A4002 - Algorithms, Data Structures and Problem Solving

-----------------------------------------------------------------------------------------------------------------

**Lab4 - Group 1**

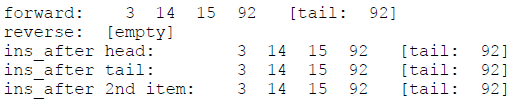
-----------------------------------------------------------------------------------------------------------------

Eric Heindl

Esteban Rueda

1. **singly liked list - list.c**

*As a starting point, create a project from the file list.c which contains structures and functions to implement and test a simply linked list. The code is ready to run and should produce the following output:*

**

*Before continuing, make sure to understand the append() function and what main() does to produce this output.*

at compiling err.h is not found -> comment function err() in the code

**

* 1. *Make a diagram of the operations needed to prepend an element to a simply linked list. Then implement the prepend() function starting on line 102.*

void prepend (List \* list, int data)

{

Item \* item;

item = malloc (sizeof \*item);

if (NULL == item) {

//err (EXIT\_FAILURE, "%s: malloc", \_\_func\_\_);

printf("malloc failed");

exit(1);

}

item->data = data;

item->next = NULL;

if (NULL == list->tail) {

list->head = item;

list->tail = item;

}

else {

item->next = list->head;

list->head = item;

}

* 1. *Make a diagram of the operations needed to insert an element into a simply linked list. Then implement the ins\_after() function starting on line 124 of the original file (the line number will have changed due to the space taken by your implementation of the prepend function).*

*void ins\_after (List \* list, Item \* pos, int data)*

*{*

*Item \* item;*

*item = malloc (sizeof \*item);*

*item->data = data;*

*item->next = pos->next;*

*pos->next = item;*

*if(item->next==NULL){list->tail=item;}*

*}*

* 1. *Make a diagram of the changes needed to remove the head of a simply linked list. Then implement the rem\_head() function (line 140 in the original source file) and uncomment the while loop at the end of main() (line 228 in the original file) in order to test your implementation.*

void rem\_head (List \* list)

{

if (NULL != list->head) {

Item \* item;

item = malloc (sizeof \*item);

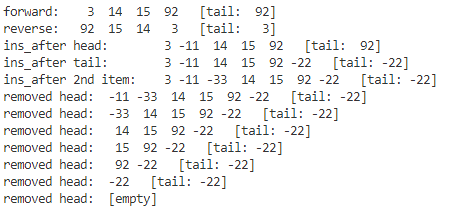
item =list->head;

list->head=list->head->next;

free(item);

}

}



1. **list-based stack - stack.c**

*The source file stack.c is a partial implementation for a program that uses a stack to print its command-line arguments in reverse.*

*It is based on the same idea as exercise 3.3: push each argument onto the stack, and then pop them off while printing them one by one.*

*(But exercise 3.3 used a vector-backed stack, and it used lines in a text file instead of command line arguments.)*

*Complete this program based on what you learned in exercise 4.1 and the comments that you find in the source file.*

*Notice that the push() function is practically the same as prepend() in the previous exercise, and pop() will be similar to rem\_head().*

*Test your program with the following arguments:*

*one "argument two" three 4*

void push (Stack \* stack, char const \* string)

{

Item \* item;

item = malloc (sizeof \*item);

if (NULL == item) {

printf("malloc failed");

exit(1);

}

item->string = string;

item->next = NULL;

if (NULL == stack->head) {

stack->head = item;

printf("%s\n",item->string);

}

else {

item->next = stack->head;

stack->head=item;

}

}

char const \* pop (Stack \* stack)

{

if (NULL != stack->head) {

Item \* item;

item = malloc (sizeof \*item);

item =stack->head;

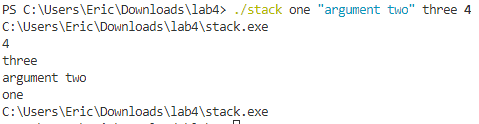
stack->head=stack->head->next;

return item->string;

}

return NULL;

}



1. **doubly linked list - dlist.c**

*The file dlist.c is a partial implementation of a program to demonstrate doubly-linked lists.It is ready to compile and run but is missing three crucial functions: rem\_item() to remove a given item from the list, ins\_after() to insert something so that it comes after a given item, and ins\_before() which inserts something in front of the given item.*

* 1. *Implement rem\_item().*

void rem\_item (List \* list, Item \* item)

{

if(list->tail!=item){

item->next->prev=item->prev;

}

else{

list->tail=item->prev;

}

if(list->head!=item){

item->prev->next=item->next;

}

else{

list->head=item->next;

}

free(item);

}

* 1. *Implement ins\_after().*

*void ins\_after (List \* list, Item \* pos, int data)*

*{*

*Item \* item;*

*item= malloc(sizeof\*item);*

*item->data=data;*

*item->next=pos->next;*

*item->prev=pos;*

*if(pos!=list->tail){pos->next->prev=item;}*

*else{list->tail=item;}*

*pos->next=item;*

*}*

* 1. *Implement ins\_before().*

*void ins\_before (List \* list, Item \* pos, int data)*

*{*

*Item \* item;*

*item= malloc(sizeof\*item);*

*item->data=data;*

*item->next=pos;*

*item->prev=pos->prev;*

*if(pos!=list->head){pos->prev->next=item;}*

*else{list->head=item;}*

*pos->prev=item;*

*}*

