

Programming for computerteknologi

Hand-in Assignment Exercises

Week 8: Designing Sequences of Program Instructions for Solving Problems

Exercise 1)

We have been given a task to consider a program for computing factorial numbers and then next determine how many arithmetic operators there's required to compute $fact(5)$ and how many arithmetic operators required to compute $fact(n)$ for any positive integer n . A scheme for the two situations is illustrated below.

	BEFORE LOOP	INSIDE LOOP	(INSIDE LOOP) X (5 OF LOOP ITERATIONS)
NUMBER OF ARITHMETIC OPERATIONS	0	2	$2 \times 5 = 10$

In this case there's two arithmetic operators inside the loop and the loop runs 5 times which results in a total of 10 arithmetic operators when computing $fact(5)$.

	BEFORE LOOP	INSIDE LOOP	(INSIDE LOOP) X (N OF LOOP ITERATIONS)
NUMBER OF ARITHMETIC OPERATIONS	0	2	$2 \times n = 2n$

In this case, there's two arithmetic operators inside the loop once more and the loop runs a total of n times when computing $fact(n)$. Therefore, the total amount of arithmetic operators can be determined by 2 times the input value for any positive integer.

Exercise 2)

We have been given a task to implement an insertion sort algorithm like the ones we have discussed in the lectures, but the implementation of the algorithm should be based on singly linked lists this time.

(I should mention that this assignment has been made in Replit, since I couldn't get my Microsoft Visual Studio Code to work. Neither the test files nor my main file. I have of course linked to all my Replits.)

```
66 // function to sort linked list using insertion sort algorithm
67 void sort(linked_list *llPtr) {
68     struct node *sortedList = NULL; // defining and initializing Sorted list
69     struct node *curr = llPtr->head; // temporary node for checking linked list
70     while (curr != NULL) {
71         struct node *next = curr->next; // saving our temp nodes next for the while loop
72         struct node *tempNode; // new placeholder node to hold node for swap
73         if (sortedList == NULL || sortedList->data >= curr->data) {
74             curr->next = sortedList;
75             sortedList = curr;
76         } else {
77             tempNode = sortedList; // running through sorted list to locate place to insert
78             while (tempNode->next != NULL && tempNode->next->data < curr->data) {
79                 tempNode = tempNode->next;
80             }
81             swap(curr, tempNode); // inserting at the right point using our swap function
82         }
83         curr = next; // resetting curr and assigning it to the next node
84     }
85     llPtr->head = sortedList; // assigning our list to the sorted list
86 }
```

<https://replit.com/join/htzrrdxikz-mikk772h>

I have tried my best to include the given tests. These have been included in my main function.
Test case 1:

```
92 // Test case 1: [-3, 22, 11, 33, 3, 2, 1] - must be: [-3, 1, 2, 3, 11, 22, 33]
93 linked_list *test1 = createLinkedList(); // initializing list
94 // inserting values in the first test list
95 insertFront(createNode(1), test1);
96 insertFront(createNode(2), test1);
97 insertFront(createNode(3), test1);
98 insertFront(createNode(33), test1);
99 insertFront(createNode(11), test1);
100 insertFront(createNode(22), test1);
101 insertFront(createNode(-3), test1);
102
103 printf("List before insertion sort\n");
104 printLL(test1); // list before sort
105 printf("\n");
106 sort(test1); // sorting the list using insertion sort algorithm
107 printf("List after insertion sort\n");
108 printLL(test1); // list after sort
```

Output:

```
List before insertion sort
| -3 | 22 | 11 | 33 | 3 | 2 | 1
List after insertion sort
| -3 | 1 | 2 | 3 | 11 | 22 | 33
```

Test case 2:

```
111 // Test case 2: [0, -1, 1] - must be: [-1, 0, 1]
112 linked_list *test2 = createLinkedList(); // initializing list
113 // inserting values in the second test list
114 insertFront(createNode(1), test2);
115 insertFront(createNode(-1), test2);
116 insertFront(createNode(0), test2);
117
118 printf("\n\nList before insertion sort\n");
119 printLL(test2); // list before sort
120 printf("\n");
121 sort(test2); // sorting the list using insertion sort algorithm
122 printf("List after insertion sort\n");
123 printLL(test2); // list after sort
```

Output:

```
List before insertion sort
| 0 | -1 | 1
List after insertion sort
| -1 | 0 | 1
```

Test case 3:

```
126 // Test case 3: [6, 6, 5, 4, 2, 3, 3, 3, 2, 1] - must be: [1, 2, 2, 3, 3, 3, 4, 5, 6, 6]
127 linked_list *test3 = createLinkedList(); // initializing list
128 // inserting values in the third test list
129 insertFront(createNode(1), test3);
130 insertFront(createNode(2), test3);
131 insertFront(createNode(3), test3);
132 insertFront(createNode(3), test3);
133 insertFront(createNode(3), test3);
134 insertFront(createNode(2), test3);
135 insertFront(createNode(4), test3);
136 insertFront(createNode(5), test3);
137 insertFront(createNode(6), test3);
138 insertFront(createNode(6), test3);
139
140 printf("\n\nList before insertion sort\n");
141 printLL(test3); // list before sort
142 printf("\n");
143 sort(test3); // sorting the list using insertion sort algorithm
144 printf("List after insertion sort\n");
145 printLL(test3); // list after sort
```

Output:

```
List before insertion sort
| 6 | 6 | 5 | 4 | 2 | 3 | 3 | 3 | 2 | 1
List after insertion sort
| 1 | 2 | 2 | 3 | 3 | 3 | 4 | 5 | 6 | 6
```

By looking at the three above test cases we can see that they have all passed the tests which means that the program has been executed successfully.

Exercise 3)

We have been given a task to implement a queue program based on singly linked lists as discussed in the lecture. We have to implement the following four functions:

- Init_queue (Initializes the queue)
- Enqueue (inserts an element into the queue)
- Dequeue (dequeues an element from the queue)
- Empty (checks if the queue is empty or not)

I have used the same logic as in last week's assignment, where we had to implement a stack queue. Stacks and queues are very similar, so therefore I should be able to make a queue with the same structures used in the stack assignment. The four implemented function look like this:

(<https://replit.com/join/mkdwyrbbp-mikk772h>)

```
7 ▼ typedef struct node {
8     int data;
9     struct node* next;
10 } node;
11
12 ▼ typedef struct {
13     node* head;
14 } queue;
```

```
17 // function to initialize queue
18 ▼ void init_queue(queue* q) {
19     q->head = NULL; // initializing queue
20     printf("SUCCESS!\n\n");
21 }
```

```
24 // function to enqueue element into list
25 ▼ void enqueue(int x, queue* q) {
26     struct node *newNode;
27     newNode = (node*) malloc(sizeof(node)); // creating new node to add to queue
28     newNode->data = x; // assigning data to the newly created node
29 ▼ if (q->head == NULL) {
30     newNode->next = NULL;
31 ▼ } else {
32     newNode->next = q->head; // adding node to the queue
33 }
34 q->head = newNode; // including node in queue
35 }
```

```
38 // function to dequeue element from list
39 ▼ int dequeue(queue *q) {
40     int count = 0; // counter to test how long the queue is
41     int number = q->head->data; // assigning output number the first data element
42     struct node *tempQueue = q->head; // temporary node
43 ▼ while (tempQueue->next != NULL) { // counting elements in queue
44     count++;
45     tempQueue = tempQueue->next;
46     number = tempQueue->data;
47 }
48 int value[count]; // creating array based on amount of elements in queue
49 ▼ for(int i = 0; i <= count; i++) {
50     value[i] = q->head->data; // assigning data elements to array
51     q->head = q->head->next;
52 }
53
54 int rev_value[count];
55 int n = count;
56 ▼ for(int i = 0; i < count; i++) { // reversing array to get queue in the right
order
57     n--;
58     rev_value[n] = value[i];
59 }
60
61 ▼ for(int i = 0; i < count; i++) { // creating new queue without the element in
the front
62     struct node *newNode;
63     newNode = (node*) malloc(sizeof(node)); // node to hold value from array
64     newNode->data = rev_value[i];
65 ▼ if (q->head == NULL) {
66     newNode->next = NULL;
67 ▼ } else {
68     newNode->next = q->head; // adding node to the new queue
69 }
70     q->head = newNode; // assigning queue to the new one
71 }
72 return number; // returning dequeued integer
73 }
```

```
76 // function to check if queue is empty
77 void empty(queue *q) {
78     if(q->head == NULL) { // checking if queue is empty
79         printf("QUEUE IS EMPTY\n\n");
80     } else {
81         printf("QUEUE IS NOT EMPTY\n\n");
82     }
83 }
```

I have tried my best to include the given tests. These have been included in my main function. We want to test (1) If the queue is empty when initialized, (2) If the queue is the same after executing both enqueue and dequeue and (3) The values dequeued must be the same as the values enqueued in the right order. All these test cases can be seen below.

```
105 queue test;
106 printf("Initializing Test Queue 1: ");
107 init_queue(&test);
108 empty(&test);
```

Initializing Test Queue 1: SUCCESS!

QUEUE IS EMPTY

(1) Here we can see that the queue is empty when initialized which makes this test successful.

```
105 queue test;
106 printf("Initializing Test Queue 1: ");
107 init_queue(&test);
108 empty(&test);
109 enqueue(2, &test);
110 display(&test);
111 dequeue(&test);
112 display(&test);
```

Initializing Test Queue 1: SUCCESS!

QUEUE IS EMPTY

The queue is:
2--->NULL

QUEUE IS EMPTY!

(2) In this case, after dequeuing the queue is left empty means that this case is also successful.

```
117  int x0 = -5;
118  enqueue(x0, &test);
119  int x1 = 10;
120  enqueue(x1, &test);
121  int x2 = 0;
122  enqueue(x2, &test);
123  int x3 = 5;
124  enqueue(x3, &test);
125
126  display(&test);
127
128  int y0 = dequeue(&test);
129  assert(x0 == y0);
130  int y1 = dequeue(&test);
131  assert(x1 == y1);
132  int y2 = dequeue(&test);
133  assert(x2 == y2);
134  int y3 = dequeue(&test);
135  assert(x3 == y3);
136
137  printf("\n\nALL TESTS PASSED!"); // assuring that my assert statements work.
    If this message gets printed in the console it means that the program has
    executed successfully
```

The queue is:
5--->0--->10--->-5--->NULL

ALL TESTS PASSED! 🎉 ☐

- (3) In this case, we use asserts to ensure that the statements are correct. It can be seen, that the values enqueued is the same as the values dequeued, which makes this test case successful too.