University of Zurich^UZH

Solution

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

Informatics II
Exercise 6 / **Solution**

March 30, 2020

Pointers

Task 1. Write a C code for a program that reads an input array `A[0...n-1]`, sorts the element of an array in *descending* order using bubble sort and prints the sorted array. You should use pointers instead of `[]` to access the elements of array.

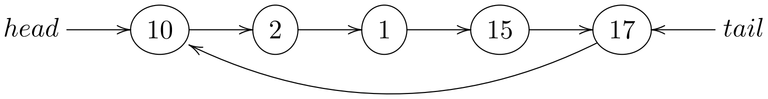
```c
void sort(int n, int* ptr) {
    int i, j, t;

    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {

            if (*(ptr + j) < *(ptr + j + 1)) {

                t = *(ptr + j + 1);
                *(ptr + j+1) = *(ptr + j);
                *(ptr + j) = t;
            }
        }
    }

    for (i = 0; i < n; i++) {
        printf("%d ", *(ptr + i));
    }
}
```

Task 2. Write a C code for a program that reads a string `source`, copy it using pointers to new string `target` and then prints the copied string as output.

```c
void copy_string(char *target, char *source){
    while(*source){
        *target = *source;
        source++;
        target++;
    }
    *target = '\0';
}
```

Linked Lists

Task 3. A circular linked list is a variation of a linked list in which the last node points to the first node, completing a full circle of nodes.



Write a C program that implements circular linked lists of integers defined as follows:

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

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

Your program must contain the following functions:

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

```
struct list* init() {
    struct list* l = malloc(sizeof(struct list));

    l->head = NULL;
    l->tail = NULL;
    return l;
}
```

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

```
void appendAtTail(struct list* listA, int val) {
    struct node* new = malloc(sizeof(struct node));

    new->val = val;
    if (listA->head == NULL){
        listA->head = new;
        listA->tail = new;
        listA->tail->next = listA->head;
    }
    else {
        new->next = listA->head;
        listA->tail->next = new;
        listA->tail = new;
    }
}
```

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

```
void appendAtHead(struct list* listA, int val) {
    struct node* new = malloc(sizeof(struct node));

    new->val = val;
    if (listA->head == NULL){
        listA->head = new;
        listA->tail = new;
        listA->tail->next = listA->head;
    }
    else {
        new->next = listA->head;
        listA->head = new;
        listA->tail->next = new;

    }
}
```

d) *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.

```
void appendAtPosition(struct list* listA, int val, int i) {

    if (i < 0 || i > size(listA)) {
        return;
    }

    if (i == 0) {
        appendAtHead(listA, val);
        return;
    }

    if (i == size(listA)) {
        appendAtTail(listA, val);
        return;
    }

    int n = 0;

    struct node* curr;
    struct node* new = malloc(sizeof(struct node));

    new->val = val;
    new->next = NULL;

    curr = listA->head;
    while (n + 1 < i)
    {
        n++;
        curr = curr->next;
    }
    new->next = curr->next;
    curr->next = new;

}
```

e) *void print(struct list *listA)* that prints the values of the elements of the list to the console enclosed in brackets, e.g. `[ 3 5 7 2 ]`.

```
void print(struct list* listA) {
    struct node* curr = listA->head;

    printf("[␣");
    if (curr != NULL){
        while (curr -> next != listA->head) {
            printf("%d␣", curr->val);
            curr = curr->next;
        }
        printf("%d␣", curr->val);
    }
    printf("]\n");
}
```

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

```
void deleteVal(struct list* listA, int val) {
    struct node* toDelete;
    struct node* prev = listA->head;

    if (prev != NULL) {
        while (prev->next != listA->head) {
            if (prev == listA->head && prev->val == val) {
                toDelete = prev;
                listA->head = prev->next;
                listA->tail->next = listA->head;
                prev = prev->next;
                free(toDelete);
            }
            else {
                if (prev->next->val == val) {
                    toDelete = prev->next;
                    prev->next = prev->next->next;
                    if (toDelete == listA->tail){
                        listA->tail = prev;
                    }
                    free(toDelete);
                }
                else {
                    prev = prev->next;
                }
            }
        }
    }
}
```

g) *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.

```
void delete(struct list* listA, int i) {
    struct node* toDelete;
    struct node* prev;
    int c;

    if (i < 0 || i > size(listA) || size(listA)==0) {
        return;
    }

    if (i == 0) {
        if (listA->head != NULL) {
            toDelete = listA->head;
            listA->head = listA->head->next;
            listA->tail->next = listA->head;
            free(toDelete);
        }
    }
    else {
        c = 1;
        prev = listA->head;
        while (c < i) {
            c++;
            prev = prev->next;
        }
        if (c==i){
            toDelete = prev->next;
            prev->next = prev->next->next;
            if (toDelete == listA->tail){
                listA->tail = prev;
            }
            free(toDelete);
        }
    }
}
```

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

1. Insert 9,4,5,3,1,2,0 to the list, in the given order i.e. output list should be like `[9, 4, 5, 3, 1, 2, 0]`.

2. Print the list to the console.

3. Remove the nodes in positions 6, 3 and 0 from the list, one after another.

4. Print the list to the console.

5. Insert 9,4,5 to the head of the list, in the given order i.e first insert 9 then 4 and finally 5 at the head and these elements should be arranged in the order `[5, 4, 9]`.

6. Insert 3,1,2,0 to the tail of the list, in the given order i.e tail should be arranged in the order `[3, 1, 2, 0]`.

7. Print the list to the console.

8. Insert 7 at position 4 in the list.

9. Print the list to the console.

10. Remove the nodes in positions 0, 3 and 6 from the list, one after another.

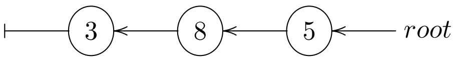
11. Print the list to the console.

12. Remove the all element from the list with value 0.

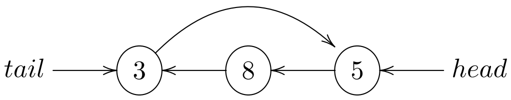
13. Print the list to the console.

```
struct list* l = init();
appendAtHead(l, 0);
appendAtHead(l, 2);
appendAtHead(l, 1);
appendAtHead(l, 3);
appendAtHead(l, 5);
appendAtHead(l, 4);
appendAtHead(l, 9);
print(l);
delete(l, 6);
delete(l, 3);
delete(l, 0);
print(l);
appendAtHead(l, 9);
appendAtHead(l, 4);
appendAtHead(l, 5);
appendAtTail(l, 3);
appendAtTail(l, 1);
appendAtTail(l, 2);
appendAtTail(l, 0);
print(l);
appendAtPosition(l, 7, 4);
print(l);
delete(l, 0);
delete(l, 3);
delete(l, 6);
print(l);
deleteVal(l, 0);
print(l);
```

Task 4. A linked list can be used to represent decimal numbers in place-value notation. In such a scheme, the first node of the list represents the least significant digit, the second node the second-least significant digit and so on. For example, the list below represents number 385.



It can be represented using the circularly linked list defined in Task 3 as follows.



Write a C program that uses a function *struct list* addTwoLists (struct list* numA, struct list* numB)* to add the two integers represented using the circularly linked lists *numA* and *numB* and returns a pointer to a new circularly list that contains the result of the addition. For example if `numA = 91` and `numB = 1249`, then the output should be

```
Sum of numA and numB = 1340
```

```
struct list* addTwoLists (struct list* numA, struct list* numB) {
  struct list* res = init();
  struct node* first = numA->head->next;
  struct node* second = numB->head->next;
  int a = numA->head->val, b=numB->head->val;
  int carry = 0, sum;

  sum = carry + a + b;
  if (sum >=10) carry = 1;
  else carry = 0;

  sum = sum % 10;
  appendAtTail(res, sum);

  while (first != numA->head || second != numB->head) {
    if (first!=numA->head) a = first->val;
    else a = 0;
    if (second!=numB->head) b = second->val;
    else b = 0;

    sum = carry + a + b;

    if (sum >=10) carry = 1;
    else carry = 0;

    sum = sum % 10;
    appendAtTail(res, sum);

    if (first != numA->head) first = first->next;
    if (second != numB->head) second = second->next;
  }

  if (carry > 0) {
    appendAtTail(res, 1);
  }
  res->tail->next = res->head;

  return res;
}
```

Initialization and Function Call:

```
struct list* a = init();
struct list* b = init();
struct list* res = NULL;
// first list: 91
appendAtHead(a, 9);
appendAtHead(a, 1);
// second list: 1249
appendAtHead(b, 1);
appendAtHead(b, 2);
appendAtHead(b, 4);
appendAtHead(b, 9);
res = addTwoLists(a, b);
printf("Sum of ");
print(a);
printf(" and ");
print(b);
printf(" = ");
print(res);
printf("\n");
```