Obsah

25 Práce s polem, řadící algoritmy

25.1 Práce s polem

25.1.1 Inicializace pole

- v C vždy nutno specifikovat délku
  - specifikováním počtu – `int nums[5];`
  - specifikace listu dat – `int nums[] = {1,2,3,4,5};`
  - specifikace obou – `int nums[5] = {1,2,3,4,5};`

25.1.2 Získání hodnoty z pole

- specifikace jména pole a indexu
  - indexování pole začíná na 0
- `int value = nums[3];`
- `int i = 0; int value = nums[i];`

25.1.3 Zápis do pole

- hodnota buňky rovna zapisované hodnotě
- `nums[4] = 6;`
- `int value = 7; int index = 2; nums[index]=value`

25.1.4 Procházení pole

- využití `for` smyčky

25.2 Vícerozměrná pole

- pole o více rozměrech, pole polí
- každý prvek pole je další pole
- `int matrix[height][width];`
- stejné čtení a zápis dat jako při jedné dimenzi

25.3 Řadící (sorting) algoritmy

- algoritmy zajišťující uspořádání dané sady (pole, seznam…)
- nejčastější numerické a abecední řazení
- v dnešní době důležitá efektivnost řadících algoritmů na velkých datech
- podmínky výstupu
  - výstup je monotónní – každý prvek není větší/menší než den předešlý (dle zadaných pravidel)

```c
#include <stdio.h>

int grades[] = {1,3,4,2,1,3,2,2,3,1};

int main() {
  int arrayLength = sizeof(grades)/sizeof(grades[0]);
  int sum = 0, min = 5, max = 1;

  for (int i = 0; i < arrayLength; i++) {
    int grade = grades[i];
    sum += grade;
    if (grade < min) min = grade;
    if (grade > max) max = grade;
  }

  printf("Average grade: %.2f\n", (float)sum/arrayLength);
  printf("Worst grade: %d\n", max);
  printf("Best grade: %d\n", min);
  return 0;
}
```

Kód 1: Příklad procházení pole

```c
/* Usual declaration */
int abc[2][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8}
};
/* Valid declaration*/
int abc[2][2] = {1, 2, 3, 4, 5, 6, 7, 8};
/* Valid declaration*/
int abc[][2] = {1, 2, 3, 4, 5, 6, 7, 8};
/* Invalid declaration - you must specify second dimension*/
int abc[][] = {1, 2, 3, 4, 5, 6, 7, 8};
/* Invalid because of the same reason  mentioned above*/
int abc[2][] = {1, 2, 3, 4, 5, 6, 7, 8};
```

Kód 2: Inicializace vícerozměrného pole

	Druhý index j			
První index i	`matrix[0][0]`	`matrix[0][1]`	`matrix[0][2]`	`matrix[0][3]`
	`matrix[1][0]`	`matrix[1][1]`	`matrix[1][2]`	`matrix[1][3]`
	`matrix[2][0]`	`matrix[2][1]`	`matrix[2][2]`	`matrix[2][3]`
	`matrix[3][0]`	`matrix[3][1]`	`matrix[3][2]`	`matrix[3][3]`

Tab. 25.1: Zobrazení 2D pole

  - výstup je permutace – jsou zachovány všechny původní prvky
- pro optimální rychlost data uložena ve strukturách s random přístupem, ne sekvenčním

25.3.1 Klasifikace algoritmů

- výpočetní složitost
  - nejlepší, nejhorší, průměrné
  - typicky dobrá složitost $O(n \log n)$, parallel sort $O(\log^2 n)$, špatný $O(n^2)$
  - optimální paralelní složitost $O(\log n)$
- využití paměti
  - některé algoritmy řadí v místě – $O(1)$
- rekurzivita – rekurzivní vs iterativní algoritmy, některé obojí (merge sort)
- stabilita – při souhlasných hodnotách je zachováno vzájemné pořadí udané vstupem
- metoda – vkládání, výměna, selekce, spojování…
  - výměnné – bubble sort, quick sort
  - selekce – cycle sort, heap sort
- sériový vs paralelní
- adaptivní – zda-li předtřídění ovlivní rychlost algoritmu
- online – sort schopný třídit stálý tok dat, např. insertion sort

Jméno	Český název	Nejlepší	Průměrné	Nejhorší	Paměť	Stabilní	Metoda
Quicksort	rychlé řazení	$n \log n$	$n \log n$	n^2	$\log n$	Ne	záměna
Merge sort	řazení slučováním	$n \log n$	$n \log n$	$n \log n$	n	Ano	slučování
Heapsort	řazení haldou	$n \log n$	$n \log n$	$n \log n$	1	Ne	výběr
Insertion sort	řazení vkládáním	n	n^2	n^2	1	Ano	vkládání
Selection sort	řazení výběrem	n^2	n^2	n^2	1	Ne	výběr
Bubble sort	bublinkové řazení	n	n^2	n^2	1	Ano	záměna

Tab. 25.2: Porovnání řadících algoritmů

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

// Swap pointers
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// Function to implement bubble sort
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n-1; i++) {
    bool swapped = false;
    for (int j = 0; j < n-i-1; j++)
      if (arr[j] > arr[j+1]) {
        swap(&arr[j], &arr[j+1]);
        swapped = true;
      }
    if (!swapped) break; // Break if already sorted
  }
}

// Function to print an array
void printArray(int arr[], int size)
{
  for (int i=0; i < size; i++)
    printf("%d ", arr[i]);
  printf("\n");
}

// Driver program to test above functions
int main() {
  int arr[] = {11, 64, 34, 25, 12, 22, 90};
  int n = sizeof(arr)/sizeof(arr[0]);
  bubbleSort(arr, n);
  printf("Sorted array: \n");
  printArray(arr, n);
  return 0;
}
```

Kód 3: Implementace bubble sortu v C