Informatics II

Tutorial Session 2

Wednesday, 2nd of March 2022

Discussion of Exercise 1,
Introduction to C, Basic Sorting

14.00 – 15.45

BIN 0.B.06

Agenda

- Review of Exercise 0
- Review of Exercise 1
- (Preview on Exercise 2)
- Summary and Wrap-Up

Discussion of Exercise 0

- Task 2: Second Largest Integer

Exercise 0, Task 2: Second Largest Integer

– What would be other approaches to solve this task?

Discussion of Exercise 1

- Task 1: True/False Questions
- Task 2: Play with Vowels
- Task 3: Even-Odd Selection Sort
- Task 4: Find the Gap

Exercise 1, Task 1: True/False Questions

a) «The bubble sort algorithm can be implemented using two nested while loops.» ✓ true

→ `for` loops and `while` loops are interchangeable

b) «The insertion sort algorithm can be implemented using two nested for loops.» ✓ true

→ `for` loops and `while` loops are interchangeable

c) «Given the same input, all three sorting algorithms always need the same number of comparisons.» ✗ false

→ all algorithms will need a different number of comparisons in general (cf. lecture slides)

d) «All three sorting algorithms only compare two adjacent elements in an array.» ✗ false

→ counter example: Both selection sort and insertion sort in general will compare elements at positions which are not next to eachother

Exercise 1, Task 1: Conversion of `for` Loops Into `while` Loops

A `for` loop can always be converted into a `while` loop and vice-versa.

Disregarding special cases (e.g. break, continue, return within the loop), the conversion from a `for` loop into a `while` loop can be done as follows:

- put the initialization before the header of the `while` loop,
- put the condition in the header of the `while` loop, and
- use the increment as the last statement within the body of the `while` loop.

```
int i;
for (i = 0; i < n; i = i + 1) {
    /* loop body */
}
```

```
int i = 0;
while (i < n) {
    /* loop body */
    i = i + 1;
}
```

Exercise 1, Task 1: Bubble Sort: For – For vs. While – While

Bubble sort with two nested `for` loops:

```
1  void bubble_sort(int A[], int n) {
2     int i;
3     for (i = n - 1; i >= 1; i--) {
4        int j;
5        for (j = 1; j <= i; j++) {
6           if (A[j] < A[j - 1]) {
7              int temp = A[j];
8              A[j] = A[j - 1];
9              A[j - 1] = temp;
10          }
11       }
12    }
13 }
```

Bubble sort with two nested `while` loops:

```
1  void bubble_sort(int A[], int n) {
2     int i = n - 1;
3     while (i >= 1) {
4        int j = 1;
5        while (j >= 1) {
6           if (A[j] < A[j - 1]) {
7              int temp = A[j];
8              A[j] = A[j - 1];
9              A[j - 1] = temp;
10          }
11          j++;
12       }
13       i--;
14    }
15 }
```

Comprehension Question: Advising Novice Programmers About Loops

Assume you were tasked to tutor young novice programmers aged 15 years old and one of them asks you the following question:

«If for loops and while loops are interchangeable why do both exist and when do I need to use which one of the two?»

What would be your answer?

A `while` loop usually feels more natural to use when there is no way to know how many iterations will be done and the number of iterations might be different depending on the input, for example when searching in an array of integers for the first occurrence of two identical values next to each other.

A `for` loop usually feels more naturally do use when the number of iterations is fixed and clear from the beginning, for example when the sum of the elements of an array of integers is to be calculated. (Many more modern programming languages have a special kind of for loop `for` this purpose: `foreach`).

Exercise 1, Task 2: Play with Vowels

- **2.1:** Implement the C function `int count_vowels(char A[])` that returns the number of vowels for a given string `A`.
- **2.2:** Given a string A[1..n], the B-Sprache string BS of A is generated as follows: We traverse the string A from the first element of A to the last one, one after another. If A[i], 1 ≤ i ≤ n, is not a vowel, A[i] is copied to the BS, otherwise, three consecutive letters: A[i], 'b' and A[i] are copied to BS. Implement the C function `void BS(char A[])` that prints the B-Sprache string of A.

Exercise 1, Task 2: High-Level Sketch of Algorithm

The algorithm to solve the given task can be split into the following steps:

(1) Retrieving the input string from the user

(2) Count the number of vowels in the input string

  (1) iterate over string

  (2) check whether current character is a letter, increase counter if this is the case

  (3) return counter

(3) Calculate length of B-Sprache string and allocate respective memory

(4) Constuct B-Sprache string

Exercise 1, Task 2: Play with Vowels: Check for Vowels

```
int is_vowel(char letter) {
     if (letter == 'A' || letter == 'E' || letter == 'I' || letter =='O' || letter == 'U' ||
         letter == 'a' || letter == 'e' || letter == 'i' || letter =='o' || letter == 'u') {
            return 1;
     }
     else {
            return 0;
     }
}
```

Exercise 1, Task 2: Play with Vowels: Count Vowels

```
int count_vowels(char A[]) {
     int count_vowels = 0;
     int pos = 0;
     while (A[pos] != '\0') {
          if (is_vowel(A[pos])) {
               count_vowels++;
          }
          pos++;
     }
     return count_vowels;
}
```

Exercise 1, Task 2: Play with Vowels: B-Sprache

```
void BS(char A[]) {
     int count_v = count_vowels(A);
     const int new_len = get_length(A) + 2 * count_v;
     char bsprache[new_len + 1];
```

Exercise 1, Task 2: Play with Vowels: B-Sprache

```
    int pos_input = 0;
    int pos_bsprache = 0;
    while (A[pos_input] != '\0') {
        if (is_vowel(A[pos_input])) {
            bsprache[pos_bsprache] = A[pos_input];
            bsprache[pos_bsprache + 1] = 'b';
            bsprache[pos_bsprache + 2] = A[pos_input];
            pos_bsprache = pos_bsprache + 3;
        }
        else {
            bsprache[pos_bsprache] = A[pos_input];
            pos_bsprache = pos_bsprache + 1;
        }
        pos_input++;
    }
    bsprache[new_len] = '\0';
    printf("B-Sprache: %s\n", bsprache);
}
```

Exercise 1, Task 3: Even-Odd Selection Sort

Given an array *A[1..n]* with n integers, the C function `void even_odd_selection_sort(int A[], int n)` prints the following elements: a sorted array *E* with the even numbers in *A* and a sorted array *O* with the odd numbers in *A*. Implement the C function `void even_odd_selection_sort(int A[], int n)`, using the selection sort as the sorting algorithm.

```c
void even_odd_selection_sort(int A[], int n) {
	int even_array[n], odd_array[n];
	int odd_count = 0, even_count = 0;
	selection_sort(A, n);
	for (int i = 0; i < n; i++) {
		if (A[i] % 2 == 0) {
			even_array[even_count] = A[i];
			even_count++;
		}
		else {
			odd_array[odd_count] = A[i];
			odd_count++;
		}
	}
	printf("Sorted even numbers: ");
	for (int i = 0; i < even_count; i++) {
		printf("%d ", even_array[i]);
	}
	printf("\nSorted odd numbers: ");
	for (int i = 0; i < odd_count; i++) {
		printf("%d ", odd_array[i]);
	}
}
```

Exercise 1, Task 4: Find the Gap

You are employed as a system administrator for the London Underground. You are responsible to manage the file servers of the company. Several processes write files the disks managed by the file server. Every time a file is accessed or modified, the file system will store a timestamp when this operation was performed. The timestamp is an integer number counting seconds from a particular epoch.

Given an array of integers, write a C program that returns the biggest gap, when there hasn't been any access or modify operations on the file system for the longest period of time. Note that the script retrieves the timestamps in no particular order and that there might be duplicate timestamps. For example, in the array A = [9, 1, 4, 9, 5, 3, 9] has two gaps and the longest gap is between the timestamps 5 and 9 and has length 4.

```c
int main() {
    printf("Values of A separated by spaces (non-number to stop): ");
    int timestamps[MAX_LENGTH];
    int pos = 0;
    while (scanf("%d", &timestamps[pos]) == 1) {
        pos++;
    }
    insertion_sort(timestamps, pos);
    int longest_gap = 0;
    for (int i = 1; i < pos; i++) {
        int gap = timestamps[i] - timestamps[i - 1];
        if (gap > 1 && gap > longest_gap) {
            longest_gap = gap;
        }
    }
    printf("Longest gap: %d\n", longest_gap);
    printf("Oldest timestamp: %d\n", timestamps[0]);
    printf("Most recent timestamp: %d\n", timestamps[pos - 1]);
    return 0;
}
```

Additional Training Exercise: RLE

Write a simple run-length encoding (RLE) function in C which operates on a given string.

For example, for the input "AAAAABBBBCCAAABDDDDDCC", it should output "A5B4C2A3BD5C2".

Preview on Exercise 2

- Task 1: Short Questions
- Task 2: Second Smallest Element (Recursively)
- Task 3: Blinking Light
- Task 4: Fractal Circles

Wrap-Up

- Summary
- Outlook
- Questions

Outlook on the Next Lab Session

Next tutorial: Wednesday, 08.03.2022, 14.00 h, BIN 0.B.06

Topics:

- Review of Exercise 2
- (Preview to Exercise 3)
- Recursion
- …
- … (your wishes)

Questions?

Thank you for your attention.