Informatics II

Tutorial Session 2

Thursday, 4th of March 2021

Discussion of Exercise 1,
Basic Sorting, Recursion

09.00 – 11.45

Online on Zoom (Meeting ID: 970 3019 1915)

Agenda

- Intro: Expectations and Wishes
- Supplement to the Introduction to the C Language
- Review of Exercise 1
- Recitation of Recursion and / or Basic Sorting
- Preview on Exercise 2
- Summary and Wrap-Up

Introduction

- What shall we do today?
- What problems did you encounter while solving Exercise 1?
- Are there any questions regarding the “Introduction to C” presentation available on OLAT?

Your Expectations

https://www.klicker.uzh.ch/algodat

Supplement to the Introduction to the C Language

- Pass by value, pass by reference
- Passing Arrays to Functions

Call By Value and Call By Reference (Pass By Value and Pass By Reference)

Consider the following code snippet: What will be the output?

```c
#include <stdio.h>

int foo1(int a) {
    a = a % 3 + 5;
    return a;
}

int main() {
    int x = 32;
    int y = foo1(x);
    printf("x = %d\n", x);
    printf("y = %d", y);
    return 0;
}
```

Call By Value and Call By Reference

Consider the following code snippet: What will be the output?

```c
#include <stdio.h>

int foo2(int a[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        a[i] = a[i] % 3 + 5;
        sum += a[i];
    }
    return sum;
}

int main() {
    int x[] = {32, 33, 34};
    int y = foo2(x, 3);
    printf("x = %d, %d, %d\n", x[0], x[1], x[2]);
    printf("y = %d", y);
    return 0;
}
```

Call By Value and Call By Reference

(from https://www.mathwarehouse.com/programming/passing-by-value-vs-by-reference-visual-explanation.php)

Call By Value and Call By Reference

The following implementation of a swap function is wrong and does not perform what one might expect:

```
1  #include <stdio.h>
2
3  void wrongSwap (char a, char b) {
4          int temp = a;
5          a = b;
6          b = temp;
7  }
8
9  int main() {
10         int a = 19;
11         int b = 91;
12         wrongSwap(a, b);
13         printf("%d", a);
14 }
```

Why does the adjacent code not work as intended?

Passing an Array to a Function

- Arrays are passed to functions always by reference (you will understand the reason for this later).
- This means that an array which is passed to a function will also be modified in the scope of the callee. Therefore, we would have to create a copy of the original version is still required.
- Since there is no way to determine the length of an array passed to a function from within that function, we always have to also pass the length of the array as an additional parameter to the function. (An exception to this are strings because the length can be determined there by help of the null terminator (i.e. '\0').
- Note that the return type of a function cannot be an array, e.g. something along the lines of `int[] myFun()` is *not* possible. Thus, you cannot return arrays as such from functions in C. (We will see later how this is dealt with.)

Discussion of Exercise 1

- Task 1: Reverse String
- Task 2: Perfect Square Number
- Task 3: Matrix Multiplication
- Task 4: Selection Sort in Ascending and Descending Order

Initial Remarks on the Discussion of Exercise Tasks

- I will be operating on the assumption that you did at least try to solve the exercise tasks yourself.
- I'm also going to presume that you did look at the sample solution.
- I do *not* presuppose that you all managed to solve the exercises.
- I do *not* think it makes sense to just read the code to you. Instead, I'll try to explain why the code looks as does and how it also might look differently and why.

- Therefore: We will first solve a similar task together. Ask any questions while we are solving the additional task. Hopefully, all questions regarding the task from the exercise sheet will be solved when we're finished with the the discussion of the additional task. If not: I will be going through the task from the exercise sheet rather quickly and you might then ask any remaining questions.

Task 1 Prep Problem: 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".

Task 1 Solution

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

(1) Retrieving the input string from the user

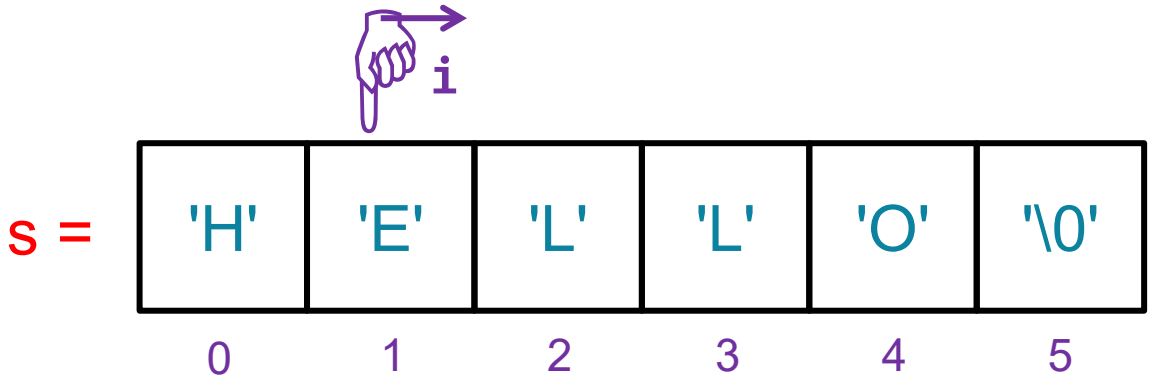
(2) Find out length of input string

(3) Reverse string

(4) Print reversed string

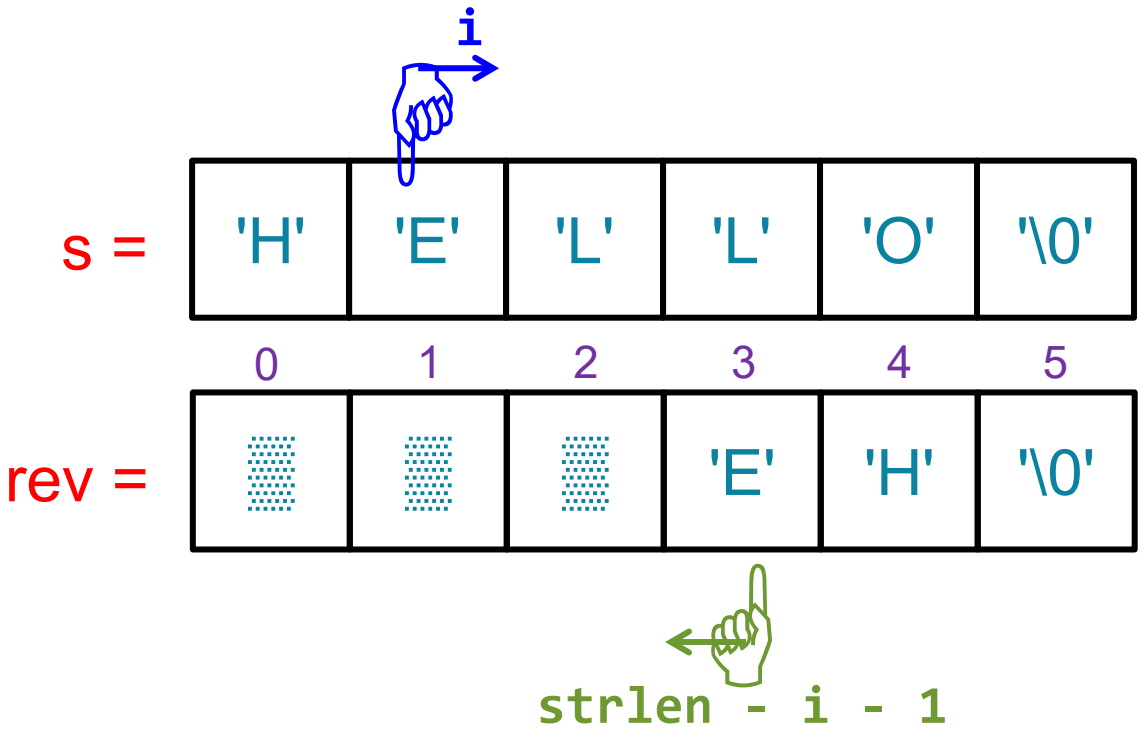
Task 1 Solution: Find Out Length of Input String

```
int strLength(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        i++;
    }
    return i;
}
```



i	*s[i]*	*s[i] == '\n'*
0	'H'	false
1	'E'	false
2	'L'	false
3	'L'	false
4	'O'	false
5	'\n'	true

Task 1 Solution: Reverse the String




```
void reverse(char s[]) {
    int strlen = strLength(s);
    int i = 0;
    char reversedString[strlen + 1];
    while (s[i] != '\0') {
        reversedString[strlen - i - 1] = s[i];
        i++;
    }
    reversedString[strlen] = '\0';
    printf("Result string: %s\n", reversedString);
}
```

Task 1 Solution

```c
#include <stdio.h>
const int MAX_LENGTH = 1000;

int strLength(char s[]) {
  int i = 0;
  while (s[i] != '\0') {
    i++;
  }
  return i;
}
```

```c
void reverse(char s[]) {
  int strlen = strLength(s);
  int i = 0;
  char reversedString[strlen+1];
  while (s[i] != '\0') {
    reversedString[strlen - i - 1] = s[i];
    i++;
  }
  reversedString[strlen] = '\0';
  printf("Result string: %s\n", reversedString);
}

int main() {
  char s[MAX_LENGTH + 1];
  int length;

  printf("Please enter a string: ");
  scanf("%[^\n]s", s);
  reverse(s);
}
```

Task 1: Solution in Pseudocode

Algo: REVERSESTRING(S)

```
strlen ← 0;
while s[i] ≠ '\0' do
    strlen = strlen + 1
    i = i + 1
reversed ← char[strlen + 1]
while s[i] ≠ '\0' do
    reversed[strlen − i − 1] ← s[i]
    i = i + 1
reversed[strlen] =' \0'
return reversed
```

Task 1: Solution: Remarks

- The condition of the `while` loops could be changed from `while(s[i] != '\0')` to `while(s[i])` because the ASCII code of `'\0'` is zero which is treated as the Boolean value `false` when evaluating the condition (and all other ASCII codes of characters are different from zero and thus correspond to the Boolean value `true`).
- One could also reverse the string in-place (i.e. using the already existing array which stores the user input) without allocating a new one. The array could also be just printed in reverse order without actually saving it to memory (the task description is not entirely clear regarding the requirements).
- There are several other valid ways how the loop for the string reversal could be constructed (e.g. for loop instead of while loop, iterating from both sides for in-place reversal etc. pp.).
- It is *not* possible that the return value of the reverse function would be changed to a string because strings are represented as character arrays in C and the return type of a function cannot be an array.
- If it hadn't been prohibited in the task description, we usually would apply the strlen function from the cstring library to determine the length of a string.

Task 2 Prep Problem: Advising Novice Programmers About Loops

Assume you were tasked to tutor young novice programmers aged 15 years old. One of them asks you when one needs to use a for loop and when one should use a while loop. What is your answer?

Task 2 Solution

Solution function:

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

bool isPerfectSquare(int num) {
  int i = 0;
  while (i * i <= num) {
    if (i * i == num) {
      return true;
    }
    i++;
  }
  return false;
}
```

Test driver:

```c
int main() {
  bool perfectSquareCheck;

  int num;

  printf("Please enter an integer to check if it is perfect square number: ");
  scanf("%d", &num);
  perfectSquareCheck = isPerfectSquare(num);

  if (perfectSquareCheck) {
    printf("Result string: %s\n", "TRUE");
  } else {
    printf("Result string: %s\n", "FALSE");
  }
  return 0;
}
```

Task 2: Comparison of the Two Versions of the Algorithm

«Naïve»:

Algo: isPerfectSquare(n)

$i \leftarrow 0$
while $i <= n$ **do**
  if $i^2 == n$ **then**
    return $true$
  $i = i + 1$
return $false$;

«Improved»:

Algo: isPerfectSquare(n)

$i \leftarrow 0$
while $i^2 \leq n$ **do**
  if $i^2 == n$ **then**
    print "$TRUE$"
  $i = i + 1$
print "$FALSE$";

Task 2: Comprehension Question

– Is the «improved» version of the function always faster than the «naïve» version or are there cases where actually the opposite is true?

Task 2: Linear Search vs. Binary Search

Task 3 Prep Problem: Cross of Chars

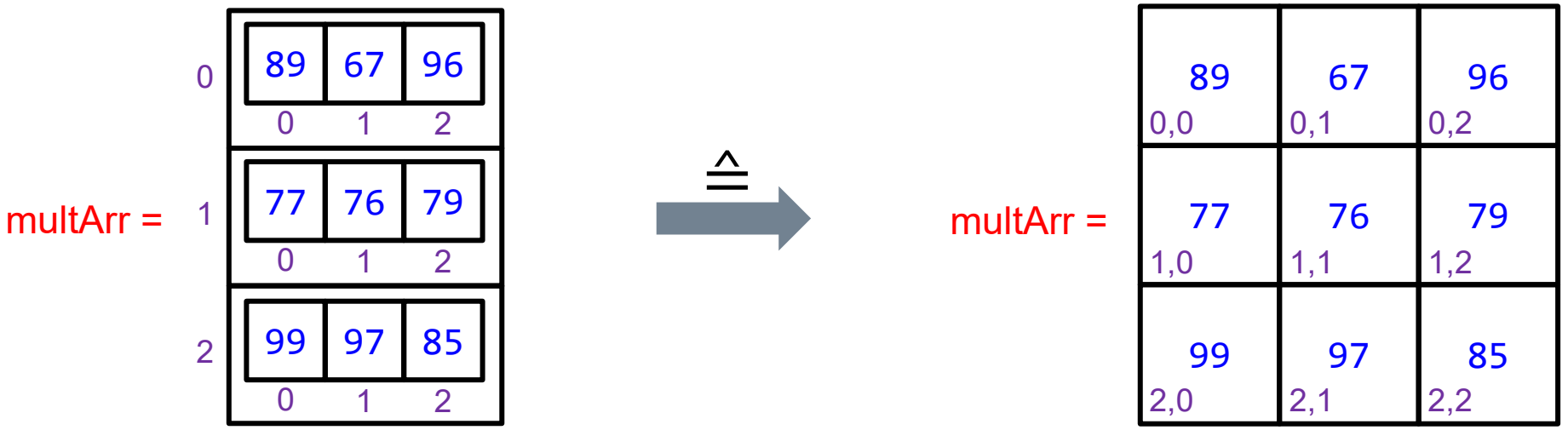
Write a program in C which will fill a 2D matrix of characters (of size 7 × 7) such that when printed, the following pattern appears on the screen:

```
X . . . . . X
. X . . . X .
. . X . X . .
. . . X . . .
. . X . X . .
. X . . . X .
X . . . . . X
```

- Comprehension questions: (i) How would you declare a 7 × 7 matrix of strings in C? (ii) What is the difference between the 2D char array used in the task and an 1D array of strings?
- (optional) Generalize the program such that the cross pattern can have a discretionary (odd) size.
- (optional) Can you do it without an array and only a single while or for loop?

Task 3: Multi-dimensional Arrays

```
int multArr[3][3];
```

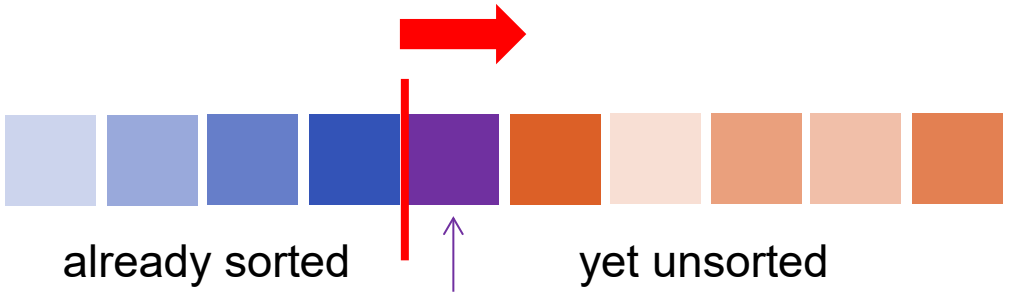


Task 3: Solution in Pseudocode

Algo: MULTIPLYMATRIX(A,B)

```
result ← 3 × 3 empty array;
for i = 1 to 3 do
    for j = 1 to 3 do
        sum ← 0
        for k = 1 to 3 do
            sum ← sum + A[i][k] × B[k][j]
        result[i][j] = sum
```

Task 4: Selection Sort



Algo: SelectionSort(A)

```
for i = 1 to n-1 do                    element currently under consideration
    k = i;
    for j = i+1 to n do                ← look at elements in unsorted part
        if A[j]<A[k] then k = j;       ← find smallest element
    exchange A[i] and A[k];
```

Task 4: Comparing Ascending and Descending Algorithm

Algo: ASCSELECTSORT(A,n)

```
for i = 1 to n − 1 do
    min = i
    for j = i + 1 to n do
        if A[j] < A[min] then
            min = j
    exchange A[i], A[min]
return A
```

Algo: DESCSELECTSORT(A,n)

```
for i = 1 to n − 1 do
    max = i
    for j = i + 1 to n do
        if A[j] > A[max] then
            max = j
    exchange A[i], A[max]
return A
```

Additional Recitation of Topics from the Lecture

- Basic Sorting
- Recursion

Additional Recitation of Topics from the Lecture

(see seperate slide decks)

Preview on Exercise 2

- Task 1: Reverse String
- Task 2: Perfect Square Number
- Task 3: Matrix Multiplication
- Task 4: Sorting

Exercise 1 – Task 1: Basic Recursion

Write the recursive function `int exponent(int x, int pow)` in C that computes the exponent of the base. An input/output example is illustrated below (input is typeset in bold):

> *Please enter the base:* ***2***
>
> *Please enter the power:* ***3***
>
> *The result is:* ***8***

Exercise 1 – Task 1: Basic Recursion: Remarks

- A single integer can be read as follows:

```
int input;
scanf("%d", &input);
```

- Think about special cases (e.g. negative or zero exponents etc.).

Exercise 1 – Task 2: Multiple Recursion

In a sequence a(n) = a_1, a_2, ..., an with n ≥ 3 integers, the first two integers $a_1 = 1$ and $a_2 = 2$ are fixed.

For i ≥ 3

$$a_i = \begin{cases} a_{i-1} / 3 + a_{i-2} & \text{if } a_{i-1} \text{ is divisible by 3} \\ a_{i-1} + a_{i-2} & \text{otherwise} \end{cases}$$

1. *Determine a_3 and a_4, sketch the trace process of your recursive function and calculate how many times your recursive function is called.*
2. *Write the recursive function `int sequence(int n)` in C that computes and prints all the integers of a(n).*

Exercise 1 – Task 2: Multiple Recursion: Remarks

- In the first part where you are tasked to “sketch the trace”, I suggest to (also) draw the recursion tree.

Exercise 1 – Task 3: Iteration and Recursion

Implement the C program that prints the index of the first uppercase letter of the given string using iteration and recursion, respectively. If there isn't any uppercase letter, return −1.

1. *Implement an iterative function* `int iterativeFirstUpper(char str[])`.
2. *Implement a recursive function* `int recursiveFirstUpper(char str[], int pos)` *where* `pos` *is the position of a character.*

Exercise 1 – Task 3: Iteration and Recursion: Remarks

- If you test your program using an input by the user (not explicitly required in the task description), you may assume a fixed maximum length for the input string and you can read it from the console as follows:

```
char input[1001];
scanf ("%[^\n]s", input);
```

- You may go with the hint from the task description and use `ch >= 'A' && ch <= 'Z'` as a condition to check whether variable `ch` is an uppercase letter or you may also use the `isupper` function from the ctype library (requires `#include <ctype.h>`).

Exercise 1 – Task 4: Pascal Triangle

The Pascal triangle is a useful tool that calculates the coefficients in the expansion of the polynomial $(a+b)^n$. Each element in the pascal triangle is associated with a coordinate (i, j) that is row i and index j of the row i. Both i and j start from 0. In row i, there are i + 1 elements. For example, in row 2, there are 3 elements, and 2 is associated with (2, 1). Among the i + 1 elements of row i, the first and last elements are both 1, any other element at (i, j) are the sum of the element at (i − 1, j − 1) and the element at (i − 1, j) in the previous row i − 1. For example, in row 2, 2 at (2, 1) is the sum of 1 at (1, 1) at row 1 and 1 at (1, 0) at row 1.

1. *Write the recursive function `int pascal(int i, int j)` in C that returns the value at the position (i, j) in the pascal triangle.*
2. *Write a function `void printPascal(int n)` that prints the first n rows of the pascal triangle.*
3. *(Optional) The pascal triangle can be printed in another way, as seen in the below example:*

…

Exercise 1 – Task 4: Sorting: Remarks

- I suggest to first write down the mathematical formula for calculating the coefficients in the Pascal triangle before heading right into the implementation stage.

Wrap-Up

- Summary
- Feedback
- Outlook
- Questions

Wrap-Up

– Summary

– I will have a feedback meeting with the lecturer: If you want me to tell me something (without mentioning your name, of course), please tell me.

Outlook on Next Thursday's Lab Session

Next tutorial: Thursday, 11.03.2021, 09.00 h, on Zoom

Topics:

- Review of Exercise 2
- Preview to Exercise 3
- Intro to Pseudocode
- Asymptotic Complexity
- Running Time Analysis
- …
- … (your wishes)

Questions?

Retrospective

https://www.klicker.uzh.ch/algodat

- Please take 5 minutes to answer these questions:
  - Was the teaching speed adequat?
  - What were the most important points / insights of today's exercise session for you?
  - What things remained unclear or were confusing? About what do you want to know more or needs clarification?
  - What I wanted to say anyway…

Thank you for your attention.