|  |  |
| --- | --- |
| **CSCI 2100E Spring 2020** | Student Name: \_Tam Yi Ki \_Student Id: \_\_\_1155126127\_\_ |

# Lab Exercise #2

*Lab exercises are always due 2 weeks after day of the lab. Submitting this lab exercise gives you* ***15%*** *bonus to your next test if you are below mean (the usual is 10%).*

*Please fill in the lab sheet and submit the completed Word doc file to blackboard.* *Places you need to fill in or work on are marked in red.*

## Demo 1

Below you will find a crude implementation of a social network in C. It doesn't work though. Why? (From Lecture Notes)

|  |
| --- |
| #include <stdio.h> #include <string.h>  struct contact {  char name[20];  int age; }; typedef struct contact Contact;  struct profile {  char name[20];  int age;  char hobbies[100];  struct contact friends[10]; }; typedef struct profile Profile;  int main() {  Profile p;  Contact p2,p3;    strcpy(p.name,"John");  p.age = 20;  strcpy(p2.name,"Chandra");  p2.age = 19;  strcpy(p3.name,"Jingyi");  p3.age = 25;    p.friends[0] = p2;  p.friends[1] = p3;    p3.age = 26;    printf("First friend of %s: %s, age %d\n",p.name,p.friends[0].name,p.friends[0].age);  printf("Second friend of %s: %s, age %d\n",p.name,p.friends[1].name,p.friends[1].age);   }  #include <stdio.h>  #include <string.h>  struct contact {  char name[20];  int age;  };  typedef struct contact Contact;  struct profile {  char name[20];  int age;  char hobbies[100];  Contact \*friends[10];  };  typedef struct profile Profile;  int main()  {  Profile p;  Contact p2,p3;    strcpy(p.name,"John");  p.age = 20;  strcpy(p2.name,"Chandra");  p2.age = 19;  strcpy(p3.name,"Jingyi");  p3.age = 25;    p.friends[0] = &p2;  p.friends[1] = &p3;    p3.age = 26;    printf("First friend of %s: %s, age %d\n",p.name,p.friends[0]->name,p.friends[0]->age);  printf("Second friend of %s: %s, age %d\n",p.name,p.friends[1]->name,p.friends[1]->age);    } |

## Demo 2

This is an incomplete implementation of a linked list. Your instructor will walk you through the code step by step

|  |
| --- |
| #include <stdio.h> #include <stdlib.h> // <--- remember this  typedef struct node {  int data; // just generic, meaningless data!  struct node \*next; } Node;  Node\* insert(Node\* listHead, int dataToInsert) {  // we need to return a new head of linkedlist because   // the head may change after insertion  if (listHead == NULL) {  // list is emtpy, let's insert our first element!  printf("inserting first element with data %d\n",dataToInsert);  Node \*newNode = malloc(sizeof(Node));  newNode->data = dataToInsert;  newNode->next = NULL;  return newNode;  } else {  // list is not empty, let's insert at the end  printf("inserting element with data %d\n",dataToInsert);  Node \*current = listHead; // CAUTION!  while (current != NULL) {  if (current->next == NULL) {  // if it's end of list...  Node \*newNode = malloc(sizeof(Node));  newNode->data = dataToInsert;  newNode->next = NULL;  current->next = newNode;  return listHead;  } else {  // otherwise...  current = current->next;  }  }  }   } void printList(Node\* listHead) {  printf("printing list: ");  Node \*current = listHead;  while (current != NULL) {  printf("%d ",current->data);  current = current->next;  }  printf("\n"); }  void printCount(Node\* listHead) {  printf("printing count: ");    // TODO: implement  }  Node\* deleteFirst(Node\* listHead) {  printf("deleting first element from list: \n");    // **TODO:** implement    return listHead; }  Node\* deleteLast(Node\* listHead) {  printf("deleting last element from list: \n");    // **TODO:** implement  return listHead; }  Node\* deleteSpecific(Node \*listHead, int dataValue) {  // assume data to delete always exists and is always unique in  // this exercise    printf("deleting %d from list \n", dataValue);     // **TODO:** implement!    return listHead; }  void printBackwardsFromZero(Node\* listHead) {  printf("printing list in backwards from the first 0 element (or end of list): ");    // **TODO:** implement..     printf("\n"); }  int main() {  Node \*myHead = NULL; // head of the linked list, not really MY HEAD!   myHead = insert(myHead, 2000);  myHead = insert(myHead, 2018);  printList(myHead);  myHead = insert(myHead, 2001);  myHead = insert(myHead, 1998);  myHead = insert(myHead, 0);  myHead = insert(myHead, 1999);  myHead = insert(myHead, 2021);  myHead = insert(myHead, 2020);  printList(myHead);  printCount(myHead);  myHead = deleteFirst(myHead);  printList(myHead);    myHead = deleteLast(myHead);  printList(myHead);    myHead = deleteSpecific(myHead, 1998);  printList(myHead);  myHead = deleteSpecific(myHead, 2018);  printList(myHead);    printBackwardsFromZero(myHead);    return 0; } |

In the above example, we have a partial implementation of linked list in C. **Don't be scared of the pointers!**

The main function is not fully working as some code is not implemented yet. However, we can see that the insert and print functions are working properly.

Notice that:

1. As mentioned, we reference a linked list by the list's **first Node** ("head")
2. Please notice the various use of pointers in the example; do you recall the **3 rules of using pointers**? (Please refer to lecture notes if not)
3. There are **two insertion cases** - if the list is empty, we need to create a new Node as the head; otherwise, we will append the Node in the end in this example

A1

A2

A3

A4

A5

NULL

X

1. We NEED to return a pointer to Node in insert because the list head pointer may change! We need to do the same for every operation that may change the list head

## Problem 1

Please try to implement the printCount function that prints out the number of Nodes in the list! If successful, you will see that when we call that in main, it will print out 8 in our specific example.

|  |
| --- |
| inserting first element with data 2000 inserting element with data 2018 printing list: 2000 2018  inserting element with data 2001 inserting element with data 1998 inserting element with data 0 inserting element with data 1999 inserting element with data 2021 inserting element with data 2020 printing list: 2000 2018 2001 1998 0 1999 2021 2020  printing count: 8 ... |

Yay! You have completed your first linked list function!

Please paste your completed function here. Good job!

|  |
| --- |
| void printCount(Node\* listHead) {  printf("printing count: ");  int num = 0;  Node \*current = listHead;  while (current != NULL) {  num++;  current = current->next;  }    printf("%d\n", num);    // TODO: implement  } |

## Problem 2

Please try to implement the unimplemented **deleteFirst** function in the Demo 2 program.

The deleteFirst function, as the name suggests, deletes the first element of the list. To do this, **simply point the listHead to the second element in the list**! You may need to handle the special case of listHead pointing to NULL (i.e. an empty list).

If done correctly, your output should look like this:

|  |
| --- |
| inserting first element with data 2000 inserting element with data 2018 printing list: 2000 2018  inserting element with data 2001 inserting element with data 1998 inserting element with data 0 inserting element with data 1999 inserting element with data 2021 inserting element with data 2020 printing list: 2000 2018 2001 1998 0 1999 2021 2020  printing count: 8 deleting first element from list:  printing list: 2018 2001 1998 0 1999 2021 2020  ... |

Please paste your completed function here. Good job!

|  |
| --- |
| Node\* deleteFirst(Node\* listHead) {  if(listHead == NULL)  printf("It is an empty list!\n");  else {  printf("deleting first element from list: \n");  listHead = listHead->next;  }  // TODO: implement  return listHead;  } |

## Problem 3

Please try to implement the second unimplemented function **deleteLast** that deletes last element in the list. It will be slightly harder than deleteFirst.

*Hint: If a node's next pointer points to NULL, it's the last node*

*Hint 2: To delete the last node, we will point the second last element's next pointer to NULL*

A1

A2

A3

A4

A5

NULL

If successful, you will see that when we call that in main, it will delete the last element and have the following output

|  |
| --- |
| inserting first element with data 2000 inserting element with data 2018 printing list: 2000 2018  inserting element with data 2001 inserting element with data 1998 inserting element with data 0 inserting element with data 1999 inserting element with data 2021 inserting element with data 2020 printing list: 2000 2018 2001 1998 0 1999 2021 2020  printing count: 8 deleting first element from list:  printing list: 2018 2001 1998 0 1999 2021 2020  deleting last element from list:  printing list: 2018 2001 1998 0 1999 2021  ... |

Please paste the finished function below. It will be great if it can handle empty list or list with only one element.

|  |
| --- |
| Node\* deleteLast(Node\* listHead) {  if(listHead == NULL)  printf("It is an empty list!\n");  else if(listHead->next == NULL)  listHead = NULL;  else {  printf("deleting last element from list: \n");  Node \*current = listHead;  while (current != NULL && current->next != NULL) {  if(current->next->next == NULL)  current->next = NULL;  current = current->next;    }  }  // TODO: implement    return listHead;  } |

## Problem 4

Please try to **implement the *deleteSpecific* function** for the linked list. Here, we will specify a number data and the *deleteSpecific* function will delete the Node with the specified number data. Notice that:

1. We assume the number data is **always** in the list and unique in the list, for simplicity
2. Like insert, there are TWO cases for delete!

*Hint: As in the lecture notes, to delete a non-head Node, we check if the* ***next Node is to be deleted****, and bypass it if so…*

*Hint 2: Your answer to last two problems is useful!*

A1

A2

A3

A4

A5

NULL

If you're successful, you will notice the output will become:

|  |
| --- |
| inserting first element with data 2000 inserting element with data 2018 printing list: 2000 2018  inserting element with data 2001 inserting element with data 1998 inserting element with data 0 inserting element with data 1999 inserting element with data 2021 inserting element with data 2020 printing list: 2000 2018 2001 1998 0 1999 2021 2020  printing count: 8 deleting first element from list:  printing list: 2018 2001 1998 0 1999 2021 2020  deleting last element from list:  printing list: 2018 2001 1998 0 1999 2021  deleting 1998 from list  printing list: 2018 2001 0 1999 2021  deleting 2018 from list  printing list: 2001 0 1999 2021 // problem 4 errata |

Please paste the completed function below:

|  |
| --- |
| Node\* deleteSpecific(Node \*listHead, int dataValue) {  // assume data to delete always exists and is always unique in  // this exercise    printf("deleting %d from list \n", dataValue);  Node \*current = listHead;  while (current != NULL && current->next != NULL && current->next->next != NULL)  {  if(current->next->data == dataValue)  current->next = current->next->next;  if(current->data == dataValue)  listHead = listHead->next;  current = current->next;  }  // TODO: implement!    return listHead;  } |

## Problem 5

Please add code to the linked list program so that our linked list become a **doubly linked list**!

In a simple linked list, the direction of traverse is one way only, from head to tail. The result is that to find a previous node of N, we need to traverse from head to N-1, which is slow. By adding another pointer in each node, the list becomes a doubly linked list that makes it easy to traverse list backwards.

HEAD

A1

A2

A3

A4

A5

NULL

Next Pointer

Previous Pointer

NULL

Each node now has two pointers:

|  |
| --- |
| typedef struct node {  int data; // the data  struct node \*next; // the next pointer  struct node \*prev; // the previous pointer } Node; |

Please update the *insert* function and implement the *printBackwardsFromZero* function so that the final output is correct. The printBackwardsFromZero will print all elements, in backwards, the first zero element from the start. **Please be reminded that the list head should now have previous point to NULL!**

A successful output should look like:

|  |
| --- |
| ...  printing list in backwards from the first 0 element (or end of list):  (assume list is 3 40 100 0 1999 2021)  100 40 3 // problem 5 errata |

Please paste the completed function below:

|  |
| --- |
| typedef struct node {  int data; // just generic, meaningless data!  struct node \*next;  struct node \*prev;  } Node;  Node\* insert(Node\* listHead, int dataToInsert) {  // we need to return a new head of linkedlist because  // the head may change after insertion  if (listHead == NULL) {  // list is emtpy, let's insert our first element!  printf("inserting first element with data %d\n",dataToInsert);  Node \*newNode = malloc(sizeof(Node));  newNode->data = dataToInsert;  newNode->next = NULL;  newNode->prev = NULL;  return newNode;  } else {  // list is not empty, let's insert at the end  printf("inserting element with data %d\n",dataToInsert);  Node \*current = listHead; // CAUTION!  while (current != NULL) {  if (current->next == NULL) {  // if it's end of list...  Node \*newNode = malloc(sizeof(Node));  newNode->data = dataToInsert;  newNode->prev = current;  newNode->next = NULL;  current->next = newNode;  return listHead;  } else {  // otherwise...  current = current->next;  }  }  }    }  void printBackwardsFromZero(Node\* listHead) {  printf("printing list in backwards from the first 0 element (or end of list): ");    // TODO: implement..  printf("\n");  Node \*current = listHead;  if(current != NULL){  while(current != NULL && current->next != NULL)  current = current->next;  while(current->prev != NULL){  if(current->data == 0){  while(current->prev != NULL){  current = current->prev;  printf("%d ", current->data);  }  }  else  current = current->prev;  }  }  } |

## 

## Problem 6 [Insertion Sort] (Hard, Optional)

Please further update the *insert* function so that we will always insert a Node in a position so that the previous Node, if not NULL, is always larger and the next Node, if not NULL, is always smaller.

If you are successful, you have implemented a sorting method called INSERTION SORT! **This exercise is a bit harder and optional, so I will not give too much hints.**

A successful output should look like:

|  |
| --- |
| inserting first element with data 2000 inserting element with data 2018 printing list: 2018 2000  inserting element with data 2001 inserting element with data 1998 inserting element with data 0 inserting element with data 1999 inserting element with data 2021 inserting element with data 2020 printing list: 2021 2020 2018 2001 2000 1999 1998 0 ← sorted  ... |

Please paste the changed function below:

|  |
| --- |
|  |

## 