DS Assignment III

1. For implementing multiple stacks (n>=3), show the necessary struct declarations with typedef notation. For the struct variables, consider the general version such as boundary [i] and top [i] as boundary and top respectively for the ith stack and that there are no global variables. Using these declarations, give implementation for the following proptotypes:

A. push (int i, int item, STACK * S)
B. pop(int I, STACK *S)
C. main()

```
//
//  MultipleStacksSingleArray.c
//  Multiple stacks in a single array. Display function wasn't needed :P
//
//  Created by Avikant Saini on 9/29/15.
//  Copyright © 2015 avikantz. All rights reserved.
//

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

#define SIZE 10
#define UNDERFLOW_INT -32767

typedef struct MultipleStacks {
    int *arr;
    int *top;
    int *boundary;
} STACK_t;

typedef STACK_t * STACK_p_t;

void push (int item, int i, STACK_p_t stack) {
    if (*(stack->top + i) == *(stack->boundary + i)) {
      printf("\n\tSTACK %d OVERFLOW!\n\n", i + 1);
      return;
    }
    *(stack->arr + ++(*(stack->top + i))) = item;
}

int pop (int i, STACK_p_t stack) {
    if (*(stack->top + i) == -1 || *(stack->top + i) == *(stack->boundary + i
- 1) + 1) {
      printf("\n\tSTACK %d UNDERFLOW!\n\n", i + 1);
      return UNDERFLOW_INT;
    }
    return *(stack->arr + (*(stack->top + i))--);
}
int main (int argc, const char * argv []) {
    int n, i;

    printf("\n\tEnter number of stacks you want: ");
    scanf("%d", &n);
```

```
    STACK_p_t stack = (STACK_p_t)malloc(sizeof(STACK_t));

    stack->arr = (int *)calloc(n * SIZE, sizeof(int));
    stack->top = (int *)calloc(n, sizeof(int));
    stack->boundary  =(int *)calloc(n, sizeof(int));

    for (i = 0; i < n; ++i) {
     *(stack->top + i) = -1;
     *(stack->boundary + i) = (i + 1) * SIZE;
    }

    int stno, ch;

    do {
     printf("\n\tEnter the stack no you want to perform operations on
(1 - %d): ", n);
     scanf("%d", &stno);

     if (stno <= 0 || stno > n) {
          printf("\n\tINVALID CHOICE!\n\n");
          stno = 1;
          continue;
     }

     do {
          printf("\n\t1. Push\n\t2. Pop\n\tCHOICE: ");
          scanf("%d", &ch);

          if (ch == 1) {
               int item;
               printf("\n\tEnter item to push into stack %d: ",
stno);
               scanf("%d", &item);
               push(item, stno - 1, stack);
          }
          else if (ch == 2) {
               int item = pop(stno - 1, stack);
               if (item != UNDERFLOW_INT)
                    printf("\n\tPopped item from stack %d: %d\n",
stno, item);
          }

     } while (ch > 0 && ch <= 2);

    } while (stno > 0 && stno <= n);

    return 0;
}
```

2. The Scratchemup garage contains a single lane that holds up to 10 cars. Cars arrive at the south end of the garage and leave from the north end. If a customer arrives to pick up a car(X) that is not the northernmost, all cars to the north of X's car are moved out, X's car is driven out, and the other cars are restored in the same order that they were in originally. Whenever a car leaves, all cars to the south are moved forward so that at all times all the empty spaces are in the south part of the garage.

Write a program using signally linked list that reads a group of input lines. Each line contains an 'A' for arrival and 'D' for departure and a license plate number (example: 979WJC). Cars are assumed to arrive and depart in the order specified by the input. The program should print a message each time that a car arrives or departs. When car arrives, the message should specify whether or not there is room in the garage for the car. If there is no room for a car, the car then proceeds to the Knockemdead garage which is similar to the Scratchemup. There is room for 8 cars at the Knockemdead garage. If both garages are full, cars wait in the street near the Scratchemup garage for a space...and of course they are queued up in the street. The size of the street queue is also 8.

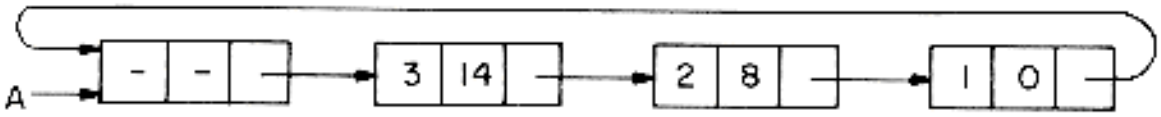
```
//
//  ParkingProblem.c
//  Parking problem using circular linked lists queue
//
//  Created by Avikant Saini on 9/29/15.
//  Copyright © 2015 avikantz. All rights reserved.
//

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

#define SIZE 50

typedef enum { NO, YES } BOOL;

typedef struct Node {
    int data;
    char *lpNo;
    struct Node * next;
} NODE_t;

typedef NODE_t * NODE_p_t;

NODE_p_t createNode () {
    NODE_p_t temp = (NODE_p_t)malloc(sizeof(NODE_t));
    temp->next = temp;
    temp->data = 0;
    temp->lpNo = (char *)malloc(SIZE * sizeof(char));
    return temp;
}
```

```
void addCar (NODE_p_t queue, char *lpno) {
    NODE_p_t temp = createNode();
    NODE_p_t p;

    strcpy(temp->lpNo, lpno);
    temp->next = queue;

    if (queue->next == queue)
      queue->next = temp;

    else {
      p = queue->next;

      while (p->next != queue)
            p = p->next;
      p->next = temp;
    }
    (queue->data)++;
}

void removeCar (NODE_p_t queue, char *lpno) {
    NODE_p_t temp = queue;
    NODE_p_t p;

    while (temp->next != queue) {
      p = temp->next;
      if (strcmp(lpno, p->lpNo) == 0) {
            (queue->data)--;
            temp->next = p->next;
            free(p);
            return;
      }
      temp = temp->next;
    }
}

BOOL containsCar (NODE_p_t queue, char *lpno) {
    NODE_p_t temp = queue->next;
    while (temp != queue) {
      if (strcmp(lpno, temp->lpNo) == 0)
            return YES;
      temp = temp->next;
    }
    return NO;
}
```

```c
int main (int argc, const char * argv []) {

    NODE_p_t scratchEmUp = createNode();
    NODE_p_t knockEmDown = createNode();
    NODE_p_t streetThugs = createNode();

    char arrDep;
    char *lpNo = (char *)malloc(SIZE * sizeof(char));

    do {
      printf("\n\tGIVE INPUT: ");
      scanf(" %c", &arrDep);
      scanf(" %s", lpNo);

      if (arrDep == 'A') {

            if (scratchEmUp->data < 10) {
                  printf("\n\tAdding car '%s' to ScratchEmUp garage.\n", lpNo);
                  addCar(scratchEmUp, lpNo);
            }

            else if (knockEmDown->data < 8) {
                  printf("\n\tAdding car '%s' to KnockEmDown garage.\n", lpNo);
                  addCar(knockEmDown, lpNo);
            }

            else if (streetThugs->data < 8) {
                  printf("\n\tAdding car '%s' to the Street.\n", lpNo);
                  addCar(streetThugs, lpNo);
            }

            else {
                  printf("\n\tParking and Street Full! Only handicap parking space
available. Are you actually thinking of parking there?\n\n");
                  continue;
            }
      }

      else if (arrDep == 'D') {

            if (containsCar(scratchEmUp, lpNo)) {
                  printf("\n\tRemoving car '%s' from ScratchEmUp garage.\n", lpNo);
                  removeCar(scratchEmUp, lpNo);
            }

            else if (containsCar(knockEmDown, lpNo)) {
                  printf("\n\tRemoving car '%s' from KnockEmDown garage.\n", lpNo);
                  removeCar(knockEmDown, lpNo);
            }
```

```
            else if (containsCar(streetThugs, lpNo)) {
                  printf("\n\tRemoving car '%s' from the street.\n", lpNo);
                  removeCar(streetThugs, lpNo);
            }

            else {
                  printf("\n\tCar '%s' not found in both garages or streets. It's
probably stolen by aliens. There's nothing you can do now, except weep silently or
loudly.\n", lpNo);
                  continue;
            }

      }

    } while (arrDep == 'A' || arrDep == 'D');

    return 0;
}
```

3. a) Write a function to represent a polynomial of order 'n' using circular linked list representation as show below. Do not declare any global variables.



Circular List Representation of $A = 3x^{14} + 2x^{8} + 1$

b) Let 'a' be a pointer to a polynomial. Write a function, *peval* to evaluate the polynomial *a* at point *x*, where *x* is some floating point number.

```
//
//  PolynomialLinkedList.c
//  Polynomial evaulation using singly circular linked list
//
//  Created by Avikant Saini on 9/29/15.
//  Copyright © 2015 avikantz. All rights reserved.

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

typedef enum { NO, YES } BOOL;
```

```c
typedef struct Node {
    int coeff;
    int pow;
    struct Node * next;
} NODE_t;

typedef NODE_t * NODE_p_t;

NODE_p_t createNode () {
    NODE_p_t temp = (NODE_p_t)malloc(sizeof(NODE_t));
    temp->next = temp;
    temp->coeff = 0;
    temp->pow = 0;
    return temp;
}

void insert (NODE_p_t list, int coeff, int pow) {
    NODE_p_t temp = createNode();
    NODE_p_t p;

    temp->coeff = coeff;
    temp->pow = pow;
    temp->next = list;

    if (list->next == list)
      list->next = temp;

    else {
      p = list->next;

      while (p->next != list)
            p = p->next;
      p->next = temp;

    }
}

double peval (NODE_p_t polyn, double x) {
    double result = 0.0;
    NODE_p_t temp = polyn->next;
    while (temp != polyn) {
      result += temp->coeff * pow(x, temp->pow);
      temp = temp->next;
    }
    return result;
}
```

```c
int main (int argc, const char * argv []) {

    NODE_p_t a = createNode();

    int i, n, coeff, pow;
    printf("\n\tEnter number of terms in the polynomial: ");
    scanf("%d", &n);

    for (i = 0; i < n; ++i) {
      printf("\n\tEnter coefficient and power for term %d: ", (i+1));
      scanf("%d %d", &coeff, &pow);
      insert(a, coeff, pow);
    }

    double x, result;
    printf("\n\tEnter value of 'x': ");
    scanf("%lf", &x);

    result = peval(a, x);

    printf("\n\tResult = %.3lf\n\n", result);

    return 0;
}
```

4. a) Write a function Union_Create() that take headers of two lists(sets) as parameters and returns the address of the merged list which is the union of these two lists.

b) Assuming the lists to be sorted, write a function that takes two sorted lists as paramters and perform the intersection of elements of two lists and return the address of the intersect_list.

```c
typedef enum { NO, YES } BOOL;

typedef struct Node {
    char data;
    struct Node * next;
} NODE_t;

typedef NODE_t * NODE_p_t;

NODE_p_t createNode () {
    NODE_p_t temp = (NODE_p_t)malloc(sizeof(NODE_t));
    temp->next = temp;
    return temp;
}
```

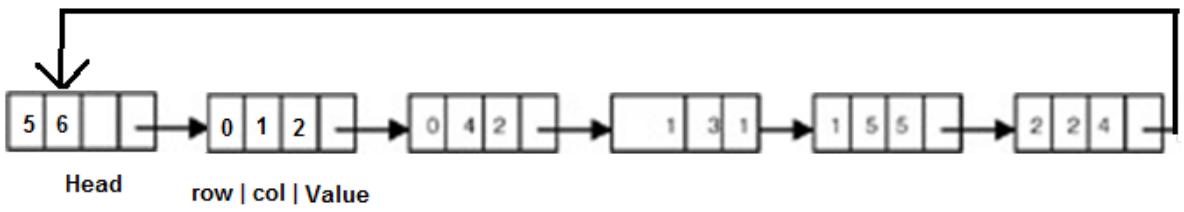
```
void insert (NODE_p_t list, char item) {
    NODE_p_t temp = createNode();
    NODE_p_t p;

    temp->data = item;
    temp->next = list;

    if (list->next == list)
      list->next = temp;

    else {
      p = list->next;
      while (p->next != list)
            p = p->next;
      p->next = temp;

    }
}

BOOL listContainsItem (NODE_p_t list, char item) {
    NODE_p_t temp = list->next;
    while (temp != list) {
      if (temp->data == item)
            return YES;
      temp = temp->next;
    }
    return NO;
}

// For the people who made this question paper: Your function name is non standard.
NODE_p_t Union_Create (NODE_p_t list1, NODE_p_t list2) {
    NODE_p_t unionList = createNode();

    NODE_p_t temp1 = list1->next;
    NODE_p_t temp2 = list2->next;

    while (temp1 != list1) {
      char item = temp1->data;
      if (!listContainsItem(unionList, item))
            insert(unionList, item);
      temp1 = temp1->next;
    }
    while (temp2 != list2) {
      char item = temp2->data;
      if (!listContainsItem(unionList, item))
            insert(unionList, item);
      temp2 = temp2->next;
    }
    return unionList;
}
```

```
NODE_p_t getIntersection (NODE_p_t list1, NODE_p_t list2) {
    NODE_p_t intersect_list = createNode();
    // Again varible name, non standard.

    NODE_p_t temp = list1->next;

    while (temp != NULL) {
      char item = temp->data;
      if (listContainsItem(list2, item) && !
listContainsItem(intersect_list, item))
           insert(intersect_list, item);
      temp = temp->next;
    }

    return intersect_list;
}
```

5. Assume that a sparse matrix is represented using a singly list representation as shown in the figure below. Head node contains the number of rows, columns of original matrix and other nodes stores row number column number and value of non-zero elements in the sparse matrix.

0	2	0	0	2	0
0	0	0	1	0	5
0	0	4	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Sparse Matrix



Representation using a circular list

Write a function to add two sparse matrices which are represented using circular list. The result of addition should be stored in a dynamically allocated 2D array (only if the dimensions match otherwise display an error message).

```c
//
//  SparseMatrix.c
//  Adding two sparse matrices using circular linked lists queue
//
//  Created by Avikant Saini on 9/30/15.
//  Copyright © 2015 avikantz. All rights reserved.
//

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

typedef enum { NO, YES } BOOL;

typedef struct Node {
    int row;
    int col;
    int value;
    struct Node * next;
} NODE_t;

typedef NODE_t * NODE_p_t;

NODE_p_t createNode () {
    NODE_p_t temp = (NODE_p_t)malloc(sizeof(NODE_t));
    temp->next = temp;
    temp->row = 0;
    temp->col = 0;
    temp->value = 0;
    return temp;
}

void insert (NODE_p_t list, int r, int c, int v) {
    NODE_p_t temp = createNode();
    NODE_p_t p;

    temp->next = list;
    temp->row = r;
    temp->col = c;
    temp->value = v;

    if (list->next == list)
      list->next = temp;

    else {
      p = list->next;
      while (p->next != list)
            p = p->next;
      p->next = temp;

    }
}
```

```c
// Create a dynamically allocated 2D array sparse matrix from linked lists
int ** sparseMatrixFromLinkedLists (NODE_p_t list1, NODE_p_t list2) {

    int ** mat = (int **)calloc(list1->row, sizeof(int *));
    int i, j;

    // Set every element to zero
    for (i = 0; i < list1->row; ++i) {
     *(mat + i) = (int *)calloc(list1->col, sizeof(int));
     for (j = 0; j < list1->col; ++j)
          *(*(mat + i) + j) = 0;
    }

    NODE_p_t temp1 = list1->next;
    NODE_p_t temp2 = list2->next;

    // Move throught the linked lists and add the value to that index of the matrix
    while (temp1 != list1) {
     *(*(mat + temp1->row) + temp1->col) += temp1->value;
     temp1 = temp1->next;
    }
    while (temp2 != list2) {
     *(*(mat + temp2->row) + temp2->col) += temp2->value;
     temp2 = temp2->next;
    }

    return mat;
}



int main (int argc, const char * argv []) {

    NODE_p_t mat1 = createNode();
    NODE_p_t mat2 = createNode();

    int i, j, r, c, v;

    printf("\n\tEnter dimensions of Matrix 1: ");
    scanf("%d %d", &mat1->row, &mat1->col);      // Header

    printf("\n\tEnter Matrix 1's non zero elements (row, column,
value). <? ? 0> i.e. v = 0 for break: ");
    do {
     scanf("%d %d %d", &r, &c, &v);
     if (v != 0)
          insert(mat1, r, c, v);
    } while (v != 0);
```

```
    printf("\n\tEnter dimensions of Matrix 2: ");
    scanf("%d %d", &mat2->row, &mat2->col);      // Header

    printf("\n\tEnter Matrix 2's non zero elements (row, column,
value). <? ? 0> i.e. v = 0 for break: ");
    do {
      scanf("%d %d %d", &r, &c, &v);
      if (v != 0)
            insert(mat2, r, c, v);
    } while (v != 0);


    if (!((mat1->row == mat2->row) && (mat1->col == mat2->col))) {
      printf("\n\tError: Unequal dimensions.\n\n");
      return 9;
    }

    int ** sum = sparseMatrixFromLinkedLists(mat1, mat2);

    printf("\n\tMatrix Sum:\n");
    for (i = 0; i < mat1->row; ++i) {
      for (j = 0; j < mat1->col; ++j)
            printf("\t%d", *(*(sum + i) + j));
      printf("\n");
    }

    printf("\n\n");
}
```