Andre Godinez

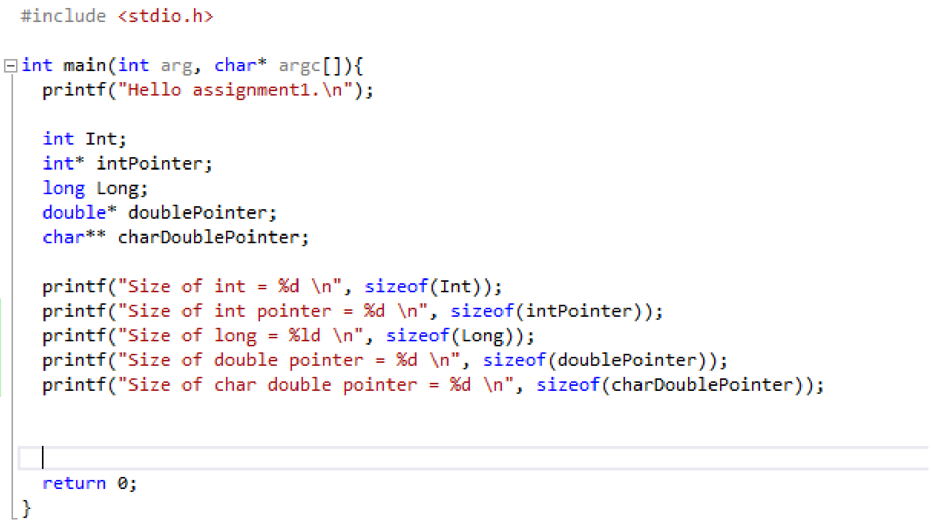
15460718

https://github.com/Ajguy97/ct331_assignment1

Question 1:

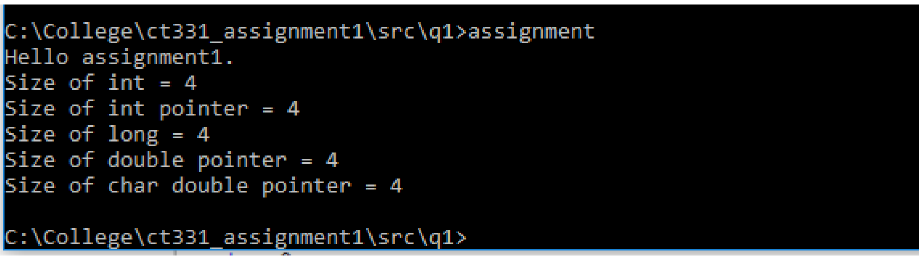
(a)

Screen shot of code:



```
#include <stdio.h>

int main(int arg, char* argc[]){
    printf("Hello assignment1.\n");

    int Int;
    int* intPointer;
    long Long;
    double* doublePointer;
    char** charDoublePointer;

    printf("Size of int = %d \n", sizeof(Int));
    printf("Size of int pointer = %d \n", sizeof(intPointer));
    printf("Size of long = %ld \n", sizeof(Long));
    printf("Size of double pointer = %d \n", sizeof(doublePointer));
    printf("Size of char double pointer = %d \n", sizeof(charDoublePointer));


    return 0;
}
```

Screen shot of cmd line output:



```
C:\College\ct331_assignment1\src\q1>assignment
Hello assignment1.
Size of int = 4
Size of int pointer = 4
Size of long = 4
Size of double pointer = 4
Size of char double pointer = 4

C:\College\ct331_assignment1\src\q1>
```

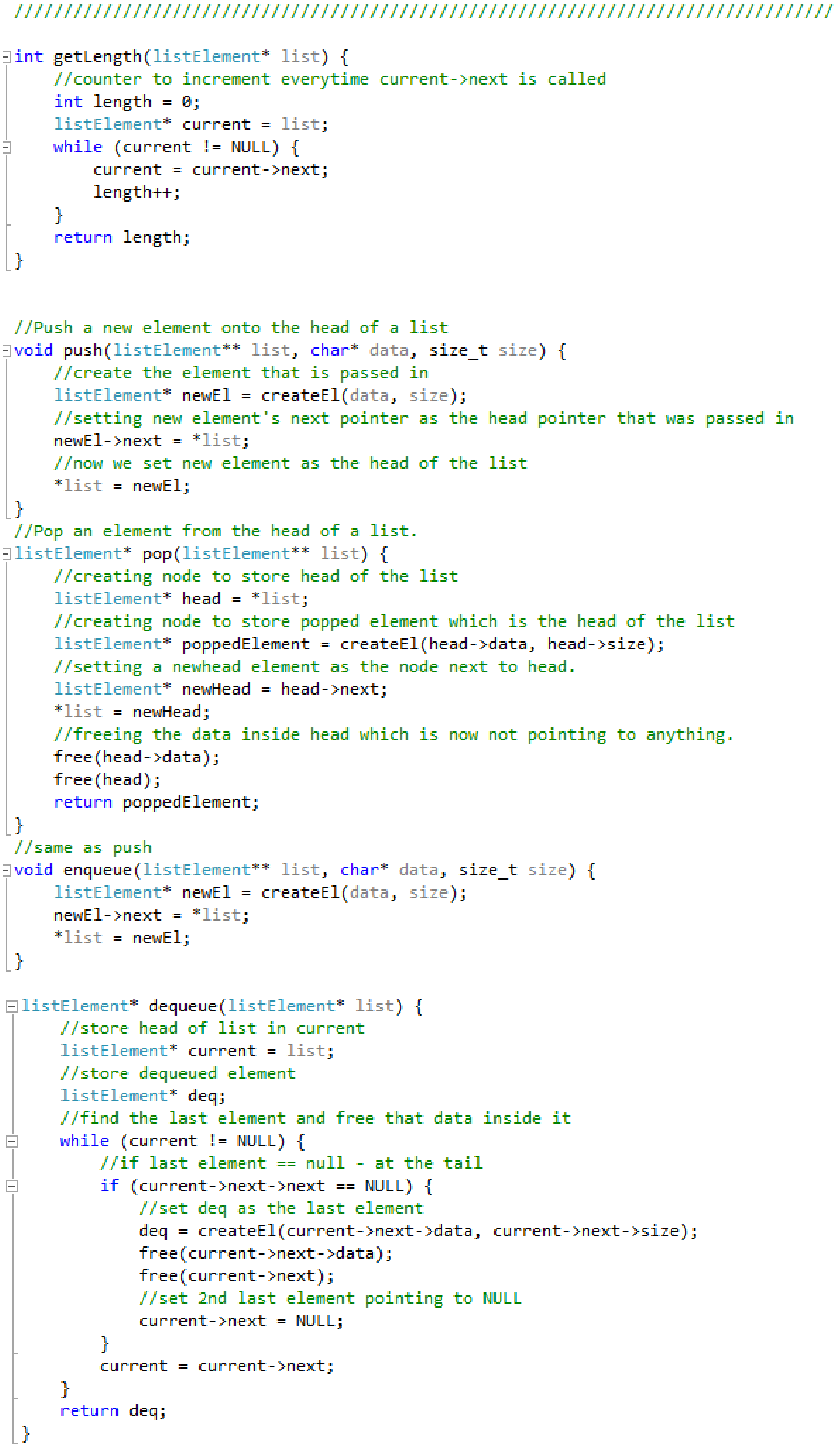
(b) Comment on results.

All data types shown have the same size. For int and long and other data types sizes are either 4 or 8 bytes depending on cpu architecture 32 bit on 64 bit and compiler settings. The same for pointers where the size is platform dependent on the computers processor architecture and the amount of bits you compile with. In this case my cpu is 64 bit but my visual studio is compiling in 32 bits therefore the sizes of all the pointers are 4. (in most cases 32 bits have size 4 bytes and 64 bits have 8 bytes.)

Question 2:

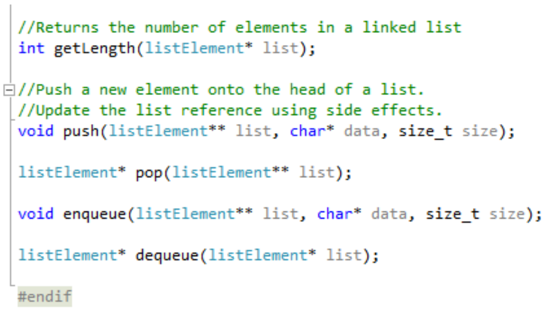
Screen shot of code :

Q2: Linkedlist.c added code



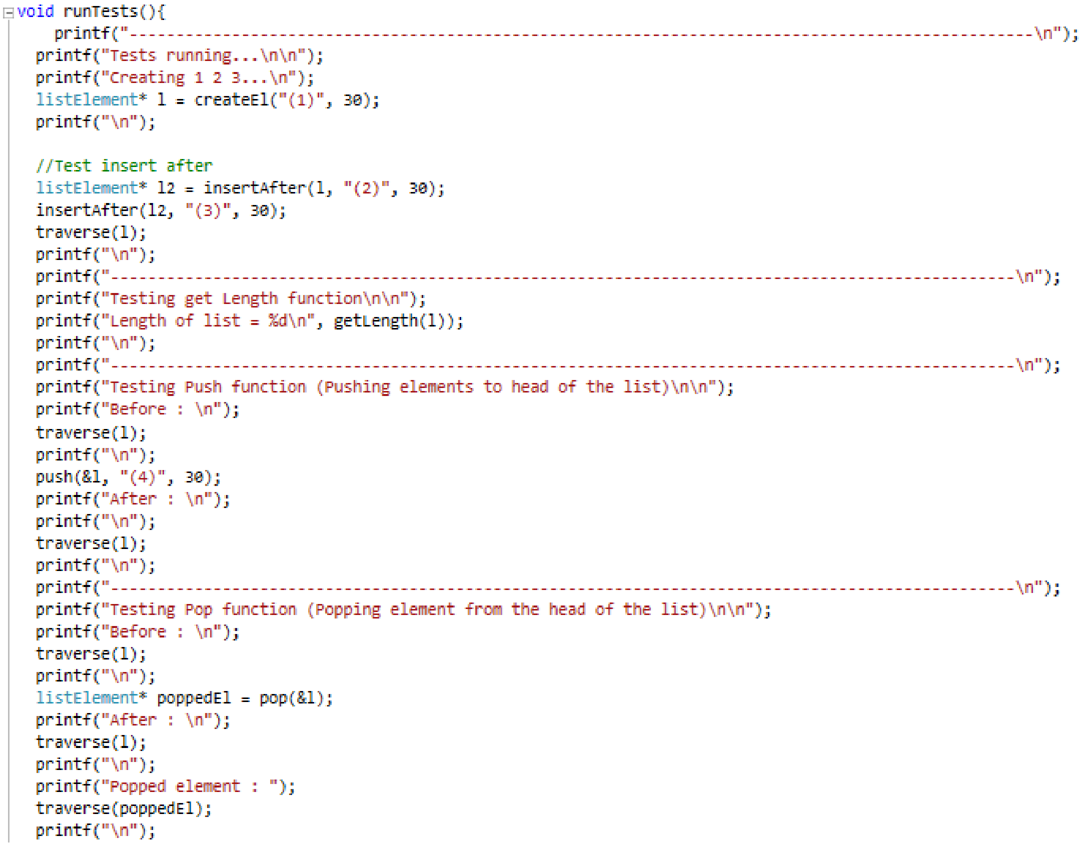
```c
//////////////////////////////////////////////////////////////////////////////////

int getLength(listElement* list) {
    //counter to increment everytime current->next is called
    int length = 0;
    listElement* current = list;
    while (current != NULL) {
        current = current->next;
        length++;
    }
    return length;
}


//Push a new element onto the head of a list
void push(listElement** list, char* data, size_t size) {
    //create the element that is passed in
    listElement* newEl = createEl(data, size);
    //setting new element's next pointer as the head pointer that was passed in
    newEl->next = *list;
    //now we set new element as the head of the list
    *list = newEl;
}
//Pop an element from the head of a list.
listElement* pop(listElement** list) {
    //creating node to store head of the list
    listElement* head = *list;
    //creating node to store popped element which is the head of the list
    listElement* poppedElement = createEl(head->data, head->size);
    //setting a newhead element as the node next to head.
    listElement* newHead = head->next;
    *list = newHead;
    //freeing the data inside head which is now not pointing to anything.
    free(head->data);
    free(head);
    return poppedElement;
}
//same as push
void enqueue(listElement** list, char* data, size_t size) {
    listElement* newEl = createEl(data, size);
    newEl->next = *list;
    *list = newEl;
}

listElement* dequeue(listElement* list) {
    //store head of list in current
    listElement* current = list;
    //store dequeued element
    listElement* deq;
    //find the last element and free that data inside it
    while (current != NULL) {
        //if last element == null - at the tail
        if (current->next->next == NULL) {
            //set deq as the last element
            deq = createEl(current->next->data, current->next->size);
            free(current->next->data);
            free(current->next);
            //set 2nd last element pointing to NULL
            current->next = NULL;
        }
        current = current->next;
    }
    return deq;
}
```

Q2: Linkedlist.h added code

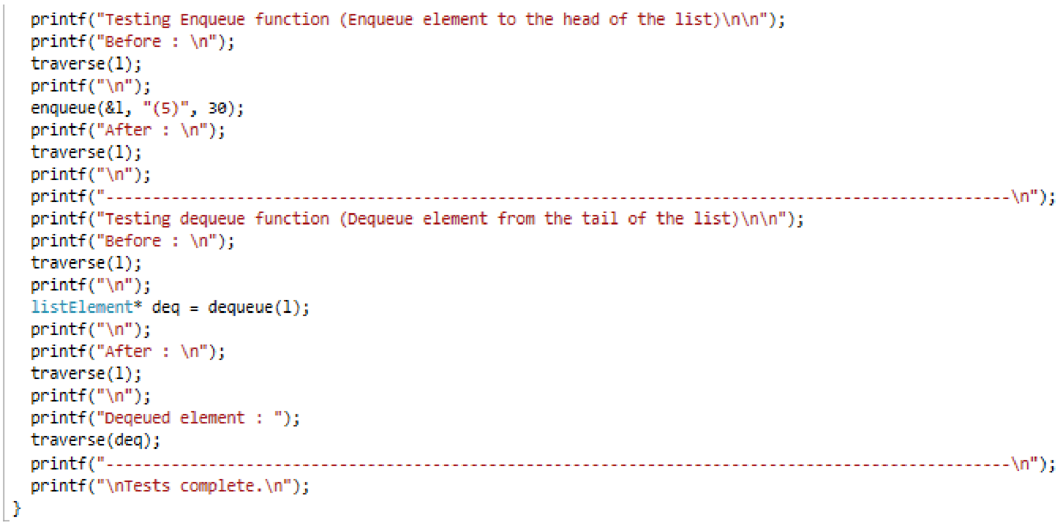


```c
//Returns the number of elements in a linked list
int getLength(listElement* list);

//Push a new element onto the head of a list.
//Update the list reference using side effects.
void push(listElement** list, char* data, size_t size);

listElement* pop(listElement** list);

void enqueue(listElement** list, char* data, size_t size);

listElement* dequeue(listElement* list);

#endif
```

Q2 : Tests.c added code

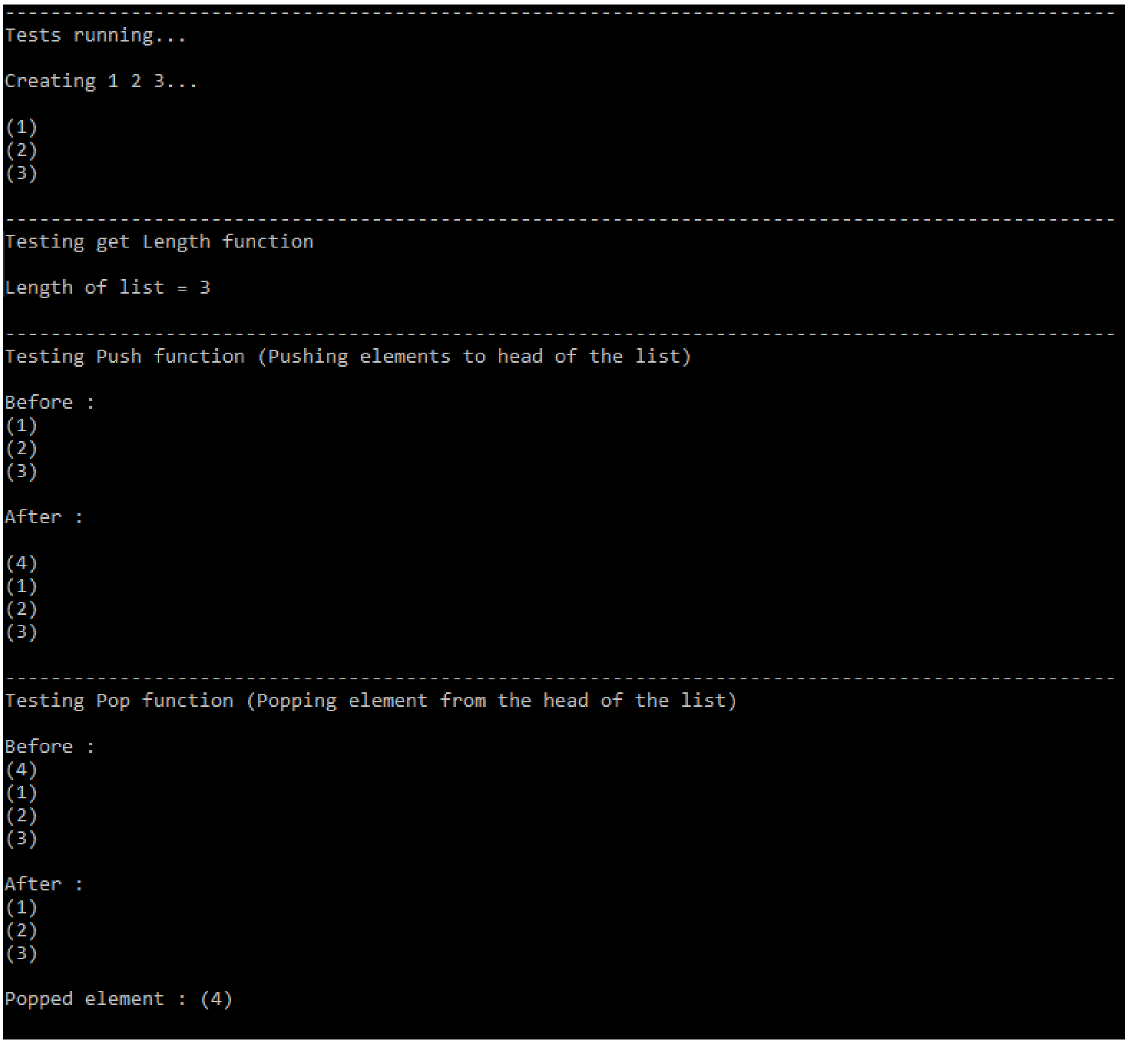


```c
void runTests(){
    printf("---------------------------------------------------------------------------------------------\n");
  printf("Tests running...\n\n");
  printf("Creating 1 2 3...\n");
  listElement* l = createEl("(1)", 30);
  printf("\n");

  //Test insert after
  listElement* l2 = insertAfter(l, "(2)", 30);
  insertAfter(l2, "(3)", 30);
  traverse(l);
  printf("\n");
  printf("---------------------------------------------------------------------------------------------\n");
  printf("Testing get Length function\n\n");
  printf("Length of list = %d\n", getLength(l));
  printf("\n");
  printf("---------------------------------------------------------------------------------------------\n");
  printf("Testing Push function (Pushing elements to head of the list)\n\n");
  printf("Before : \n");
  traverse(l);
  printf("\n");
  push(&l, "(4)", 30);
  printf("After : \n");
  printf("\n");
  traverse(l);
  printf("\n");
  printf("---------------------------------------------------------------------------------------------\n");
  printf("Testing Pop function (Popping element from the head of the list)\n\n");
  printf("Before : \n");
  traverse(l);
  printf("\n");
  listElement* poppedEl = pop(&l);
  printf("After : \n");
  traverse(l);
  printf("\n");
  printf("Popped element : ");
  traverse(poppedEl);
  printf("\n");
```

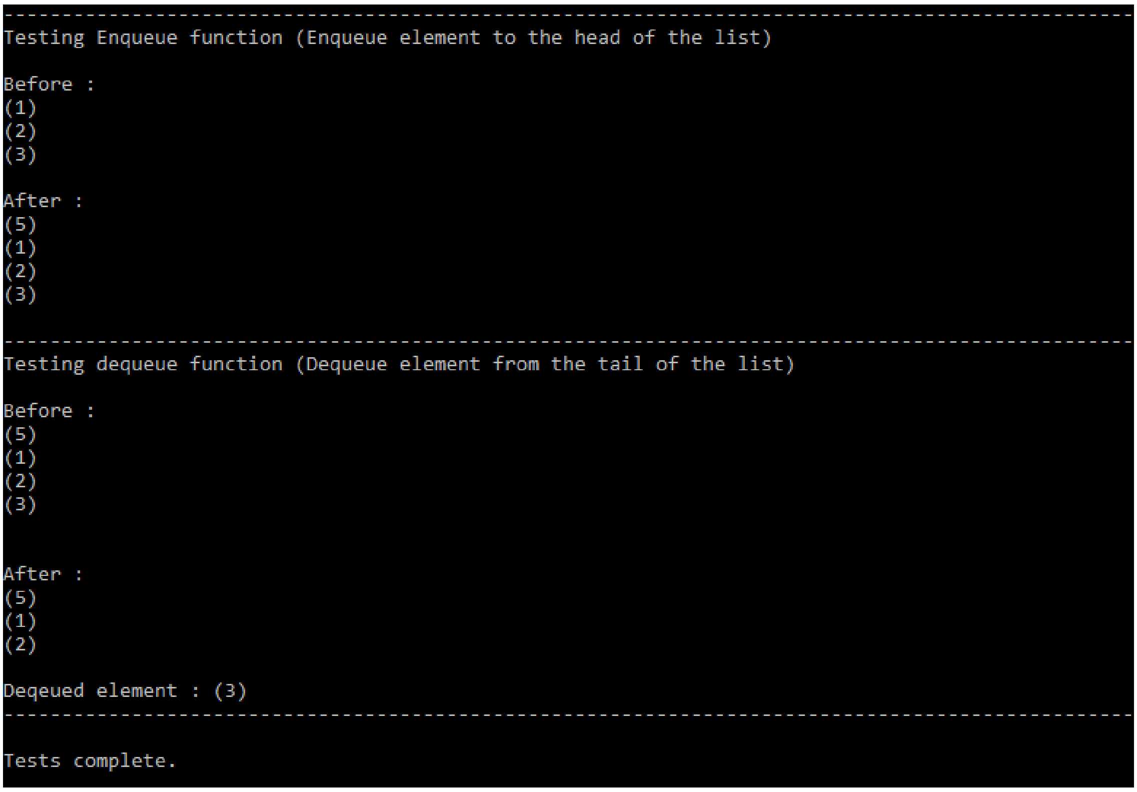


```c
  printf("Testing Enqueue function (Enqueue element to the head of the list)\n\n");
  printf("Before : \n");
  traverse(l);
  printf("\n");
  enqueue(&l, "(5)", 30);
  printf("After : \n");
  traverse(l);
  printf("\n");
  printf("---------------------------------------------------------------------------------------------\n");
  printf("Testing dequeue function (Dequeue element from the tail of the list)\n\n");
  printf("Before : \n");
  traverse(l);
  printf("\n");
  listElement* deq = dequeue(l);
  printf("\n");
  printf("After : \n");
  traverse(l);
  printf("\n");
  printf("Deqeued element : ");
  traverse(deq);
  printf("---------------------------------------------------------------------------------------------\n");
  printf("\nTests complete.\n");
}
```

Q2 : Screen shot of cmd line output:



```
------------------------------------------------------------------------------------------------
Tests running...

Creating 1 2 3...

(1)
(2)
(3)

------------------------------------------------------------------------------------------------
Testing get Length function

Length of list = 3

------------------------------------------------------------------------------------------------
Testing Push function (Pushing elements to head of the list)

Before :
(1)
(2)
(3)

After :

(4)
(1)
(2)
(3)

------------------------------------------------------------------------------------------------
Testing Pop function (Popping element from the head of the list)

Before :
(4)
(1)
(2)
(3)

After :
(1)
(2)
(3)

Popped element : (4)
```

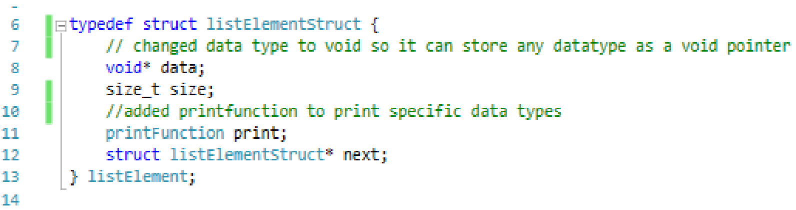


```
------------------------------------------------------------------------------------------------
Testing Enqueue function (Enqueue element to the head of the list)

Before :
(1)
(2)
(3)

After :
(5)
(1)
(2)
(3)

------------------------------------------------------------------------------------------------
Testing dequeue function (Dequeue element from the tail of the list)

Before :
(5)
(1)
(2)
(3)


After :
(5)
(1)
(2)

Deqeued element : (3)
------------------------------------------------------------------------------------------------

Tests complete.
```

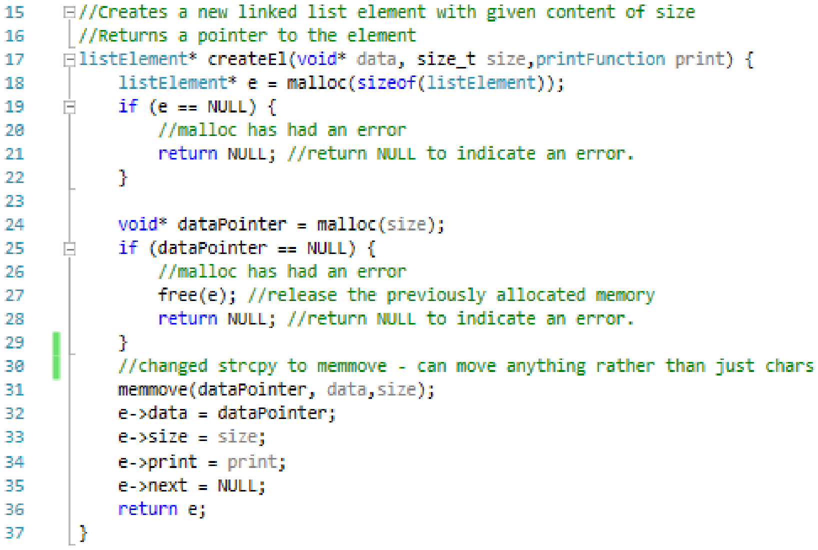
Question 3:

Screen shot of code :

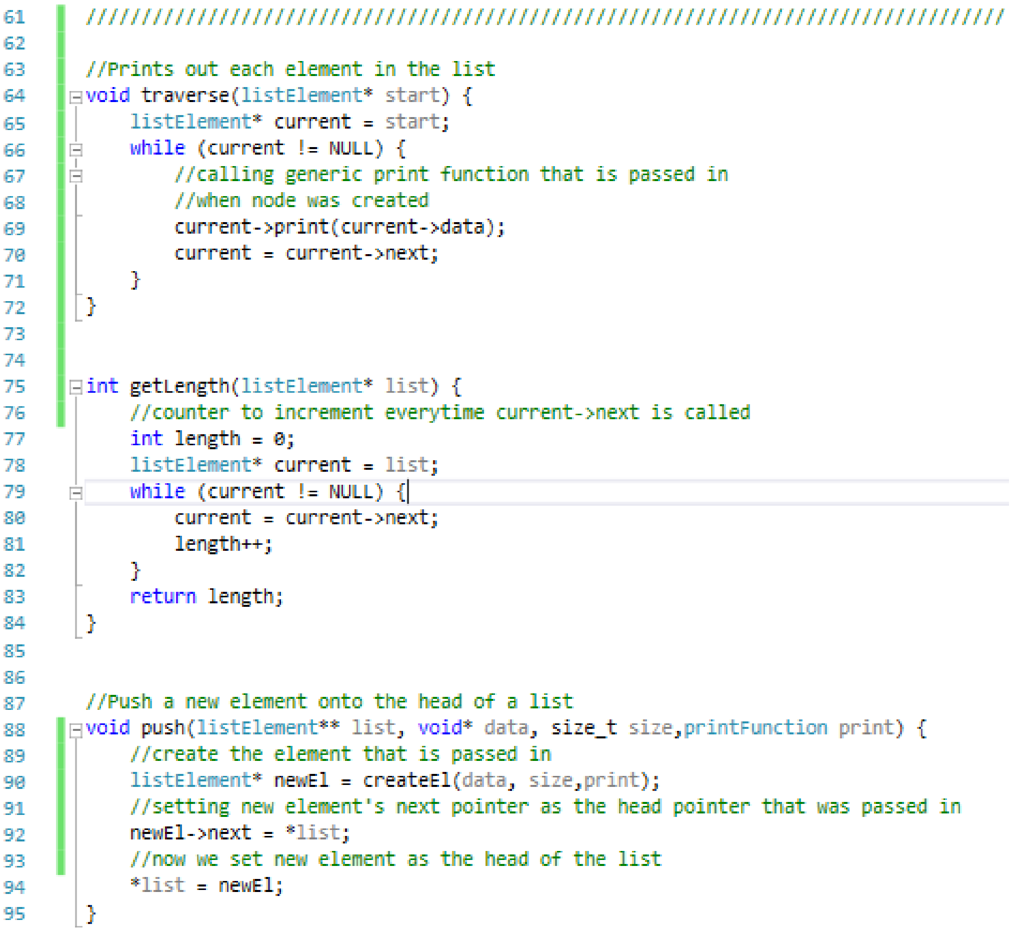
Q3 Genericlinkedlist.c added code:



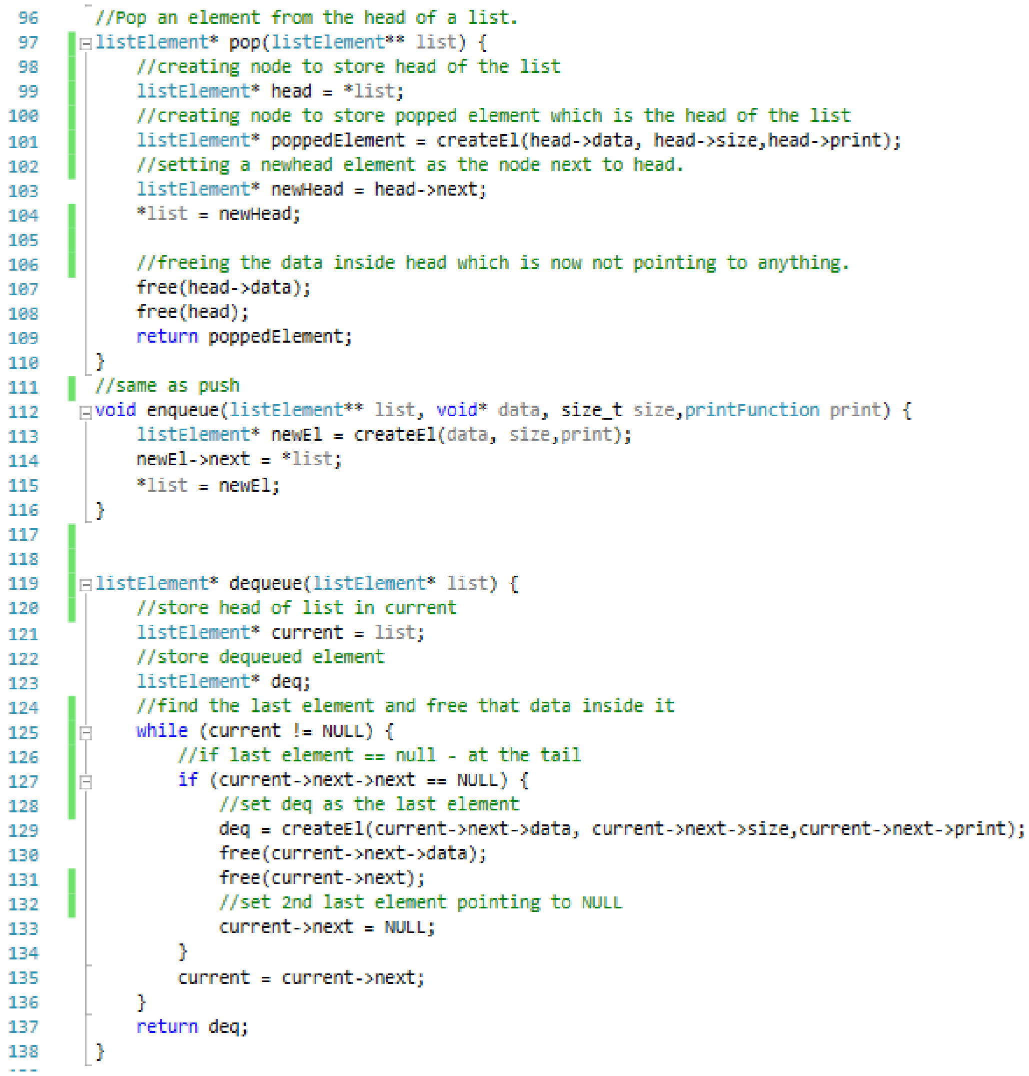
```c
typedef struct listElementStruct {
    // changed data type to void so it can store any datatype as a void pointer
    void* data;
    size_t size;
    //added printfunction to print specific data types
    printFunction print;
    struct listElementStruct* next;
} listElement;

```



```c
//Creates a new linked list element with given content of size
//Returns a pointer to the element
listElement* createEl(void* data, size_t size,printFunction print) {
    listElement* e = malloc(sizeof(listElement));
    if (e == NULL) {
        //malloc has had an error
        return NULL; //return NULL to indicate an error.
    }

    void* dataPointer = malloc(size);
    if (dataPointer == NULL) {
        //malloc has had an error
        free(e); //release the previously allocated memory
        return NULL; //return NULL to indicate an error.
    }
    //changed strcpy to memmove - can move anything rather than just chars
    memmove(dataPointer, data,size);
    e->data = dataPointer;
    e->size = size;
    e->print = print;
    e->next = NULL;
    return e;
}
```

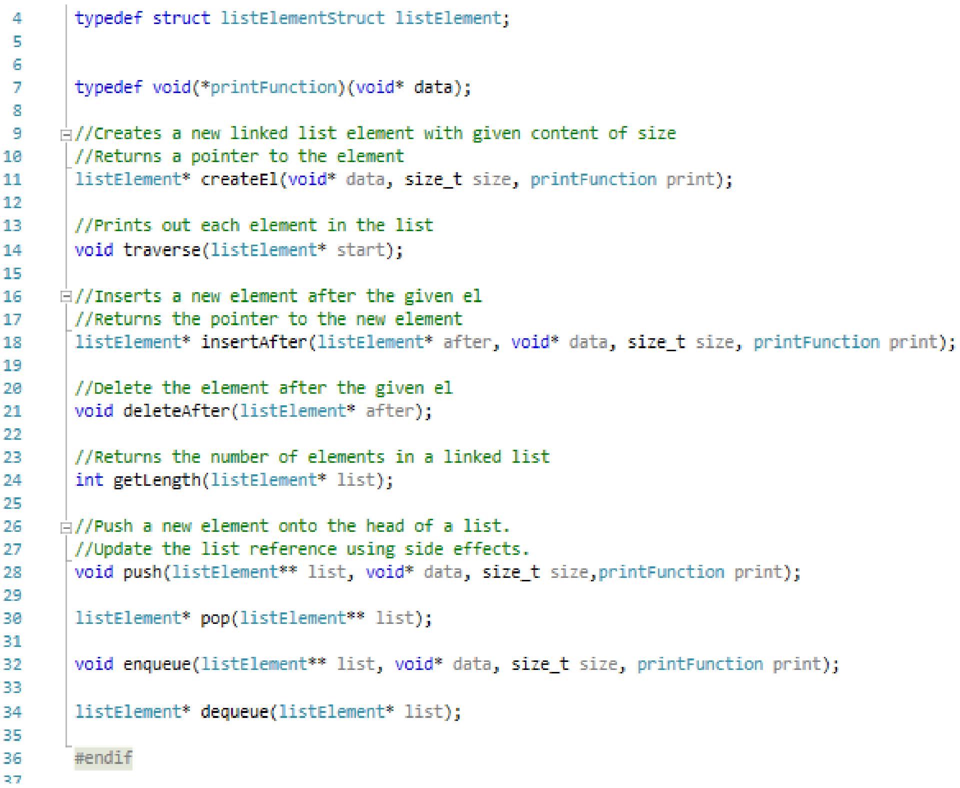


```c
///////////////////////////////////////////////////////////////////////////////////////

//Prints out each element in the list
void traverse(listElement* start) {
    listElement* current = start;
    while (current != NULL) {
        //calling generic print function that is passed in
        //when node was created
        current->print(current->data);
        current = current->next;
    }
}


int getLength(listElement* list) {
    //counter to increment everytime current->next is called
    int length = 0;
    listElement* current = list;
    while (current != NULL) {
        current = current->next;
        length++;
    }
    return length;
}


//Push a new element onto the head of a list
void push(listElement** list, void* data, size_t size,printFunction print) {
    //create the element that is passed in
    listElement* newEl = createEl(data, size,print);
    //setting new element's next pointer as the head pointer that was passed in
    newEl->next = *list;
    //now we set new element as the head of the list
    *list = newEl;
}
```



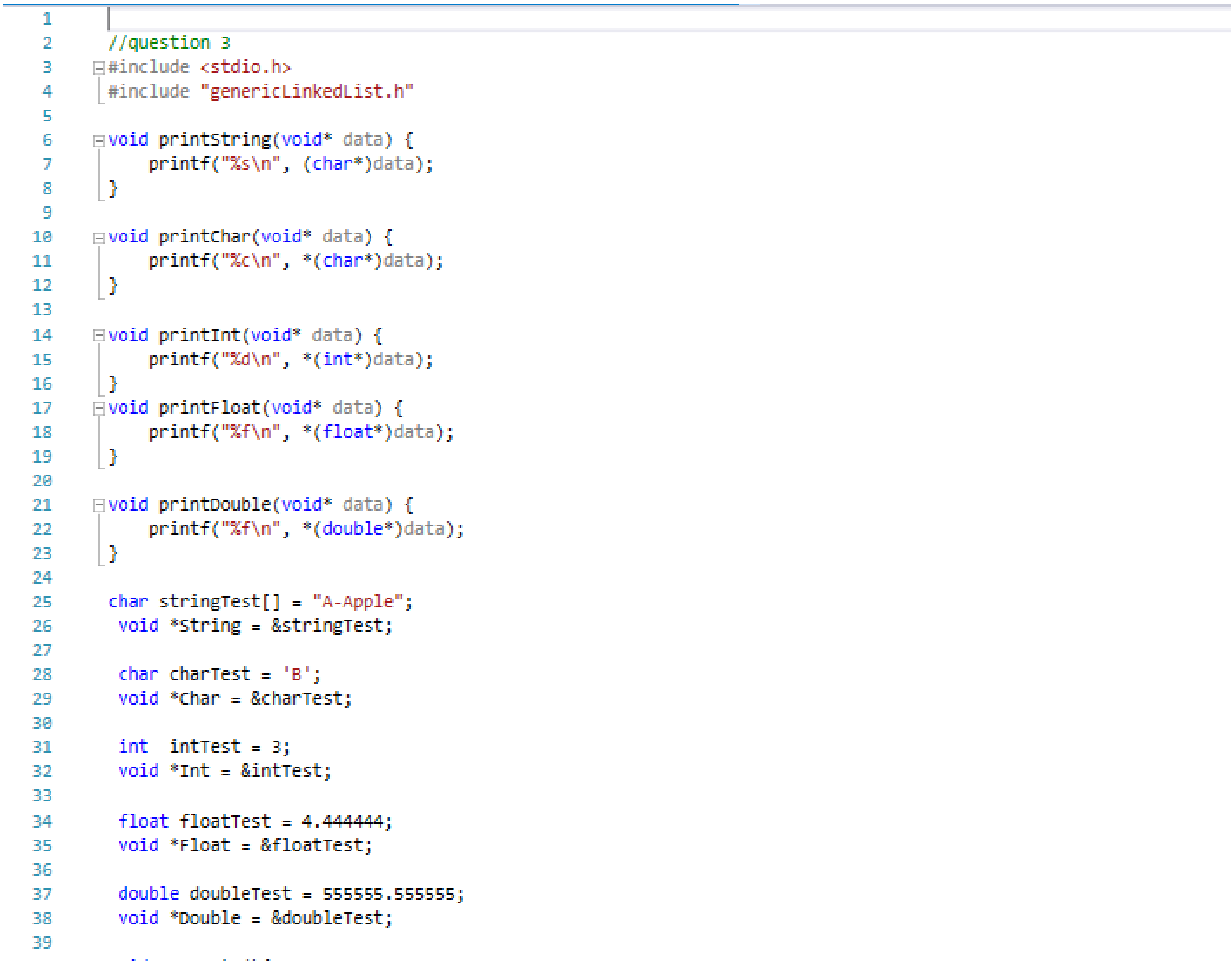
```c
//Pop an element from the head of a list.
listElement* pop(listElement** list) {
    //creating node to store head of the list
    listElement* head = *list;
    //creating node to store popped element which is the head of the list
    listElement* poppedElement = createEl(head->data, head->size,head->print);
    //setting a newhead element as the node next to head.
    listElement* newHead = head->next;
    *list = newHead;

    //freeing the data inside head which is now not pointing to anything.
    free(head->data);
    free(head);
    return poppedElement;
}
//same as push
void enqueue(listElement** list, void* data, size_t size,printFunction print) {
    listElement* newEl = createEl(data, size,print);
    newEl->next = *list;
    *list = newEl;
}


listElement* dequeue(listElement* list) {
    //store head of list in current
    listElement* current = list;
    //store dequeued element
    listElement* deq;
    //find the last element and free that data inside it
    while (current != NULL) {
        //if last element == null - at the tail
        if (current->next->next == NULL) {
            //set deq as the last element
            deq = createEl(current->next->data, current->next->size,current->next->print);
            free(current->next->data);
            free(current->next);
            //set 2nd last element pointing to NULL
            current->next = NULL;
        }
        current = current->next;
    }
    return deq;
}
```

Q3 Genericlinkedlist.h added code:

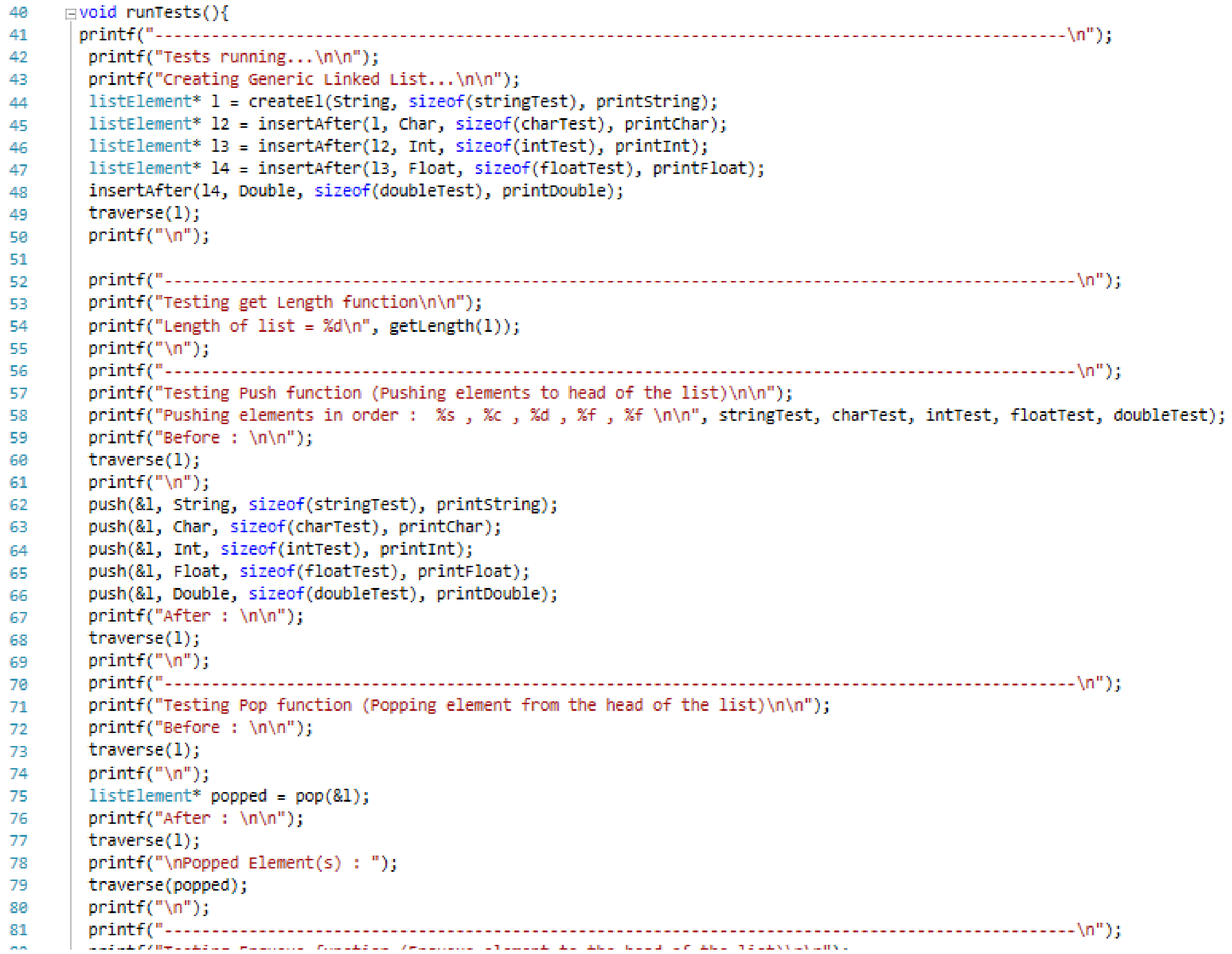


```c
typedef struct listElementStruct listElement;


typedef void(*printFunction)(void* data);

//Creates a new linked list element with given content of size
//Returns a pointer to the element
listElement* createEl(void* data, size_t size, printFunction print);

//Prints out each element in the list
void traverse(listElement* start);

//Inserts a new element after the given el
//Returns the pointer to the new element
listElement* insertAfter(listElement* after, void* data, size_t size, printFunction print);

//Delete the element after the given el
void deleteAfter(listElement* after);

//Returns the number of elements in a linked list
int getLength(listElement* list);

//Push a new element onto the head of a list.
//Update the list reference using side effects.
void push(listElement** list, void* data, size_t size,printFunction print);

listElement* pop(listElement** list);

void enqueue(listElement** list, void* data, size_t size, printFunction print);

listElement* dequeue(listElement* list);

#endif
```

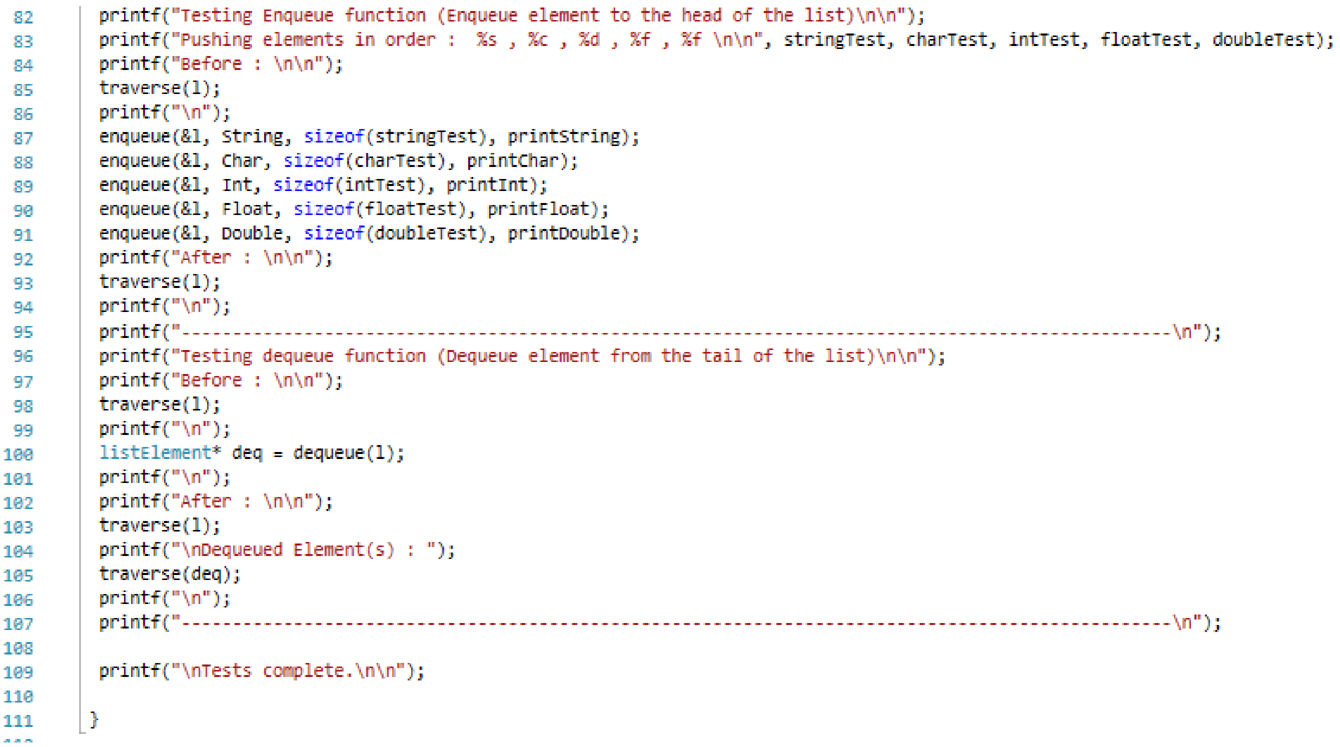
Q3 Tests.c added code:



```c

//question 3
#include <stdio.h>
#include "genericLinkedList.h"

void printString(void* data) {
    printf("%s\n", (char*)data);
}

void printChar(void* data) {
    printf("%c\n", *(char*)data);
}

void printInt(void* data) {
    printf("%d\n", *(int*)data);
}
void printFloat(void* data) {
    printf("%f\n", *(float*)data);
}

void printDouble(void* data) {
    printf("%f\n", *(double*)data);
}

char stringTest[] = "A-Apple";
 void *String = &stringTest;

 char charTest = 'B';
 void *Char = &charTest;

 int  intTest = 3;
 void *Int = &intTest;

 float floatTest = 4.444444;
 void *Float = &floatTest;

 double doubleTest = 555555.555555;
 void *Double = &doubleTest;

```

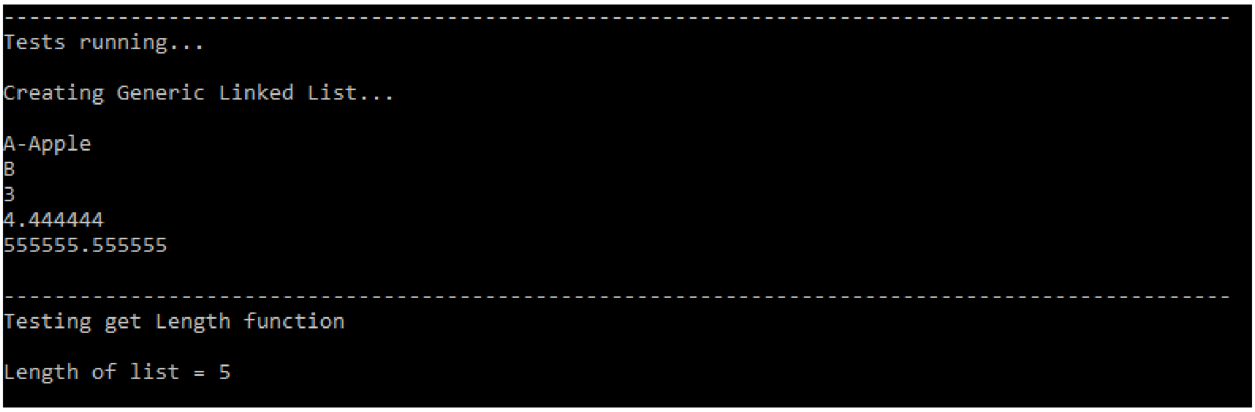


```c
void runTests(){
printf("-----------------------------------------------------------------------------------------------\n");
 printf("Tests running...\n\n");
 printf("Creating Generic Linked List...\n\n");
 listElement* l = createEl(String, sizeof(stringTest), printString);
 listElement* l2 = insertAfter(l, Char, sizeof(charTest), printChar);
 listElement* l3 = insertAfter(l2, Int, sizeof(intTest), printInt);
 listElement* l4 = insertAfter(l3, Float, sizeof(floatTest), printFloat);
 insertAfter(l4, Double, sizeof(doubleTest), printDouble);
 traverse(l);
 printf("\n");

 printf("-----------------------------------------------------------------------------------------------\n");
 printf("Testing get Length function\n\n");
 printf("Length of list = %d\n", getLength(l));
 printf("\n");
 printf("-----------------------------------------------------------------------------------------------\n");
 printf("Testing Push function (Pushing elements to head of the list)\n\n");
 printf("Pushing elements in order :  %s , %c , %d , %f , %f \n\n", stringTest, charTest, intTest, floatTest, doubleTest);
 printf("Before : \n\n");
 traverse(l);
 printf("\n");
 push(&l, String, sizeof(stringTest), printString);
 push(&l, Char, sizeof(charTest), printChar);
 push(&l, Int, sizeof(intTest), printInt);
 push(&l, Float, sizeof(floatTest), printFloat);
 push(&l, Double, sizeof(doubleTest), printDouble);
 printf("After : \n\n");
 traverse(l);
 printf("\n");
 printf("-----------------------------------------------------------------------------------------------\n");
 printf("Testing Pop function (Popping element from the head of the list)\n\n");
 printf("Before : \n\n");
 traverse(l);
 printf("\n");
 listElement* popped = pop(&l);
 printf("After : \n\n");
 traverse(l);
 printf("\nPopped Element(s) : ");
 traverse(popped);
 printf("\n");
 printf("-----------------------------------------------------------------------------------------------\n");
```

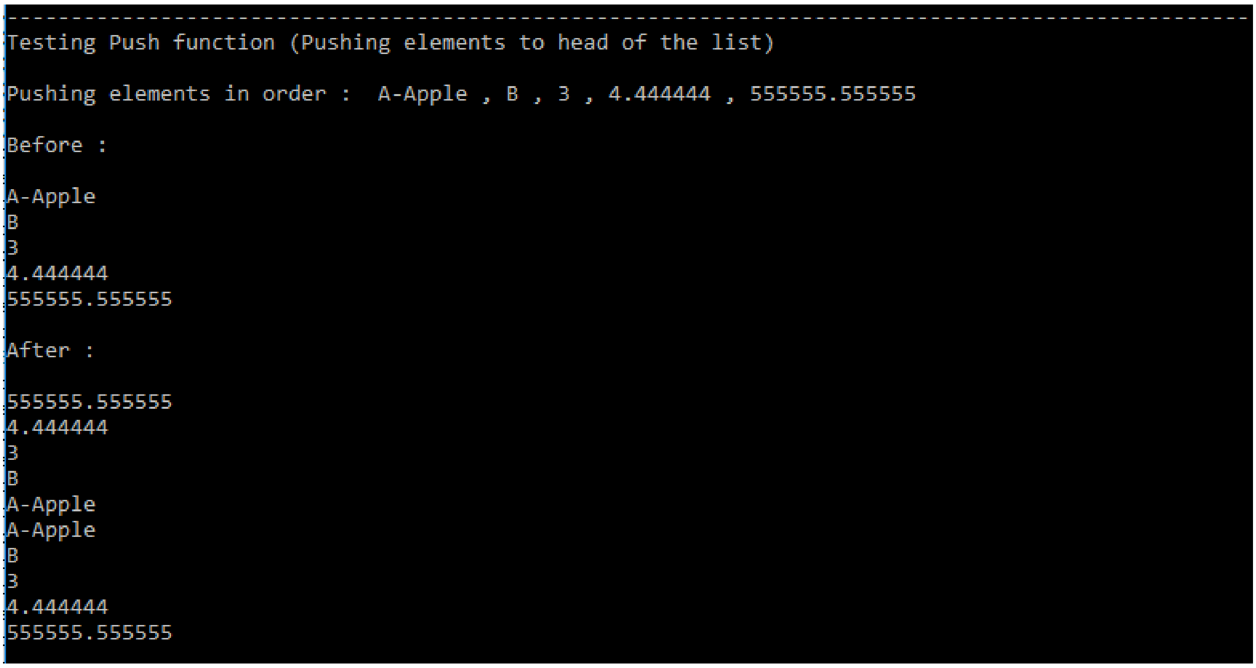


```c
    printf("Testing Enqueue function (Enqueue element to the head of the list)\n\n");
    printf("Pushing elements in order :  %s , %c , %d , %f , %f \n\n", stringTest, charTest, intTest, floatTest, doubleTest);
    printf("Before : \n\n");
    traverse(l);
    printf("\n");
    enqueue(&l, String, sizeof(stringTest), printString);
    enqueue(&l, Char, sizeof(charTest), printChar);
    enqueue(&l, Int, sizeof(intTest), printInt);
    enqueue(&l, Float, sizeof(floatTest), printFloat);
    enqueue(&l, Double, sizeof(doubleTest), printDouble);
    printf("After : \n\n");
    traverse(l);
    printf("\n");
    printf("-----------------------------------------------------------------------------------------------\n");
    printf("Testing dequeue function (Dequeue element from the tail of the list)\n\n");
    printf("Before : \n\n");
    traverse(l);
    printf("\n");
    listElement* deq = dequeue(l);
    printf("\n");
    printf("After : \n\n");
    traverse(l);
    printf("\nDequeued Element(s) : ");
    traverse(deq);
    printf("\n");
    printf("-----------------------------------------------------------------------------------------------\n");

    printf("\nTests complete.\n\n");

}
```

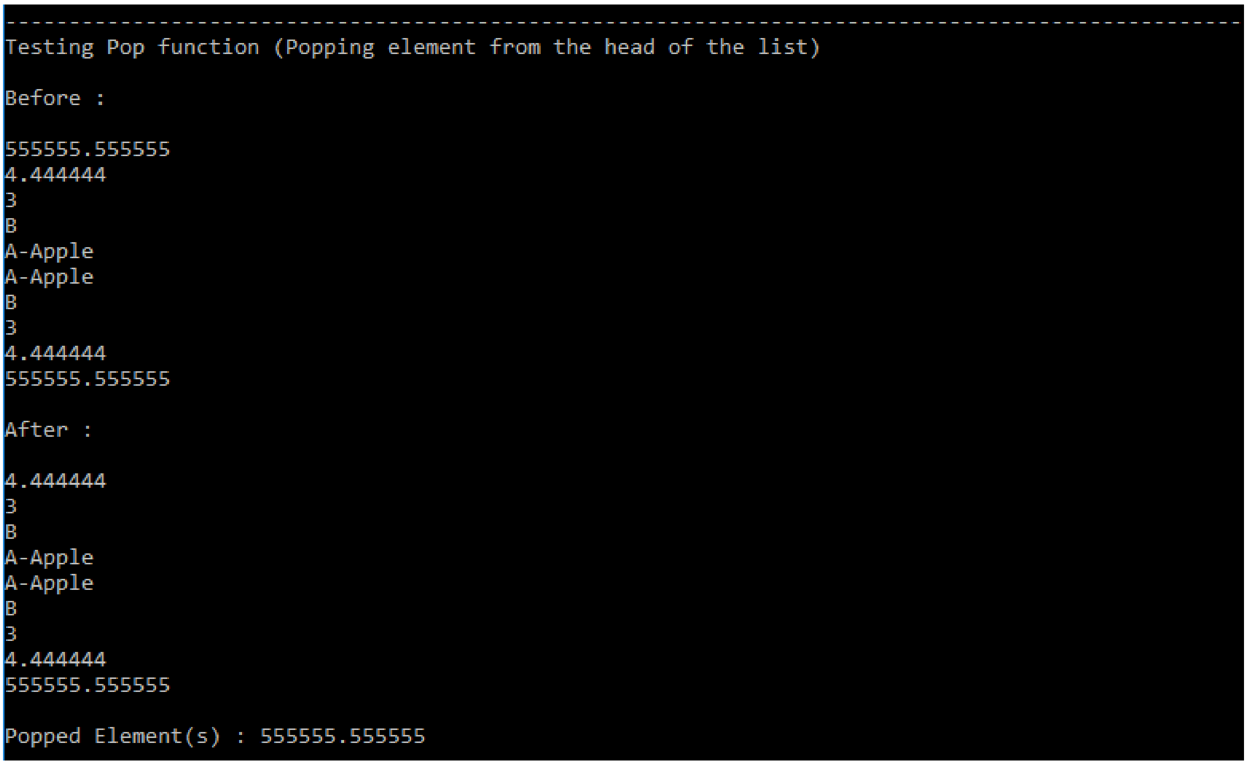
Screen shot of cmd line output:



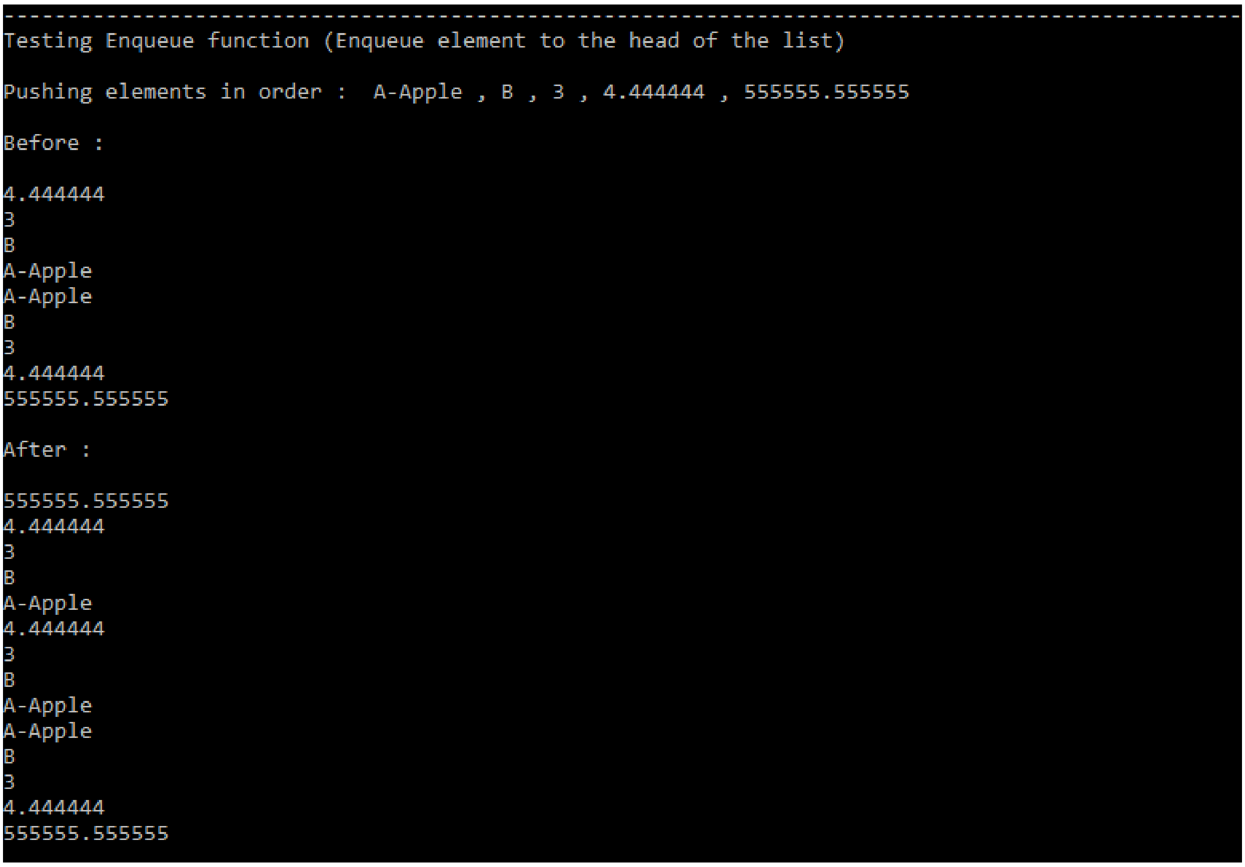
```
---------------------------------------------------------------------------------------------------
Tests running...

Creating Generic Linked List...

A-Apple
B
3
4.444444
555555.555555

---------------------------------------------------------------------------------------------------
Testing get Length function

Length of list = 5
```



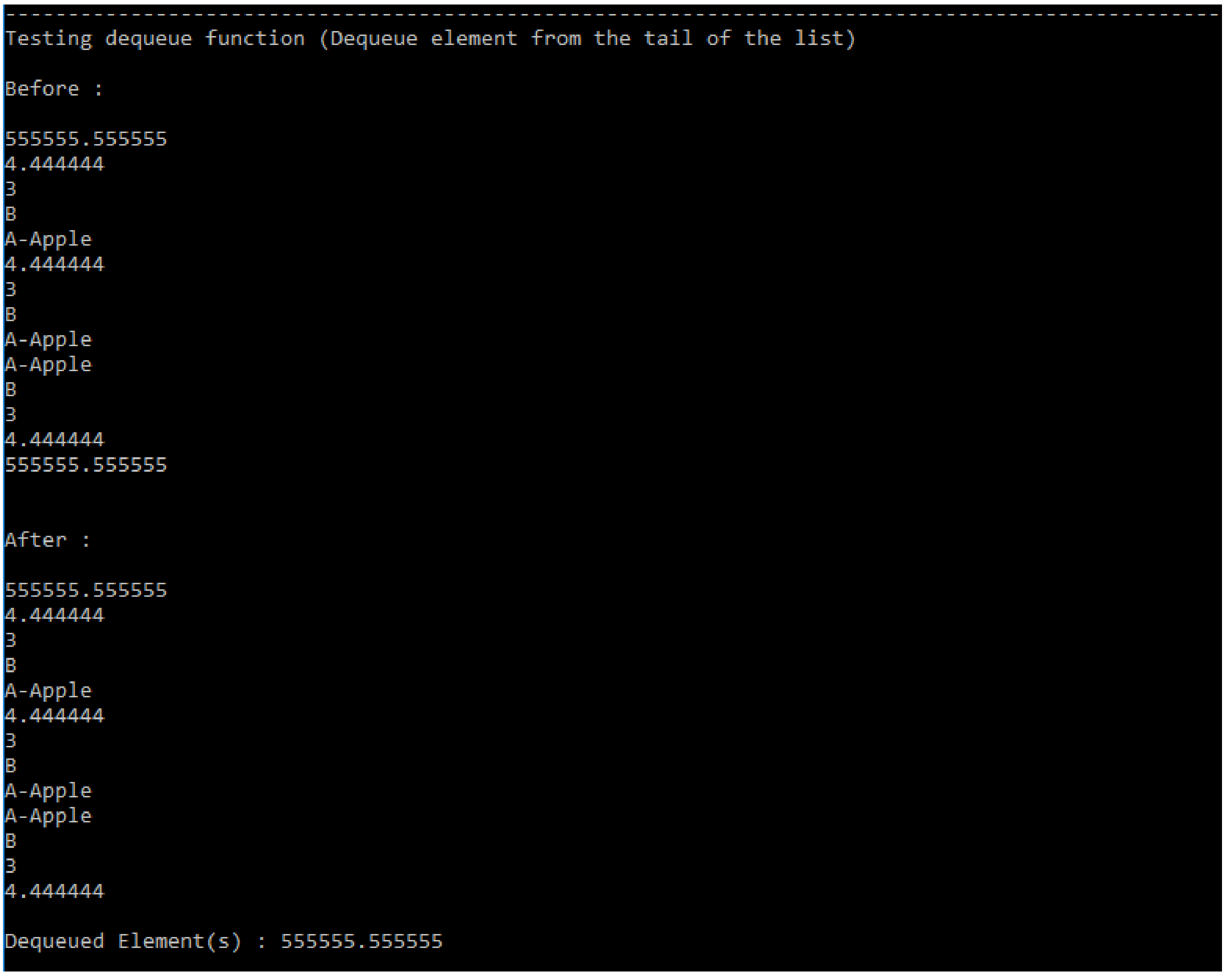
```
---------------------------------------------------------------------------------------------------
Testing Push function (Pushing elements to head of the list)

Pushing elements in order :  A-Apple , B , 3 , 4.444444 , 555555.555555

Before :

A-Apple
B
3
4.444444
555555.555555

After :

555555.555555
4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555
```



```
-----------------------------------------------------------------------------------------------
Testing Pop function (Popping element from the head of the list)

Before :

555555.555555
4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555

After :

4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555

Popped Element(s) : 555555.555555
```



```
-----------------------------------------------------------------------------------------------
Testing Enqueue function (Enqueue element to the head of the list)

Pushing elements in order :  A-Apple , B , 3 , 4.444444 , 555555.555555

Before :

4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555

After :

555555.555555
4.444444
3
B
A-Apple
4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555
```



```
----------------------------------------------------------------------------------------------------
Testing dequeue function (Dequeue element from the tail of the list)

Before :

555555.555555
4.444444
3
B
A-Apple
4.444444
3
B
A-Apple
A-Apple
B
3
4.444444
555555.555555


After :

555555.555555
4.444444
3
B
A-Apple
4.444444
3
B
A-Apple
A-Apple
B
3
4.444444

Dequeued Element(s) : 555555.555555
```

Question 4:

(a)

Memory required to traverse a linked list in reverse tail to head

- Traversing from head to tail needs 1 node pointer and call node->next to get the next node in the list until next is NULL – Linear function therefore not as memory intensive.
- Traversing from tail to head
    - Create a node – current, tail - set current as head pointer passed in.
    - While (current->next !=null) – if it's equal to null then only 1 node in linked list – return that node.

      If(current->next == NULL || current->next == tail) – that means this is the tail of the list
      - Do whatever with current -> print it or return this node
      - set tail as this node
      - set current as head pointer passed in again.

    - Current = current->next
- This function is recursive and depends on amount of nodes in the linked list therefore more memory required.

(b)

How could the structure of a linked list be changed to make this less memory intensive?

- Usage of doubly linked list.
- Node has a pointer to next & previous node.
- Allows traversal head to tail & tail to head without recursion in functions. Eg. Current->next & current->prev