Jednorozměrná pole a řetězce

Pavel Čeleda

celeda@liberouter.org

Kurz jazyka C - přednáška č. 6

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**
  - rozsah indexů je 0 až `pocet-1`,

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**
  - rozsah indexů je `0` až `pocet-1`,
  - x[0]=5; /* OK */

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**
  - rozsah indexů je 0 až `pocet-1`,
  - x[0]=5; /* OK */
  - x[10]=1; /* ERROR - chybny pristup k 11 prvku pole */

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**
  - rozsah indexů je `0` až `pocet-1`,
  - x[0]=5; /* OK */
  - x[10]=1; /* ERROR - chybny pristup k 11 prvku pole */
- jazyk C nekontroluje meze polí !!!

Jednorozměrná pole

- pole = datová struktura složená ze stejných prvků,
- dolní mez pole je 0, pole začíná vždy prvkem s indexem 0,
- definice statického pole - **TYP x[pocet];** např. **int x[10];**
  - rozsah indexů je `0` až `pocet-1`,
  - x[0]=5; /* OK */
  - x[10]=1; /* ERROR - chybny pristup k 11 prvku pole */
- jazyk C nekontroluje meze polí !!!
- ```
  #define MAX 10
  int x[MAX], y[MAX * 2], z[MAX + 1];
  ```
  ```

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];
- typedef char RCV_BUF[10];

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];
- typedef char RCV_BUF[10];

```
RCV_BUF buf_uart0, buf_uart1;
buf_uart0[3] = 0;
```

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];
- typedef char RCV_BUF[10];

```
RCV_BUF buf_uart0, buf_uart1;
buf_uart0[3] = 0;
```

- ```
  for(i=0; i<10; i++) /* nulovani pole */
   buf_uart0[i] = 0;
  ```
  ```

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];
- typedef char RCV_BUF[10];

```
RCV_BUF buf_uart0, buf_uart1;
buf_uart0[3] = 0;
```

- ```
  for(i=0; i<10; i++) /* nulovani pole */
   buf_uart0[i] = 0;
  ```
- ```
  for(i=0; i<10; i++) /* vypis pole */
   printf("%d, ", buf_uart0[i]);
  ```

Jednorozměrná pole - statická

- zavedení nového datového typu
  typedef ZNAMY_TYP NOVY_TYP[pocet];
- typedef char RCV_BUF[10];

```
RCV_BUF buf_uart0, buf_uart1;
buf_uart0[3] = 0;
```

- `for(i=0; i<10; i++) /* nulovani pole */`
  `buf_uart0[i] = 0;`
- `for(i=0; i<10; i++) /* vypis pole */`
  `printf("%d, ", buf_uart0[i]);`
- `for(i=0; i<10; i++) /* kopirovani pole */`
  `buf_uart0[i] = buf_uart1[i];`

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] =` *`bázová adresa`* `x + i * sizeof(typ);`

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] = `*`bázová adresa`*` x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] =` *`bázová adresa`* `x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`
- `int a_var[10];` definice statického pole

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] =` *`bázová adresa`* `x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`
- `int a_var[10];` definice statického pole
- `int *p_i;` definice pointer
  `p_i = (int *) malloc(10*sizeof(int));`

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] = `*`bázová adresa`*` x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`
- `int a_var[10];` definice statického pole
- `int *p_i;` definice pointer
  `p_i = (int *) malloc(10*sizeof(int));`
- `p_i[0] == *p_i`
  `p_i[1] == *(p_i+1)`
  `p_i[2] == *(p_i+2)`
  `p_i[3] == *(p_i+3)`

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] =` *`bázová adresa`* `x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`
- `int a_var[10];` definice statického pole
- `int *p_i;` definice pointer
  `p_i = (int *) malloc(10*sizeof(int));`
- `p_i[0] == *p_i`
  `p_i[1] == *(p_i+1)`
  `p_i[2] == *(p_i+2)`
  `p_i[3] == *(p_i+3)`
- práce se statickým a dynamickým polem je stejná,

Pole a pointery - dynamická pole

- výpočet adresy prvku v poli
  `&x[i] =` *`bázová adresa`* `x + i * sizeof(typ);`
- indexování pole `x[i]` $==$ `*(x+i)`
- `int a_var[10];` definice statického pole
- `int *p_i;` definice pointer
  `p_i = (int *) malloc(10*sizeof(int));`
- `p_i[0] == *p_i`
  `p_i[1] == *(p_i+1)`
  `p_i[2] == *(p_i+2)`
  `p_i[3] == *(p_i+3)`
- práce se statickým a dynamickým polem je stejná,
- statické a dynamické pole se pouze liší způsobem definice a alokace paměti.

Velikost pole

- ```
  int x[10], *p_x;
  p_x = (int *) malloc(10 * sizeof(int));
  ```
  ```

Velikost pole

- `int x[10], *p_x;`
  `p_x = (int *) malloc(10 * sizeof(int));`
- `sizeof(x) == 10 * sizeof(int)` tedy např. 20

Velikost pole

- `int x[10], *p_x;`
  `p_x = (int *) malloc(10 * sizeof(int));`
- `sizeof(x) == 10 * sizeof(int)` tedy např. 20
- `sizeof(p_x) == 10 * sizeof(int *)`
  tedy např. 4 - velikost adresy

Velikost pole

- `int x[10], *p_x;`
  `p_x = (int *) malloc(10 * sizeof(int));`
- `sizeof(x) == 10 * sizeof(int)` tedy např. 20
- `sizeof(p_x) == 10 * sizeof(int *)`
  tedy např. 4 - velikost adresy
- pole měnící svoji velikost $\longrightarrow$ vytvoření nového pole a překopírování obsahu starého pole a následné zrušení původního pole,

Velikost pole

- `int x[10], *p_x;`
  `p_x = (int *) malloc(10 * sizeof(int));`
- `sizeof(x) == 10 * sizeof(int)` tedy např. 20
- `sizeof(p_x) == 10 * sizeof(int *)`
  tedy např. 4 - velikost adresy
- pole měnící svoji velikost $\longrightarrow$ vytvoření nového pole a překopírování obsahu starého pole a následné zrušení původního pole,
- funkce `realloc()` z `stdlib.h`
  `void *realloc(void *ptr, size_t size);`

Pole jako parametry funkcí

- parametr pole je do funkce předáván odkazem, předá se adresa začátku pole,

Pole jako parametry funkcí

- parametr pole je do funkce předáván odkazem, předá se adresa začátku pole,
- `double maxim(double pole[])`
  `double maxim(double *pole)`

Pole jako parametry funkcí

- parametr pole je do funkce předáván odkazem, předá se adresa začátku pole,
- `double maxim(double pole[])`
  `double maxim(double *pole)`
- schází informace o velikosti pole
  → `double maxim(double pole[], int pocet)`

Pole jako parametry funkcí

- parametr pole je do funkce předáván odkazem, předá se adresa začátku pole,
- `double maxim(double pole[])`
  `double maxim(double *pole)`
- schází informace o velikosti pole
  → `double maxim(double pole[], int pocet)`
-
```
void vypln(double pole[])
{
 int i;
 for(i=0;i<10;i++)
 pole[i]=i;
}
```

Řetězce

- speciální typ jednorozměrného pole,

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',
- statický řetězec `-char s_stat[10];`

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',
- statický řetězec -`char s_stat[10];`
- dynamický řetězec - `char *s_dyn;`
  `s_dyn = (char *) malloc (10);`

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',
- statický řetězec -`char s_stat[10];`
- dynamický řetězec - `char *s_dyn;`
  `s_dyn = (char *) malloc (10);`
- `char s1[10] = "ahoj";` *definice s inicializací*

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',
- statický řetězec -`char s_stat[10];`
- dynamický řetězec - `char *s_dyn;`
  `s_dyn = (char *) malloc (10);`
- `char s1[10] = "ahoj";` *definice s inicializací*
- `char s2[] = "ahoj";` *definice s inicializací bez udání délky řetězce*

Řetězce

- speciální typ jednorozměrného pole,
- je složen vždy z prvků typu **char**,
- řetězec je vždy ukončen znakem '\0',
- při alokování paměti musíme alokovat i místo pro '\0',
- řetězec nemající '\0' je neukončen a za řetězec se bere vše do výskytu '\0',
- statický řetězec -`char s_stat[10];`
- dynamický řetězec - `char *s_dyn;`
  `s_dyn = (char *) malloc (10);`
- `char s1[10] = "ahoj";` *definice s inicializací*
- `char s2[] = "ahoj";` *definice s inicializací bez udání délky řetězce*
- `char *s3 = "ahoj";`

Řetězce

- ```
  char str[10];
  str = "ahoj"; /* nelze */
  ```
  ```

Řetězce

- ```
  char str[10];
  str = "ahoj"; /* nelze */
  ```
- ```
  char *str;
   str = (char *) malloc(10);
   strcpy(str, "ahoj"); /* ok */
  ```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

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

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{









}
```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{
   char str[11];




}
```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{
   char str[11];

   printf("Zadej retezec: ");



}
```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{
   char str[11];

   printf("Zadej retezec: ");
   scanf("%s", str);


}
```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{
   char str[11];

   printf("Zadej retezec: ");
   scanf("%s", str);
   printf("Retezec je %s", str);

}
```

Práce s řetězcem

- čtení řetězce z klávesnice - `scanf("%s", s1);`
- tisk řetězce na obrazovce - `printf("%s", s1);`

```
#include <stdio.h>

int main(void)
{
   char str[11];

   printf("Zadej retezec: ");
   scanf("%s", str);
   printf("Retezec je %s", str);
   return 0;
}
```

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`
- kopírování řetězce - `char *strcpy(char *dest, char *src);`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`
- kopírování řetězce - `char *strcpy(char *dest, char *src);`
- spojení řetězců - `char *strcat(char *dest, char *src);`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`
- kopírování řetězce - `char *strcpy(char *dest, char *src);`
- spojení řetězců - `char *strcat(char *dest, char *src);`
- nalezení znaku v řetězci - `char *strchr(char *s, int c);`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`
- kopírování řetězce - `char *strcpy(char *dest, char *src);`
- spojení řetězců - `char *strcat(char *dest, char *src);`
- nalezení znaku v řetězci - `char *strchr(char *s, int c);`
- porovnání dvou řetězců - `int strcmp(char *s1,char *s2);`

Standardní funkce pro práci s řetězci

- hlavičkový soubor `string.h`
- délka řetězce - `int strlen(char *s);`
- kopírování řetězce - `char *strcpy(char *dest, char *src);`
- spojení řetězců - `char *strcat(char *dest, char *src);`
- nalezení znaku v řetězci - `char *strchr(char *s, int c);`
- porovnání dvou řetězců - `int strcmp(char *s1,char *s2);`
- nalezení podřetězce v řetězci - `char *strstr(char *s1,char *s2);`

Převody řetězců na číslo

- standardní vstup - `int scanf(const char *format, ...);`

Převody řetězců na číslo

- standardní vstup - `int scanf(const char *format, ...);`
- standarní výstup - `int printf(const char *format, ...);`

Převody řetězců na číslo

- standardní vstup - `int scanf(const char *format, ...);`
- standardní výstup - `int printf(const char *format, ...);`
- ASCII to integer - `int atoi(const char *nptr);`

Převody řetězců na číslo

- standardní vstup - `int scanf(const char *format, ...);`
- standarní výstup - `int printf(const char *format, ...);`
- ASCII to integer - `int atoi(const char *nptr);`
- ASCII to long - `long atol(const char *nptr);`

Převody řetězců na číslo

- standardní vstup - `int scanf(const char *format, ...);`
- standarní výstup - `int printf(const char *format, ...);`
- ASCII to integer - `int atoi(const char *nptr);`
- ASCII to long - `long atol(const char *nptr);`
- ASCII to float - `double atof(const char *nptr);`

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

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

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{









}
```

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{
   int i;
   char jmeno[50];




}
```

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{
   int i;
   char jmeno[50];

   for(i=0; i<20; i++) {




}
```

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{
   int i;
   char jmeno[50];

   for(i=0; i<20; i++) {
      sprintf(jmeno, "obr%02d.jpg", i);



}
```

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{
   int i;
   char jmeno[50];

   for(i=0; i<20; i++) {
      sprintf(jmeno, "obr%02d.jpg", i);
      printf("%s\n", jmeno);
   }

}
```

Formátované čtení a zápis z a do řetězce

- vstup z řetězce - `int sscanf(const char *str, const char *format, ...);`
- výstup do řetězce - `int sprintf(char *str, const char *format, ...);`

```
#include <stdio.h>

int main(void)
{
   int i;
   char jmeno[50];

   for(i=0; i<20; i++) {
      sprintf(jmeno, "obr%02d.jpg", i);
      printf("%s\n", jmeno);
   }
   return 0;
}
```