University of Zurich^UZH

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Informatics II
Exercise 6

March 25, 2019

Pointers

Task 1. Write a pseudo code for a program that reads an input array `A[0...n-1]`, reverse the elements of an array and prints the reordered array. For reversing the array you should use two pointers.

Task 2. What is the output of the following program?

```
#include<stdio.h>

void main() {
    int a[] = {0, 1, 2, 3, 4};
    int i, ∗p;

    for (p = a; p ≤&a[4]; p++) {
        printf(”%d␣”, ∗p);
    }
    printf(”\n”);

    p = &a[0];
    for (i = 0; p + i < a + 4; i++) {
        printf(”%d␣”, ∗(p+i));
        p=p+1;
    }
    printf(”\n”);

    p = a+4;
    for (i = 0; i ≤4; i++) {
        printf(”%d␣”, p[−i]);
    }
    printf(”\n”);
}
```

Task 3. Create a C program that reads elements of a rectangular matrix of integers and then creates transpose of a matrix and prints the transposed rectangular matrix. Allocate space for matrix dynamically.
<u>**Note:**</u> For allocation of space use malloc and do not forget to free the allocated

space using free. For the creation of matrix, you should use double pointer (pointer to pointer). Double pointer will keep the address of the allocated array of pointers (represents the rows of matrix) and each pointer in the array will point to one allocated matrix row.

Linked Lists

Task 4. Write a C program that implements single-linked lists of integers defined as follows:

```
struct list {
  struct node* head;
};
```

```
struct node {
  int val;
  struct node* next;
};
```

Your program must contain the following functions:

a) *struct list* init()* that initializes a list.

b) *int size(struct list *listA)* that returns number of elements in the list.

c) *void appendAtTail(struct list *listA, int val)* that appends a new node containing integer `val` at the end of the list.

d) *void appendAtHead(struct list *listA, int val)* that appends a new node containing integer `val` at the beginning of the list.

e) *void appendAtPosition(struct list *listA, int val, int i)* that appends a new node containing integer `val` at the *i-th* position in the list. The index of the first element is 0.

f) *void reverse(struct list *listA)* that reverses the list in O(n) time.

g) *void clear(struct list *listA)* that removes all nodes from the list.

h) *void deleteVal(struct list *listA, int val)* that removes all nodes from the list that have value `val`.

i) *void delete(struct list *listA, int i)* that removes the node in the *i-th* position in the list. The index of the first element is 0.

j) *void print(struct list *listA)* that prints the values of the elements of the list to the console enclosed in brackets, e.g. `[ 1 2 3 4 ]`.

k) *void avg(struct list *listA)* that finds the average value of all elements in the list and prints the value in the console after the message “`avg =`”.

IMPORTANT: Every time you add or remove a node make sure you request (malloc) and free (free) the memory involved, correspondingly.

After you have implemented and tested these functions with lists of your choice, include the following sequence in your *main()* method:

1. Insert 1,2,3,4 to the head of the list, in the given order.
2. Insert 8,7,6,5 to the tail of the list, in the given order.
3. Print the list to the console.
4. Reverse the list.
5. Insert 0 at position 4 in the list.
6. Print the list to the console.
7. Remove the nodes in positions 0, 2, 3 and 5 from the list, one after another.
8. Print the list to the console.
9. Remove the all element from the list with value 0.
10. Print the list to the console.
11. Find the average value of all elements in the list and print the value to the console.
12. Clear the list.
13. Print the list to the console.