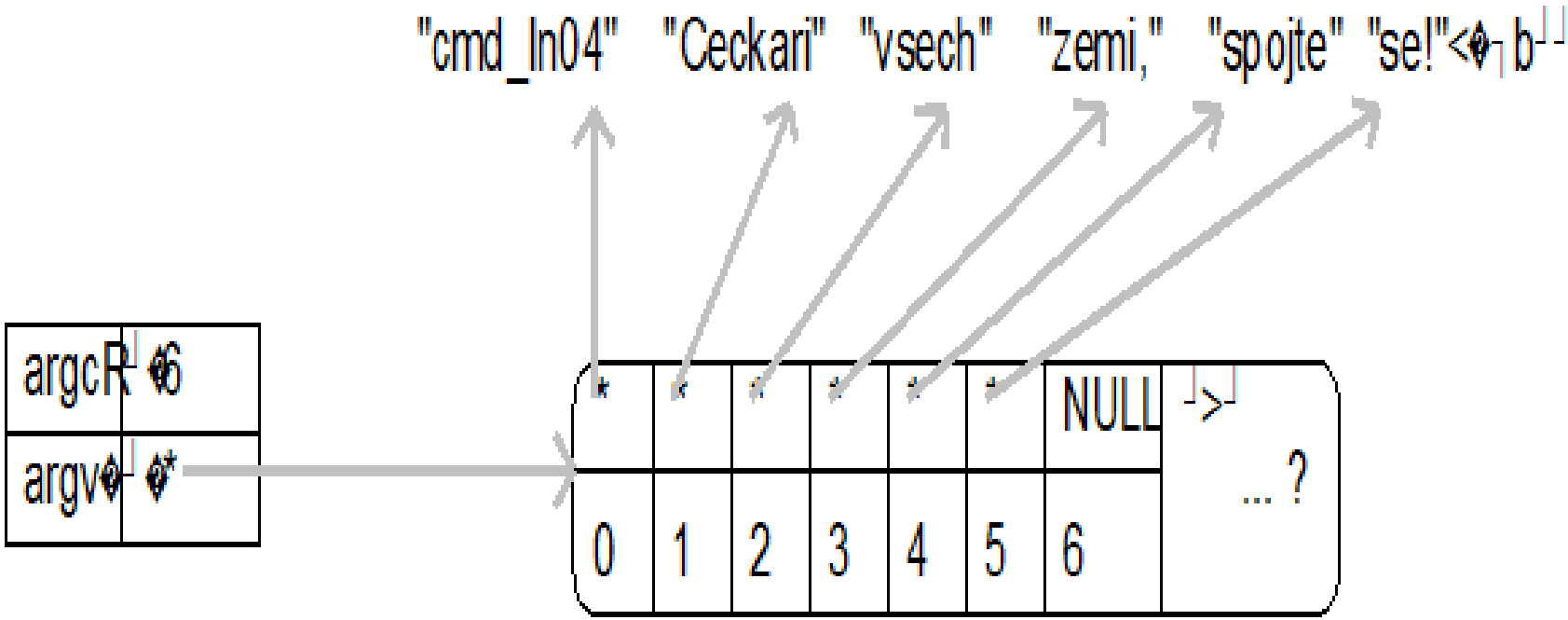
```c
#include <stdio.h>

int main(int argc, char **argv)
{
  while (argc-- > 0)
    printf((argc > 0) ? "%s " : "%s\n", *argv++);
  return 0;
}
```

# Argumenty příkazového řádku

Nechť funkce main a příkazový řádek jsou následující:

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

```
>cmd_ln04 Ceckari vsech zemi, spojte se!
```



# Ukazatele a pole – shrnutí

hlavní zásady pro práci s ukazateli, poli a řetězci:

- nesmíme se odkazovat na neinicializovaný ukazatel,
- je–li `p` ukazatel na nějaký typ, pak `*p` je právě ta hodnota, na kterou `p` ukazuje,
- identifikátor pole je jen konstantní ukazatel na toto pole,
- řetězec je znakové pole ukončené zarážkou,
- využívejme aritmetiky ukazatelů.