Assignment 1 - Programming Paragdims

Daniel Hannon (19484286)

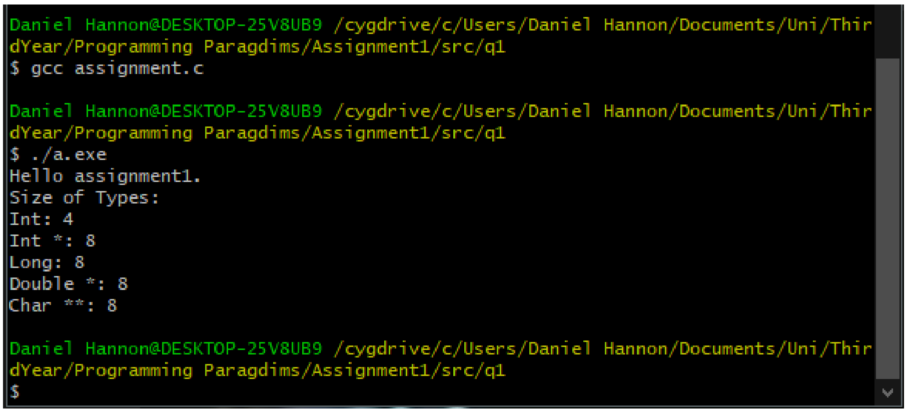
October 2021

1 Question 1

1.1 Code

```c
#include <stdio.h>
#include <stdlib.h>
/* Question 1 Code
  By Daniel Hannon(19484286)
*/
int main(int arg, char* argc[]){
  printf("Hello assignment1.\n");
  //Create required data types and fill it with dummy data
  int myInt = 5;
  int * pInt = &myInt;
  long myLong = 12;
  double * pDouble = (double *) malloc(sizeof(double));
  *pDouble = 1.2323423;
  char ** dpChar = (char **) malloc(sizeof(char *));
  *(dpChar) = (char*) malloc(sizeof(char));
  **dpChar = 'a';
  //Print the data
  printf("Size of Types:\nInt: %ld\nInt *: %ld\nLong: %ld\nDouble *: %ld\nChar **: %ld\n",sizeof(myInt),sizeof(pInt),sizeof(myLong),sizeof(pDouble),sizeof(dpChar));
  //For some reason when I try to free the pointers it SegFaults so I did not do that
  return 0;
}
```

1.2 Output



```
Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/Thir
dYear/Programming Paragdims/Assignment1/src/q1
$ gcc assignment.c

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/Thir
dYear/Programming Paragdims/Assignment1/src/q1
$ ./a.exe
Hello assignment1.
Size of Types:
Int: 4
Int *: 8
Long: 8
Double *: 8
Char **: 8

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/Thir
dYear/Programming Paragdims/Assignment1/src/q1
$
```

1.3 Comments

1. Int has size 4 as it is 4 bytes(32 bits long)
2. Long has size 8 as it is 8 bytes long (64 bytes)
3. All the pointers (Int *, Double *, Char **) have length 8 as I am using a 64bit operating system and thus all pointers reference a 64 bit address. If I were using a 32 bit operating system, the pointers would be 32 bits long

2 Question 2

2.1 Code

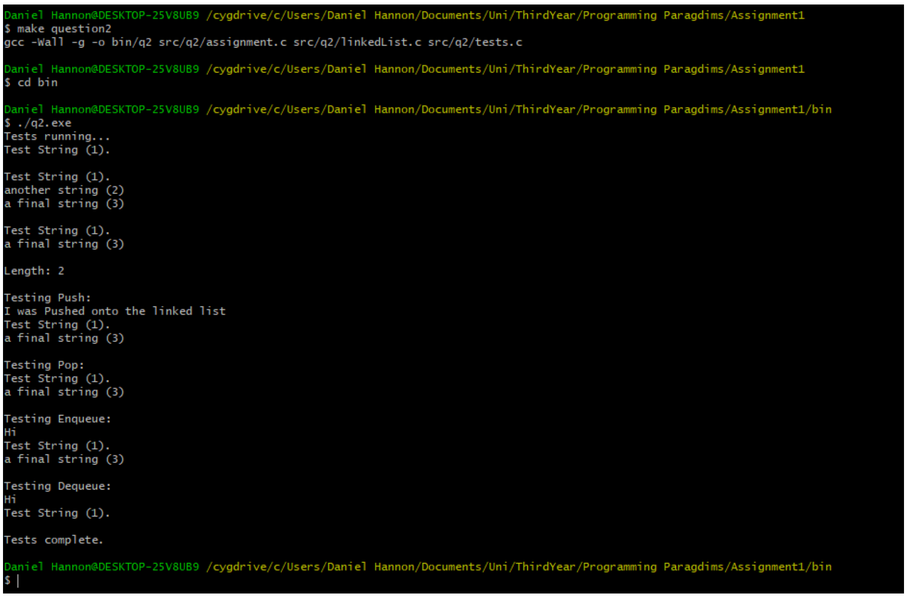
2.1.1 Code I wrote for linkedlist.h

```
int length(listElement * list) {
  int count = 0;
  listElement * temp = list;
  while(temp != NULL) {
    count++;
    temp=temp->next;
  }
  return count;
}

void push(listElement ** list, char* data, size_t size) {
  listElement* elem = createEl(data,size);
  elem->next = *list;
  *list = elem;
}

listElement* pop(listElement** list) {
  listElement* temp = *list;
  //reassign the start of the linked list to the second element
  *list = (*list)->next;
  //Remove reference to the start of the linked list
  temp->next = NULL;
  return temp;
}

void enqueue(listElement** list, char* data, size_t size) {
  //Does the same thing as push
  push(list,data,size);
}

listElement * dequeue(listElement* list) {
  //Create a Temp variable for traversal
  listElement * temp = list;
  //Queues with only One element
  if(temp->next == NULL) {
    list = NULL;
    return temp;
  }

  while(temp->next->next != NULL) {
    temp = temp->next;
  }

  listElement * temp2 = temp->next;
  temp->next = NULL;
  return temp2;
}
```

2.1.2 tests.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "tests.h"
#include "linkedList.h"

void runTests(){
  printf("Tests running...\n");
  listElement* l = createEl("Test String (1).", 30);
  //printf("%s\n%p\n", l->data, l->next);
  //Test create and traverse
  traverse(l);
  printf("\n");

  //Test insert after
  listElement* l2 = insertAfter(l, "another string (2)", 30);
  insertAfter(l2, "a final string (3)", 30);
  traverse(l);
  printf("\n");

  // Test delete after
  deleteAfter(l);
  traverse(l);
  printf("\n");

  //Test linkedlist Length
  printf("Length: %d\n\n",length(l));
  printf("Testing Push:\n");
  push(&l,"I was Pushed onto the linked list",50);
  traverse(l);
  printf("\nTesting Pop:\n");
  listElement * popped = pop(&l);
  free(popped);
  traverse(l);
  printf("\nTesting Enqueue:\n");
  enqueue(&l,"Hi",5);
  traverse(l);
  printf("\nTesting Dequeue:\n");
  listElement * last = dequeue(l);
  free(last);
  traverse(l);
  printf("\nTests complete.\n");
}
```

2.2 Output



```
Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1
$ make question2
gcc -Wall -g -o bin/q2 src/q2/assignment.c src/q2/linkedList.c src/q2/tests.c

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1
$ cd bin

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1/bin
$ ./q2.exe
Tests running...
Test String (1).

Test String (1).
another string (2)
a final string (3)

Test String (1).
a final string (3)

Length: 2

Testing Push:
I was Pushed onto the linked list
Test String (1).
a final string (3)

Testing Pop:
Test String (1).
a final string (3)

Testing Enqueue:
Hi
Test String (1).
a final string (3)

Testing Dequeue:
Hi
Test String (1).

Tests complete.

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1/bin
$ |
```

3 Question 3

3.1 Code

3.1.1 assignment.c

```c
#include <stdio.h>
#include "genericLinkedList.h"
#include "tests.h"

int main(int arg, char* argc[]){
  runTests();
  return 0;
}
```

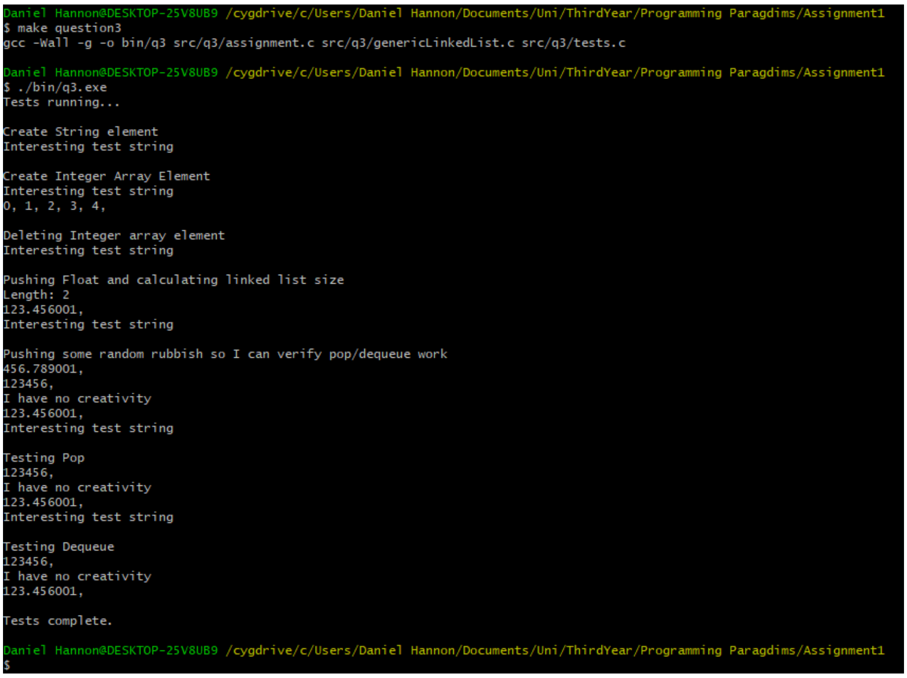
3.1.2 genericLinkedList.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "genericLinkedList.h"

typedef struct genericListElementStruct {
  void * data;
  size_t size;
  fptr printFunc;
  struct genericListElementStruct * next;
} genericListElement;

genericListElement* createGEl(void * data, size_t size, void(printFunc)(void*,size_t)) {
  genericListElement * gel = (genericListElement *) malloc(sizeof(genericListElement));
  //Copy element in and stuff, I use memcpy for obvious reasons
  gel->data = malloc(size);
  memcpy(gel->data,data,size);
  gel->size = size;
  //Prevent junk data here :)
  gel->next = NULL;
  gel->printFunc = printFunc;
  return gel;
}

void genericTraverse(genericListElement * start) {
  genericListElement * curr = start;
  while(curr != NULL) {
    curr->printFunc(curr->data,curr->size);
    curr = curr->next;
  }
}

//inserts a new element after the given el
genericListElement * genericInsertAfter(genericListElement * after, void* data, size_t size, void(printFunc)(void*,size_t)) {
  genericListElement * new = createGEl(data,size,printFunc);
  genericListElement * temp = after->next;
  after->next = new;
  new->next = temp;
  return new;
}

void genericDeleteAfter(genericListElement * after) {
  if(after->next != NULL) {
    genericListElement * temp = after->next;
    after->next = temp->next;
    free(temp->data);
    free(temp);
  }
}

int genericLength(genericListElement * start) {
  int count = 0;
  genericListElement * curr = start;
  while(curr!= NULL) {
    count++;
    curr = curr->next;
  }
  return count;
}

void genericPush(genericListElement ** list, void* data, size_t size, void(printFunc)(void*,size_t)) {
  genericListElement * new = createGEl(data,size,printFunc);
  new->next = (*list);
  *list = new;
}

genericListElement * genericPop(genericListElement ** list) {
  genericListElement * temp = *(list);
  *list = (*list)->next;
  temp->next = NULL;
  return temp;
}

void genericEnqueue(genericListElement ** list, void * data, size_t size, void(printFunc)(void *,size_t)) {
  genericListElement * new = createGEl(data,size,printFunc);
  new->next = *(list);
  *(list) = new;
}

genericListElement* genericDequeue(genericListElement * list) {
  genericListElement * temp = list;
  if(list == NULL) {
    return NULL;
  }else if(list->next == NULL) {
    return list;
  } else {
    while(temp->next->next != NULL) {
      temp = temp->next;
    }
    genericListElement * temp2 = temp->next;
    temp->next = NULL;
    return temp2;
  }
}

void printGenericString(void * data,size_t size) {
  char * stringData = (char *) data;
  int strLength = size/sizeof(char);
  for(int i = 0; i<strLength; i++) {
    //Special String specific end case
    if(*(stringData + i) == '\0') {
      break;
    }
    printf("%c",*(stringData + i));
  }
  printf("\n");
}

void printGenericInt(void * data,size_t size) {
  int* intData = (int *) data;
  int intLen = size/sizeof(int);
  for(int i = 0; i < intLen; i++) {
    printf("%d, ",*(intData + i));
  }
  printf("\n");
}

void printGenericFloat(void * data,size_t size) {
  float* floatData = (float *) data;
  int floatLen = size/sizeof(float);
  for(int i = 0; i < floatLen; i++) {
    printf("%f, ",*(floatData+i));
  }
  printf("\n");
}

void printGenericDouble(void * data,size_t size) {
  double* doubleData = (double *) data;
  int doubleLen = size/sizeof(double);
  for(int i = 0; i < doubleLen; i++) {
    printf("%lf, ",*(doubleData+i));
  }
  printf("\n");
}
```

3.1.3 tests.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "genericLinkedList.h"

void runTests(){
  printf("Tests running...\n");

  printf("\nCreate String element\n");
  genericListElement * l = createGEl("Interesting test string",30,&printGenericString);
  genericTraverse(l);
  printf("\nCreate Integer Array Element\n");
  //Make it a pointer so free does not cause a segfault
  int * numArray = (int *) malloc(5 * sizeof(int));
  for(int i = 0; i < 5; i++) {
    *(numArray+i) = i;
  }
  genericInsertAfter(l,numArray,5*sizeof(int),&printGenericInt);
  free(numArray);
  genericTraverse(l);
  printf("\nDeleting Integer array element\n");
  genericDeleteAfter(l);
  genericTraverse(l);
  printf("\nPushing Float and calculating linked list size\n");
  float testNum = 123.456;
  float * ptestNum =  &testNum;
  genericPush(&l,ptestNum,sizeof(float),&printGenericFloat);
  printf("Length: %d\n",genericLength(l));
  genericTraverse(l);
  printf("\nPushing some random rubbish so I can verify pop/dequeue work\n");
  genericPush(&l,"I have no creativity",30,&printGenericString);
  int test1234 = 123456;
  float testfloat1234 = 456.789;
  genericPush(&l,&test1234,1*sizeof(int),&printGenericInt);
  genericPush(&l,&testfloat1234,sizeof(float),&printGenericFloat);
  genericTraverse(l);

  printf("\nTesting Pop\n");
  genericListElement * temp = genericPop(&l);
  free(temp);
  genericTraverse(l);
  printf("\nTesting Dequeue\n");
  genericListElement * temp2 = genericDequeue(l);
  free(temp2);
  genericTraverse(l);
  printf("\nTests complete.\n");
}
```

3.2 Output



```
Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1
$ make question3
gcc -Wall -g -o bin/q3 src/q3/assignment.c src/q3/genericLinkedList.c src/q3/tests.c

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1
$ ./bin/q3.exe
Tests running...

Create String element
Interesting test string

Create Integer Array Element
Interesting test string
0, 1, 2, 3, 4,

Deleting Integer array element
Interesting test string

Pushing Float and calculating linked list size
Length: 2
123.456001,
Interesting test string

Pushing some random rubbish so I can verify pop/dequeue work
456.789001,
123456,
I have no creativity
123.456001,
Interesting test string

Testing Pop
123456,
I have no creativity
123.456001,
Interesting test string

Testing Dequeue
123456,
I have no creativity
123.456001,

Tests complete.

Daniel Hannon@DESKTOP-25V8UB9 /cygdrive/c/Users/Daniel Hannon/Documents/Uni/ThirdYear/Programming Paragdims/Assignment1
$
```

3.3 Comments

In Order to achieve the generic linkedlist I had to figure out how to pass a function as an arguement and stuff, I wrote the print functions in a way that would ensure that I could print arrays as I felt that would be cool and practical.
fptr is defined as "typedef void(*fptr)(void *,size_t);" in genericLinkedList.h
When it came to copying the data for the pointers I used memcpy as it seemed to be the logical solution

4 Question 4

4.1 Reversing a Singly Linked List

What seems to be the logical way to traverse a singly linked list is to create a stack and push every element to the stack and then at the end you can traverse that as it is the linked list in a reversed order, Albeit it requires you to make a copy of the linked list and thus it will occupy a total of twice the total memory occuppied by the Linked List. It would also require you to recreate the stack every time that the linked list is updated which could become quite computationally expensive as the linked list becomes larger.

4.2 Possible method to make it less expensive in terms of memory and computation

A way to deal with this would be to instead of having a Singly-Linked List we could make it a Doubly-Linked List, that being each element has a pointer to the element before it in the linked list. instead of occupying twice the memory for a reverse traversal, it would occupy 4/8 bytes extra per node depending on whether or not you have a 32 or 64 bit operating system. In order to get the reverse of the linked list we would require to search to the end and then simply use the pointer to the previous node on each node until you return to the start. In order to eliminate that we could make the list circular (Last node has the first one as the next one and vice versa) but that might require extra memory as you would need some way to determine if you reached the start again