NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC1007 Data Structures and Algorithms **2021/22 Semester 2**

Solution 1: Linked Lists

School of Computer Science and Engineering *Nanyang Technological University*

Q1 You are given the following structure definitions and variable declarations,

```
struct person{
  char firstName[15];
  char lastName[15];
  struct{
     int age;
     float height;
     float weight;
     char firstName[15];
     }Info,*InfoPtr;
  struct person personP;
  }student1;
typedef struct person person_t;
person_t* studentPtr = &student1;
person_t** studentPtrPtr = &studentPtr;
```

a is there any syntax error?

b Write an expression that can be used to access *age* from *studentPtr*.

c Write an expression that can be used to access *age* from *studentPtrPtr*.

S1 **a** Yes. A structure cannot contain an instance of itself. We only can include its pointer of type struct person.

- Since the Structure definition of Info and *InfoPtr does not provide any **structure tag**, it is a one-time use of definition. You will not be able to declare any variable separately later.
- It is okay to define a new Structure definition in another Structure definition although it is not a good practice.
- It is okay to have two firstName because they belong to two different structure definitions.

b Here we assume that *InfoPtr is a pointer to Info. InfoPtr = &Info.

```
(*studentPtr).InfoPtr->age
studentPtr->InfoPtr->age
(*studentPtr).Info.age
studentPtr->Info.age
```

c It is noted that studentPtrPtr is a pointer to a pointer.

```
(*studentPtrPtr)->InfoPtr->age);
(*(*studentPtrPtr)).InfoPtr->age);
(*(*studentPtrPtr)).Info.age);
```

To have a better understanding of Structure definition, you may run the following example code.

```
#include <stdio.h>
#include <stdlib.h>

struct person{
    char first[15];
    char last[15];
    struct{
        int age;
        float height;
        float weight;
    }Info,*InfoPtr;
}student1;
typedef struct person person_t;

int main()
{
    person_t* studentPtr = &student1;
    person_t** studentPtrPtr = &studentPtr;
    studentPtr->Info.age = 15;
    studentPtr->InfoPtr = &(student1.Info);
    printf("%d\n",(*studentPtrPtr)->InfoPtr->age);
    printf("%d\n",(*(*studentPtrPtr)).InfoPtr->age);
    printf("%d\n",(*(*studentPtrPtr)).Info.age);
    printf("%d\n",student1.Info.age);
    printf("%d\n",student1.InfoPtr->age);
    return 0;
}
```

Q2 Rewrite insertNode(ListNode **ptrHead, int i, int item) given in the lecture by using a recursive approach.

S2 The code of insertNode() given in the lecture nodes

```
    int insertNode(ListNode **ptrHead, int i, int item){
    ListNode  *cur, *newNode;
    // If empty list or inserting first node, update head pointer
    if (*ptrHead == NULL || i == 0){
        newNode = (ListNode *) malloc(sizeof(ListNode));
        newNode->item = item;
        newNode->next = *ptrHead;
        *ptrHead = newNode;
        return 1;
    }
    // Find the nodes before and at the target position
    // Create a new node and reconnect the links
    else if ((cur = findNode(*ptrHead, i-1)) != NULL){
        newNode = (ListNode *) malloc(sizeof(ListNode));
        newNode->item = item;
```

```
16          newNode->next = cur->next;
17          cur->next = newNode;
18          return 1;
19      }
20      return 0;
```

The recursive version of inderNode().

```
1     int insertNode(ListNode **ptrHead, int i, int item){
2     ListNode  *cur, *newNode;
3     if(*ptrHead ==NULL || i==0){
4           newNode = (ListNode *) malloc(sizeof(ListNode));
5           newNode->item = item;
6           newNode->next = *ptrHead;
7           *ptrHead = newNode;
8           return 1;
9      }
10    else
11        return insertNode(&((*ptrHead)->next),--i,item);
12  }
```

Q3 We assign the link of the last node to the first node instead of assigning it to a null value. This turns the linked list into a circular linked list. Let *Aptr* and *Bptr* point to any two nodes in the linked list. What is the outcome of the following functions?

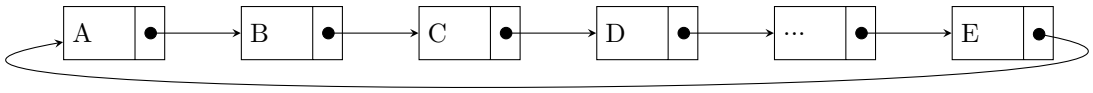

Figure 1.1: A Circular Linked List

```
1   typedef struct node{
2        int item;
3        struct node next;
4        }ListNode;
5
6   void Q3F1(ListNode *Aptr, ListNode *Bptr)
7   {
8     Q3F2(Aptr, Bptr);
9     Q3F2(Bptr, Aptr);
10  }
11  void Q3F2(ListNode *s, ListNode *q)
12  {
13      ListNode *temp = s;
14
15      while(temp->next != q) temp = temp->next;
16      temp->next = s;
17  }
```

S3 The Q3F1() will split the circular linked list into two stand-alone circular linked lists. Each separated linked list contains either node of *Aptr* or node of *Bptr*.