

CPSC 3125, Operating Systems Lab Assignment

Lab 07

1 Instructions

Using the starter code below, implement a linked list. You are given these four functions:

- `int menu();`
- `void dowhat(int);`
- `Node *create_ll(Node *);`
- `Node *display(Node *);`

You implemented these four functions last week:

- `Node *insert_beg(Node *);`
- `Node *insert_end(Node *);`
- `Node *insert_before(Node *);`
- `Node *insert_after(Node *);`

You will implement these five functions this week:

- `Node *delete_beg(Node *);`
- `Node *delete_end(Node *);`
- `Node *delete_node(Node *);`
- `Node *delete_after(Node *);`
- `Node *delete_list(Node *);`

I have given you an executable you can use as a guide. This is a strenuous exercise, but you will learn a lot about pointer manipulatio.

2 Starter Code

```

1  /*****
2  * Name: lab07.c
3  * Author: C data structures
4  * October 23, 2021
5  * Purpose: Write a program to create a linked list and perform insertions and deletions of
        all cases.
6  * Write functions to sort and finally delete the entire list at once.
7  * Compile with: lab07.c -o lab07.exe -Wall
8  * *****/
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <malloc.h>
13
14 struct node
15 {
16     int data;
17     struct node *next;
18 };
19
20 typedef struct node Node;
21
22 //variable declarations
23 int option;
24 Node *start = NULL;
25

```

```

26 //function declarations
27 int menu();
28 void dowhat(int);
29
30 Node *create_ll(Node *);
31 Node *display(Node *);
32 Node *insert_beg(Node *);
33 Node *insert_end(Node *);
34 Node *insert_before(Node *);
35 Node *insert_after(Node *);
36
37 int main()
38 {
39     system("clear");
40     do
41     {
42         option = menu();
43         printf("start == %p\n", start);
44         dowhat(option);
45     }
46     while(option != 13);
47
48     return 0;
49 }
50
51 int menu()
52 {
53     printf("\n\n*****MAIN_MENU*****");
54     printf("\n1: Create a list");
55     printf("\n2: Display the list");
56     printf("\n3: Add a node at the beginning");
57     printf("\n4: Add a node at the end");
58     printf("\n5: Add a node before a given node");
59     printf("\n6: Add a node after a given node");
60     printf("\n13: EXIT");
61     int option;
62     printf("\n\nEnter your option: ");
63     scanf("%d", &option);
64
65     return option;
66 }
67
68 void dowhat(int option)
69 {
70     switch(option)
71     {
72         case 1: start = create_ll(start);
73                 printf("\nLINKED_LIST_CREATED");
74                 break;
75         case 2: start = display(start);
76                 break;
77         case 3: start = insert_beg(start);
78                 break;
79         case 4: start = insert_end(start);
80                 break;
81         case 5: start = insert_before(start);
82                 break;
83         case 6: start = insert_after(start);
84                 break;
85     }
86 }
87
88 Node *create_ll(Node *start)
89 {
90     Node *new_node, *ptr;
91     int num;
92     printf("\nEnter the data (-1 to end): ");
93     scanf("%d", &num);

```

```

94     while(num != -1)
95     {
96         new_node = (Node*) malloc( sizeof(Node));
97         new_node->data = num;
98         if( start==NULL)
99         {
100             //printf(" if _branch, _start ==_NULL\n");
101             new_node->next = NULL;
102             start = new_node;
103         }
104         else
105         {
106             //printf(" else _branch, _start ==_%p\n", start);
107             ptr = start;
108             while(ptr->next!=NULL)
109                 ptr = ptr->next;
110             ptr->next = new_node;
111             new_node->next = NULL;
112         }
113         printf(" _Enter _the _data_: ");
114         scanf("%d", &num);
115     }
116     return start;
117 }
118
119 Node *display(Node *start)
120 {
121     Node *ptr;
122     ptr = start;
123     printf("\nLinked _List _\n");
124     printf(" _start->%p\n", start);
125     while(ptr != NULL)
126     {
127         printf(" _%p<-_%d->%p\n", ptr, ptr->data, ptr->next);
128         ptr = ptr->next;
129     }
130     printf(" _n_\n");
131     return start;
132 }
133
134 Node *insert_beg(Node *start)
135 {
136     //declare a pointer to a ne Node
137     //declare a new integer as a data value
138     printf(" _n_Enter _the _data_: ");
139     //get user input for the data
140     //malloc memory for a new Node
141     //initialize the data member to the new integer
142     //initialize the next member to the start of the list
143     //set start to the address of the new Node
144     printf("in _insert_beg() _new_node _address: %p, _new_node->data: %d, _new_node->next: %p\n", new_node, new_node->data, new_node->next);
145     return start;
146 }
147
148 Node *insert_end(Node *start)
149 {
150     //declare a pointer to a new Node, and a pointer to a iteration node
151     //declare a new integer as a data value
152     printf(" _Enter _the _data_: ");
153     //get user input for the data
154     //malloc memory for a new Node
155     //initialize the data member to the new integer
156     //initialize the next member to NULL (the end of the list)
157     //set the iteration pointer to start
158     while(ptr->next != NULL)
159         ptr = ptr->next;
160     //set the iteration pointer next member to the new Node

```

```

161     printf("in insert_end() — new_node address: %p, new_node->data: %d, new_node->next: %p\n", new_node, new_node->data, new_node->next);
162     return start;
163 }
164
165 Node *insert_before(Node *start)
166 {
167     //declare a pointer to a new Node, a pointer to a iteration node, and a pointer to the
168     //“pre” insertion Node
169     //declare a new integer as a data value, and a new integer to hold the value before
170     //which the new Node is to be inserted
171     printf("\nEnter the data: ");
172     //get the data value from the user
173     printf("\nEnter the value before which the data has to be inserted: ");
174     //get the “before” value from the user
175     //malloc memory for a new Node
176     //initialize the data member to the new integer
177     //set the iteration pointer to start
178     while(ptr->data != val)
179     {
180         preptr = ptr;
181         ptr = ptr->next;
182     }
183     //set the next member of the pre-pointer to the new Node
184     printf("in insert_before() — new_node address: %p, new_node->data: %d, new_node->next: %p\n", new_node, new_node->data, new_node->next);
185     return start;
186 }
187
188 Node *insert_after(Node *start)
189 {
190     //declare a pointer to a new Node, a pointer to a iteration node, and a pointer to the
191     //“pre” insertion Node
192     //declare a new integer as a data value, and a new integer to hold the value after which
193     //the new Node is to be inserted
194     printf("\nEnter the data: ");
195     //get the data value from the user
196     printf("\nEnter the value after which the data has to be inserted: ");
197     //get the “after” value from the user
198     //malloc memory for a new Node
199     //initialize the data member to the new integer
200     //set the iteration pointer to start
201     //set the preptr to ptr
202     while(preptr->data != val)
203     {
204         preptr = ptr;
205         ptr = ptr->next;
206     }
207     //set the next member of the pre-pointer to the new Node
208     //set the next member of the new Node to the pointer
209     printf("in insert_after() — new_node address: %p, new_node->data: %d, new_node->next: %p\n", new_node, new_node->data, new_node->next);
210     return start;
211 }
212
213 Node *delete_beg(Node *start)
214 {
215     if(start == NULL)
216     {
217         printf("no list to delete\n");
218         return NULL;
219     }
220     //declare a pointer to a Node object
221     //set pointer to start
222     //set start variable to the next member of the start Node
223     //free the pointer
224     return start;
225 }

```

```

222
223 Node *delete_end(Node *start)
224 {
225     //declare to Node pointers , one to ptr and one to preptr
226     //set ptr to start
227     //iterate through list to end
228     while(ptr->next != NULL)
229     {
230         preptr = ptr;
231         ptr = ptr->next;
232     }
233     //set the next member of preptr to NULL
234     //free the pointer
235     return start;
236 }
237
238 Node *delete_node(Node *start)
239 {
240     //declare variables to ptr and preptr
241     //declare an integer variable
242     printf("\nEnter the value of the node which has to be deleted: ");
243     //read the input from the user
244     //set ptr to start
245     if(ptr->data == val) //the value to delete is the first item
246     {
247         start = delete_beg(start);
248         return start;
249     }
250     else
251     {
252         while(ptr->data != val)
253         {
254             preptr = ptr;
255             ptr = ptr->next;
256         }
257         //set the next member of preptr to the next member of ptr
258         //free the ptr
259         return start;
260     }
261 }
262
263 Node *delete_after(Node *start)
264 {
265     //declare pointers to ptr and preptr
266     //declare an integer variable
267     printf("\nEnter the value after which the node has to be deleted: ");
268     //get input from the user
269     //set ptr to start
270     //set preptr to pstr
271     while(preptr->data != val)
272     {
273         preptr = ptr;
274         ptr = ptr->next;
275     }
276     //set the next member of preptr to the next member of ptr
277     //free the ptr
278     return start;
279 }
280
281 Node *delete_list(Node *start)
282 {
283     if(start == NULL)
284     {
285         printf("no list to delete\n");
286         return NULL;
287     }
288     //declare a pointer to Node
289     //set the ptr to start

```

```

290     while(1)
291     {
292         //set start to the return value to the delete_bet function
293         if(start == NULL)
294         {
295             printf("no_list_to_delete\n");
296             return NULL;
297         }
298         else
299             //set ptr to start
300     }
301     return start;
302 }

```

3 Output

See the demonstration in class and the accompanying executable.

4 Lab deliverable

Your deliverable consists of (1) the C source code, and (2) a text document showing the output of the program.