Design a stack with operations on middle element

```
/* Program to implement a stack that supports findMiddle() and deleteMiddle
  in O(1) time */
#include <stdio.h>
#include <stdlib.h>

/* A Doubly Linked List Node */
struct DLLNode
{
   struct DLLNode *prev;
   int data;
   struct DLLNode *next;
};

/* Representation of the stack data structure that supports findMiddle()
  in O(1) time.  The Stack is implemented using Doubly Linked List. It
  maintains pointer to head node, pointer to middle node and count of
  nodes */
struct myStack
{
   struct DLLNode *head;
   struct DLLNode *mid;
   int count;
};

/* Function to create the stack data structure */
struct myStack *createMyStack()
{
   struct myStack *ms =
             (struct myStack*) malloc(sizeof(struct myStack));
   ms->count = 0;
   return ms;
};

/* Function to push an element to the stack */
```

```
void push(struct myStack *ms, int new_data)
{
   /* allocate DLLNode and put in data */
   struct DLLNode* new_DLLNode =
            (struct DLLNode*) malloc(sizeof(struct DLLNode));
   new_DLLNode->data  = new_data;

   /* Since we are adding at the begining,
     prev is always NULL */
   new_DLLNode->prev = NULL;

   /* link the old list off the new DLLNode */
   new_DLLNode->next = ms->head;

   /* Increment count of items in stack */
   ms->count += 1;

   /* Change mid pointer in two cases
      1) Linked List is empty
      2) Number of nodes in linked list is odd */
   if (ms->count == 1)
   {
       ms->mid = new_DLLNode;
   }
   else
   {
      ms->head->prev = new_DLLNode;

      if (ms->count & 1) // Update mid if ms->count is odd
         ms->mid = ms->mid->prev;
   }

   /* move head to point to the new DLLNode */
   ms->head  = new_DLLNode;
}

/* Function to pop an element from stack */
int pop(struct myStack *ms)
{
   /* Stack underflow */
```

```
    if (ms->count  ==  0)
    {
        printf("Stack is empty\n");
        return -1;
    }

    struct DLLNode *head = ms->head;
    int item = head->data;
    ms->head = head->next;

    // If linked list doesn't become empty, update prev
    // of new head as NULL
    if (ms->head != NULL)
        ms->head->prev = NULL;

    ms->count -= 1;

    // update the mid pointer when we have even number of
    // elements in the stack, i,e move down the mid pointer.
    if (!((ms->count) & 1 ))
        ms->mid = ms->mid->next;

    free(head);

    return item;
}

// Function for finding middle of the stack
int findMiddle(struct myStack *ms)
{
    if (ms->count  ==  0)
    {
        printf("Stack is empty now\n");
        return -1;
    }

    return ms->mid->data;
}

// Driver program to test functions of myStack
```

```
int main()
{
    /* Let us create a stack using push() operation*/
    struct myStack *ms = createMyStack();
    push(ms, 11);
    push(ms, 22);
    push(ms, 33);
    push(ms, 44);
    push(ms, 55);
    push(ms, 66);
    push(ms, 77);

    printf("Item popped is %d\n", pop(ms));
    printf("Item popped is %d\n", pop(ms));
    printf("Middle Element is %d\n", findMiddle(ms));
    return 0;
}
```

Output:

```
Item popped is 77
Item popped is 66
Middle Element is 33
```