Written Homework #2

CS 163: Data Structures

Alves Silva, Otavio Augusto

PSU ID: 902840168

**1. Recursion**

Above we have a recursive function to sum all data in a Linear Linked List.

**int** list::sum\_total(node \* head)

{

**if**(!head)

**return** 0;

**else**

**return** head->data + sum\_total(head->next);

}

For this code, we will use the following LLL:

NULL

Data 15

Data 10

Data 5

HEAD

After call this function by the ADT *list*, we will have the beginning of the box trace. At the point A of the code a recursive call is made, and the new invocation of the function sum\_total begins execution.

head->data= 10

A: sum\_total(Head->next)=?

return ?

head->data= 15

A: sum\_total(Head->next)=?

return ?

At point A a recursive call is made, and the new invocation of the function sum\_total begins execution:

head->data= 10

A: sum\_total(Head->next)=?

return ?

head->data= 15

A: sum\_total(Head->next)=?

return ?

head->data= 5

A: sum\_total(Head->next)=?

return ?

At point A a recursive call is made, and the new invocation of the function sum\_total begins execution:

head->data= 10

A: sum\_total(Head->next)=?

return ?

head->data= 15

A: sum\_total(Head->next)=?

return ?

head->data= 5

A: sum\_total(Head->next)=?

return ?

head->next = NULL

return 0

The base case was reached (head = NULL), so this invocation of sum\_total completes and return a value to the caller:

head->data= 10

A: sum\_total(Head->next)=?

return ?

head->data= 15

A: sum\_total(Head->next)=?

return ?

head->data= 5

A: sum\_total(Head->next)= **0**

return **0 + 5**

head->next = NULL

return 0

The method value is returned to calling box, which continues execution:

head->data= 10

A: sum\_total(Head->next)= **5**

return **10 + 5**

head->data= 15

A: sum\_total(Head->next)=?

return ?

head->data= 5

A: sum\_total(Head->next)=0

return 5

The method value is returned to calling box, which continues execution:

head->data= 10

A: sum\_total(Head->next)=5

return 15

head->data= 15

A: sum\_total(Head->next)=**15**

return **15 + 15**

The current invocation of sum\_total completes and returns the value to the initial call.

head->data= 15

A: sum\_total(Head->next)=**15**

return **30**

The value 30 is returned.

**2. Ethics**

My ethics says that I have to say the truth, even if he is my friend. However, I can’t end with the expectations of a future career or job of a friend just because of some arguments. I cannot be his judge. Therefore, the best thing to do is not lie and says just that there isn't anything to say.

**3. Algorithm**

Algorithm to copy a Circular Linked List of integers.

After the list that will be copied was populated.

List from.list

Execute the following algorithm:

Start

If the from.head node from the from.list, which will be copied, is NULL

Return a failed message

Else

Create an auxiliary pointer and point to the from.head

from.current = from.head;

Create and allocate memory for a head node of the copy list

head = new node

Copy data from the from.head

head -> data= from.head -> data

Point the next pointer of copy head to him-self

head -> next = head

Create an auxiliary node and point to thehead value of the copy list

current = head

If the next pointer of from.current isn’t from.head

Loop – While the next pointer from.current isn’t from.head.

Create a new node and linking the list

current->next = new node

current = current->next

Copy data from the from.list

current -> data = from.current->next->data

Traverse the from.list

from.current = from.current -> next

End Loop

At the final of the list, link the last node with the his head

current->next = head

End

**4. Experiencing Linux**

After compile the program using the debugging flag (-g).

* **Locate a segmentation fault**

To locate a seg fault you will probably have a signal SIGSEV. Therefore, we have to open the backtrace and see which frame occurred the segmentation fault.

* **Display the contents of a data member**

Using the command: *print expression*. Print the value of a variable or expression.

* **Backtrace**

A backtrace is a summary of how your program got where it is. For example after the segmentation fault the backtrace can be used to see what happen exactly. We can use this tipping the command *backtrace* in the gdb.