McMaster University
SFWR ENG 2MD3 Winter 2020
Assignment 1
Name: Manuel R. Lemos
Student Number: 400177763
Student ID: lemosm1
Due: Sunday January 20, 2019 at 23:55

I. Pointers (12 marks)

Question 1 (2 marks each part)

a) Suppose A is a variable declared in C to be a pointer variable containing pointer to storage type T. How do you dynamically allocate a new block of storage of type T and obtain a pointer to it in C?

```
T* ptr = (T*) malloc(sizeof(T));
```

b) Declare A to be a pointer to integer and assign a value of 20 to its referent. How would you write an expression whose value is twice the value of A's referent? [3 marks]

```
int B = 20;
int * A = &B;

int C = 2*(*A); //C has a value twice that of A's referent
int * DA = &C;
printf("%d", *DA);
```

c) What are aliases?

Aliases are a means to access the same memory location using different names. This can be represented by two or more pointers pointing to the same location such that if the referent of one pointer is appended, it will be appended for all pointers pointing to that referent

d) What is a dangling pointer?

A dangling pointer is a pointer which points to a memory location which has been deleted or freed. A dangling pointer occurs as a result of; deallocation of memory, a variable going out of scope, and a function call.
Once dereferenced, a pointer will yield a garbage value when called (ptr*).

Question 2 (4 marks)
Provide answers to questions 1, 2 of Exercise 2.3 of the book. (Please note a typo in the book- a semicolon is missing in line 2 of both questions!)

1. What does the following procedure print when it is executed?

```
1  void P(void)
2  {
3      A = (IntegerPointer)malloc(sizeof(int));
4      B = (IntegerPointer)malloc(sizeof(int))
5      *A = 19;
6      *B = 5;
7      A = B;
8      *B = 7;
9      printf("%d\n",*A);
10 }
```

Output: 7

Why: A = B sets the memory address reference of A equal to the memory address reference of B

Thus, any further changes to the referent of B impact the referent of A and vice versa

2. What does the following procedure print when it is executed?

```
1  void P(void)
2  {
3      A = (int *)malloc(sizeof(int));
4      B = (int *)malloc(sizeof(int))
5      *A = 19;
6      *B = 5;
7      *A = *B;
8      *B = 7;
9      printf("%d\n",*A);
10 }
```

Output: 5
Why: *A = *B sets the value A points to = to the value B points to. Thus, the further change to the value B points to does not update A*

II. Linked List (23 marks)
Question 3 (4 marks each part)
Provide answers to questions 1 to 5 of Exercise 2.5 of the book.

1. Write a Function InsertNewFirstNode(A, &L);

```
void InsertNewFirstNode (AirportCode A, NodeType **O)
{
  NodeType *N = (NodeType*) malloc(sizeof(NodeType));
  strcpy(N->Airport, A);
  N->Link = *O;
  *O = N;
}
```

2. Write a function DeleteFirst(&L);

```
void DeleteFirst(NodeType **L)
{
   NodeType *N;
   if (*L != '\0')
   {
       N = *L;
       *L = N->Link;
       free (N);
   }
}
```

3. Write a function InsertMBeforeN(&M, &N);

```
void InsertMBeforeN(NodeType *M, NodeType *N)
{
    AirportCode A;
    M->Link = N->Link;
    N->Link = M;
    strcpy(A, N->Airport);
    strcpy(N->Airport, M->Airport);
    strcpy(M->Airport, A);
}
```

4. Write a function Copy(L);

```
NodeType * Copy(NodeType *Q)
{
    if (Q == '\0') return '\0';
    NodeType *N = (NodeType*) malloc(sizeof(NodeType));
    strcpy(N->Airport, Q->Airport);
    N->Link = Copy(Q->Link);
    return N;
}
```

5. Write a function Reverse(&L);

```
void Reverse(NodeType **L)
{
    NodeType *N;
    NodeType *S;
    S = '\0';
    while (*L != '\0')
    {
        N = *L;
        *L = N->Link;
        N->Link = S;
        S = N;
    }
    *L = S;
}
```

Question 4 (3 marks)

Provide answer to question 9 of Exercise 2.5 of the book.

When one reaches the final node of the list, the node L* points to is freed. Following this, one must store a null pointer in the variable L outside the function. For this to be possible, one must use &L as a parameter when the function is called. Thus, *L = NULL inside the function, stores null where &L points.

Evidently, if only one node existed within a list, and *L were used as opposed to **L, there would be no means to set the external L = NULL.

This is a result of the parameter being passed for the value of L during the function call being the address of the first node within the list, which in turn leaves no means to change the value of the external variable L once the pointer to the first node has been deleted and made null.