Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Practice Sheet #10

Topic: Linked Lists

Date: 01-03-2017

1. You are given a linked list. Your task is to create two new linked lists, the first of which should contain the 1st, 3rd, 5th,..elements and the second the 2nd, 4th, 6th,.. elements of the input list. The following code segment provides a solution. Fill in the blanks to complete the segment. Evaluation of your answer will depend on overall correctness.

   The function **createLists** assumes that there is a dummy node at the beginning of each list (input as well as output). The input list is headed by the pointer **head**. Odd-numbered elements are to be stored in the list headed by the pointer **oddhead**, and the even-numbered elements are to be stored in the list headed by the pointer **evenhead**. Assume that the input pointer **head** already points to a properly allocated list with a dummy node at the beginning. Assume also that both **oddhead** and **evenhead** are already allocated memory only for the dummy nodes. We number the elements of the input list from 1.

```
/* First define a structure for a node in the list */
typedef struct nodeTag {
int data;
/* Declare the self-referencing pointer */
next;
} node;

void createLists ( node *head , node *oddhead , node *evenhead )
{
node *src, *dest1, *dest2;
int flag = 1;
/* Initialize the source and destination pointers to point to the dummy nodes */
src = head; dest1 = oddhead; dest2 = evenhead;
/* As long as the source list is not fully traversed */
while (_____) {
if (flag == 1) { /* Insert in the odd list */
/* Allocate memory for a new node */

_____
/* Advance the destination pointer by one node */

_____
/* Copy data from source */

_____
} else { /* Insert in the even list */
/* Allocate memory for a new node */

_____
/* Advance the destination pointer by one node */

_____
/* Copy data from source */

_____
}
src = src -> next; /* Look at the next node in the input */
```

```
if (flag == 1) flag = 2; else flag = 1; /* Switch the destination */
}
dest1 -> next = dest2 -> next = NULL; /* Terminate the destination lists */
}
```

2. We are given a linked list and we want to traverse it in a way such that the links are reversed by the time we complete the traversal. Complete the function **traverseReverse** that achieves this objective. You can assume that the input pointer is not **NULL**. Do not use any new variables or malloc/calloc instructions. your program should accomplish the goal by careful manipulation of the pointers.

(a) (! (x>=z))
(b) (x<=z)
(c) ((x<y) && (y<z))
(d) ((x<y) || (y<z))

```
struct node {
char data;
struct node *link;
};

struct node *traverseReverse ( struct node *p1 )
{
struct node *p2, *p3;
/* p2 points the part of the list that has already been reversed and p1 points to
the part that is yet to be reversed. p3 may be used as an auxiliary pointer. */
/* Initialize the pointers */
p2 = p1; p1 = p1->link; p2->link = NULL;
/* As long as the unprocessed part contains more nodes to process */
while (_____ ) {
/* Reverse another link from the unprocessed part. Use pointer assignments only. */
_____

}
p1->link = p2; /* Link p1 to head the reversed list */
return _____; /* Return a pointer to the header of the
reversed list */
}
```

3. The following function **recReverse** recursively reverses a linked list. Which one of the two functions would you prefer and why?

```
struct node *recReverse ( struct node *p )
{
struct node *q, *head;
if (p->link == NULL) return p; /* one node */
/* Recursively reverse the list of at least 2 nodes minus first element */
head = p; p = p->link; q = p = recReverse(p);
/* Append the first node at the end of the reversed list */
while (q->link != NULL) q = q->link;
q->link = head; head->link = NULL; return p;
}
```

4. **(a)** Give a suitable **typedef** to represent a linked list of integers.

(b) Write the C function which takes a list of integers, an integer i and another integer d. It enters d after the i-th element in the list. If i = 0, then it enters at the beginning of the list. Mention clearly whether a dummy header node is used in your function.

(c) In a *circular linked list*, the **next** pointer of the last node points to the starting node of the list. Write a *recursive* C function that prints the elements of a circular linked list of integers in the *reverse* order (that is, from end to beginning).

Use the following function prototype:

void printCList (clist l, const clist h);
Here the second parameter points to the beginning of the list and is kept constant across the calls. Assume that no dummy header node is used in the circular linked list.

5. Consider the type definitions given below. Suppose that a linked list is made up of nodes of type struct node. The last node points to NULL.

```
struct node {
int key;
struct node * next;
};

typedef struct node* link;
```

6. **(a)** Write a function deletesecond() to delete the second node of the list, given "*head*" which points to the first element of the list. Assume that there are at least two nodes on the list.

(b) Write a function `count()` that takes the head of a linked list as input, and returns the number of nodes in the linked list.

(c) Consider the following function `baz()` that takes a pointer to the first node of a linked list.

```
void baz (link head) {
        if (head == NULL)
        return ;
        baz (head->next->next) ;
        printf ("%d ", head->key) ;
        }
```

What will be printed when the function is called with the linked list shown below?
Head -> 1 - > 2 -> 3 -> 4 -> 5 -> 6 -||

(d) Consider the same linked list as shown in the previous figure, with "head" pointing to the first node of the list. Show the changes in the list structure after the following code segment is executed by drawing a diagram. Clearly indicate where, `q` and `p` point to after the execution of the program segment.

```
link head, q, p; q = NULL;
        while(head!=NULL) {
```

```
p=head; head = head->next;
p->next = q;
q = p;
}
```

7. Assume that you are given a linked list with an even number of nodes (excluding the dummy node at the beginning). Your task is to write a function in order to locate the middle of the list, and to subsequently insert two new elements in that middle position. For the sake of ease of programming, you may assume that a dummy node is maintained at the beginning of a linked list. Assume you have a linked list with six elements 43, 71, 82, 9, 37, 64. After inserting two new elements 91 and 5 at the middle, the list changes to 43, 71, 82, 91, 5, 9, 37, 64.

In order to locate the middle of the list, we use two pointers p and q. They are initialized to point to the first node of the list (the dummy node). Subsequently, in a loop, p advances down the list by two cells, whereas q advances down the list by one cell. When p reaches the end of the list, the pointer q is ready for insertion at the middle. Do not allocate two cells together in a single memory allocation call, since if you do so, you cannot free a cell individually.

```
typedef struct _node {
int data;
_________________
} node;
void insertMid (node *head, int n1, int n2 )
{
node *p, *q; /* Do not use any other variable */
p = q = head;
/* While the middle of the linked list is not located */
while (_____________________ != NULL ) {
p =_____________________ ; /* p advances by two cells */
q = _____________________________; /* q advances by one cell */
}
/* Here, q points to the node after which insertion will be made */
/* Allocate memory to the first new node */
p =_____________________________ ;
/* Allocate memory to the second new node */
_____________________________________
/* Set the data fields */
p -> data = _____________________; _____________________= n2;
/* Establish link from the second node created */
_____________________________________________
/* Establish link from the list to include the new nodes */
_____________________________________________________
}
```

8. What is the output of the following program?

```
struct node {
int cval;
struct node *next;
}
main()
{
struct node N1, N2, N3;
N1.cval = 1; N2.cval = 10; N3.cval = 100;
```

```
N1.next = &N2; N2.next = &N3; N3.next = &N1;
printf("%d,%d", N2.next -> cval, N2.next -> next -> cval);
}
```

(a) 1,10
(b) 1,100
(c) 10,100
(d) 100,1

9. In this exercise, we deal with linked lists. In the linked list (say L1), if a node stores the integer n, the next node stores the integer $n - \lfloor\sqrt{n}\rfloor$. This means that we have a list of monotonically decreasing integers. The list terminates after the node containing the value 0.

Define a node in the list in the usual way:

```
typedef struct _node {
int data;
struct _node *next;
} node;
```

(a)Write a function to create a single list as explained above. The list starts with a supplied integer value n, and subsequently uses the above formula for the subsequent nodes.

```
node *genSeq1 ( int n )
{
node *L, *p;
/* Create the fist node to store n */
L =_____
/* Initialize the running pointer p for subsequent insertions */
p =_____ ;
while (_____ ) { /* As long as n is not reduced to zero */
/* Compute the next value of n */
n -= _____;
/* Allocate memory */
_____
/* Store n in the new node, and advance */
_____
}
/* Terminate the list */
return L; /* Return the header */
}
```

(b) Now, we create another list (say L2). This second list starts with another value, and contains nodes storing integer values satisfying the same formula used in the first list. After some iterations, two lists must encounter a common value. From this node onwards, the second list follows the same nodes and links as the first list. Complete the following C function to create the second list. The header to the first list is passed to this function. Also, the starting value n for the second list is passed.

```
node *genSeq2 ( node *L1, int n )
{
node *L2, *p1, *p2;
```

```
/* Skip values in the first list larger than n */
p1 = L1; while ( _____) p1 = p1 -> next;
/* If n is already present in the first list */
if ( _____) return _____;
/* Create the fist node in the second list to store n */
L2 =_____ ;
;
/* Initialize the running pointer p for subsequent insertions */
p2 =_____ ;
while (1) { /* Let us decide to return inside the loop */
n -= _____-; /* Next value of n */
/* p1 skips all values in the first list, larger than the current n */
while _____ p1 = p1 -> next;
if ( _____) { /* n found in first list */
; /* Adjust the second list */
_____
Return_____ ; /* Return header to second list */
}
/* n not found in first list, so create a new node in second list */
}
}
```

(c) Complete the following function that, given the headers L1 and L2 as input, returns a pointer to the first common node in these two lists.

```
node *getIntersection ( node *L1, node *L2 )
{
node *p1, *p2;
p1 =_____ ; p2 = _____; /* Initialize pointers */
while (1) { /* Return inside the loop */
/* If the common node is located, return an appropriate pointer */
if ( _____) return _____;
/* else if p1 points to a larger integer than p2 */
else if (_____ ) _____;
else _____;
}
}
```

10. You are given a linked list of integers which are in ascending order. However, there may be duplicates, which should be removed. Write a 'C' code fragment that removes the duplicates and returns the number of distinct elements in the list.

Ans.

```
typedef struct llNodeTag { // ...via struct for linked list of ints
int val; // value stored in node
struct llNodeTag *nextP; // pointer to next node
} llNodeTyp, *llNodePtr ;
// type names for struct and pointer to struct 3
int removeDuplicates (llNodePtr headP) {
int elemCount=0; // needs to be incremented when a new value is seen
llNodePtr n1P, n2P;
n1P = headP;
if (n1P != NULL) { // list is not empty
// n2P is initialised to point to the successor of n1P
n2P = n1P->nextP ; 1
```

```
elemCount++ ; 1
} else return 0; 1
// keep looping to find duplicates
while ( n2P ! = NULL ) { 1
// test for a duplicate
if (n1P->val == n2P->val ) { // duplicate found 1
// steps to remove the duplicate at n2P
n1P->nextP = n2P->nextP ; 1
free(n2P) ; 1
n2P = n1P->nextP ; 1
} else { // advance in the list
n1P = n2P ; 1
n2P n1P->nextP ; 1
elemCount++ ; 1
}
}
return elemCount ; // final step 1
}
```

11. In this exercise, we deal with linked lists which are kept sorted in the increasing order of the data values. To this end, we define a node in the list in the usual way as:

```
typedef struct _node {
int data;
struct _node *next;
} node;
```

A list is specified by a pointer to the first node in the list. We assume that all the nodes in the list store valid data items, that is, there is no dummy node at the beginning of the list.

Let **H** be a pointer to the first node in a sorted linked list. **H** is **NULL** if the list is empty. We plan to insert a data value **a** in the list such that the list continues to remain sorted after the insertion. The value **a** to be inserted may or may not be already present in the list. Complete the following function to carry out this sorted insertion. The function should return a pointer to the first node in the modified list.

```
node *sortedinsert ( node *H, int a ) {
node *p, *q;
/* Create a new node to store the new value */
p =_____; /* Allocate memory */
_____; /* Store a in the new node */
/* Handle insertion before the first node of the list. This should include
the case of insertion in an empty list. */
if (_____) {
_____;
return _____ ;
}
/* Now we are in a situation where we insert after a node. We first let q point to
the node after which the data item a will be inserted. */
q = _____ ; /* Initialize q for the search */
/* Loop on q to locate the correct insertion position */
```

```
while ( _____) q = q -> next;
/* Finally, insert p after q */
_____;
return _____ ;
}

node *sortedinsert ( node *H, int a ) {
node *p, *q;
/* Create a new node to store the new value */
p = (node *)malloc(sizeof(node)) ; /* Allocate memory */
p -> data = a ; /* Store a in the new node */
/* Handle insertion before the first node of the list. This should include
the case of insertion in an empty list. */
if ( (H == NULL) || (a <= H -> data) ) {
p -> next = H;
return p ;
}
/* Now we are in a situation where we insert after a node. We first let q point to
the node after which the data item a will be inserted. */
q = H ; /* Initialize q for the search */
/* Loop on q to locate the correct insertion position */
while ( (q -> next) && (a > q -> next -> data) ) q = q -> next;
/* Finally, insert p after q */
p -> next = q -> next; q -> next = p;
return H ;
}
```

12. (i) Complete the SortedMerge() function below that takes two non-empty lists, each of which is sorted in increasing order, and merges the them into one list which is in increasing order. SortedMerge() should return the new list. Assume that the elements of the lists are distinct and there are no common elements among the lists. For example if the first linked list a is 5->10->15 and the other linked list b is 2->3->20, then SortedMerge() should return a pointer to the head node of the merged list 2->3->5->10->15->20. The linked list node structure is defined as: struct node {int data; struct node *next;}.

```
struct node *SortedMerge(struct node *a, struct node*b){
struct node *mergedList, *head;
if(a->data < b->data)_____; else_____;
head = mergedList;
while(a!= NULL && b != NULL){
if(a->data < b->data){mergedList->next = a;_____; }
else{ mergedList->next = b; _____;}
mergedList = mergedList->next;
} /* if a list is exhausted before the other */
if(a == NULL) _____;
else _____;
return _____;
}

struct node *SortedMerge(struct node *a, struct node*b){
struct node *mergedList, *head;
if(a->data < b->data) mergedList = a; else mergedList = b;
```

```
head = mergedList;
while(a!= NULL && b != NULL){
if(a->data < b->data){mergedList->next = a; a = a->next; }
else{ mergedList->next = b ; b = b->next;}
mergedList = mergedList->next;
} /* if a list is exhausted before the other */
if(a == NULL) mergedList->next = b;
else mergedList->next = a;
return head;
}
```

13. Write C program statements for the following operations.
(a) Define a node of a circular linked list which contains a complex number.
(b) Assume that the head of the above circular list is pointed by a pointer named head. Write a function which takes the head of the list as argument and returns the sum of complex numbers in the list.
Ans. ??

14. Consider the function f defined below.

```
struct item
{
   int data;
   struct item * next;
};
int f(struct item *p)
{
   return ((p == NULL) || (p->next == NULL) ||
         ((P->data <= p->next->data) &&
         f(p->next)));
}
```

For a given linked list p, the function f returns 1 if and only if
(a) the list is empty or has exactly one element
(b) the elements in the list are sorted in non-decreasing order of data value
(c) the elements in the list are sorted in non-increasing order of data value
(d) not all elements in the list have the same data value.

15. Consider the following C program segment

```
struct CellNode
{
 struct CelINode *leftchild;
 int element;
 struct CelINode *rightChild;
}

int Dosomething(struct CelINode *ptr)
{
   int value = 0;
   if (ptr != NULL)
   {
    if (ptr->leftChild != NULL)
      value = 1 + DoSomething(ptr->leftChild);
    if (ptr->rightChild != NULL)
      value = max(value, 1 + DoSomething(ptr->rightChild));
```

```
    }
    return (value);
}
```

The value returned by the function DoSomething when a pointer to the root of a non-empty tree is passed as argument is

(a) The number of leaf nodes in the tree
(b) The number of nodes in the tree
(c) The number of internal nodes in the tree
(d) The height of the tree

16. Figure out the output of the below code written in C programming language. State relevant assumptions if any during computation.

```
#include<stdio.h>
int main()
        {
            char m= 100;
printf("m= %d",m);
            return 0;
         }
```

(a) Error at Line 5
(b) m=100
(c) Error at Line 4
(d) No output

--*--