USN	1	S	I							

3CCI02

Siddaganga Institute of Technology, Tumkur – 572 103

Department of Computer Science and Engineering

Semester : III **Odd 2016-17** Subject : Data structures-3CCI01

Assignment – 4

1. Programming Assignment : Nearest Larger Number in Stack

Nearest larger number (NLN) of element of stack is defined as the closest greater number that are pushed before the element, if there are no such element then NLN will be -1. Given a stack of unique positive integer, return another stack with NLN of each element in the same order of original stack. Consider example of stack, where 4 is top-most and 3 bottom-most. 3 7 5 6 8 4 Then NLN of each element is given by -1 -1 7 7 -1 8 Explanation:

*** Number greater than 4 which is closest and below stack is 8, Similarly the number greater 7 is greater than 6 and closest to 6 Since there is no element greater than 8 below the stack NLN is -1***

and so on ..

You have complete returnStackWithNearestGreater(Stack inputStack) that return the NLN Stack. Input N a1 a2 a3 a4 .. aN where N is number of elements in the stack, ai are elements of the stack where aN is top-most element of the stack Ouput: b1 b2 b3 b4 .. bN where b1 is NLN for a1, b2 is NLN for a2 and so on…. Sample Test Cases Input Output

Test Case 1

10 1 2 3 4 5 6 7 8 9 0

-1 -1 -1 -1 -1 -1 -1 -1 -1 9

Test Case 2

5 17 43 32 12 49

-1 -1 43 32 -1

Test Case 3

5 9 8 10 3 5

-1 9 -1 10 10

Test Case 4

8 6 7 5 9 8 3 1 2

-1 -1 7 -1 9 8 3 3

Test Case 5

9 81 72 73 68 63 21 43 91 88

-1 81 81 73 68 63 63 -1 91

Solution:

```
/*

 Date : 14/11/2016

Returning Nearest greater element stack */

#include <stdio.h>

#include <stdlib.h>

typedef struct{

int top;

int arr[20];

}Stack;

Stack returnStackWithNearestGreater(Stack inputStack){

Stack NLNStack;

int val,top1;

int max=inputStack.top;

NLNStack.top=max+1;

while(inputStack.top!=-1){

   top1=inputStack.top;

   val=inputStack.arr[top1];

   top1--;

   while(top1!=-1){

      if(inputStack.arr[top1]>val){

         NLNStack.arr[--NLNStack.top]=inputStack.arr[top1];

         break;

      }

      top1--;

   }

   if(top1==-1)

      NLNStack.arr[--NLNStack.top]=-1;
```

```
    inputStack.top--;

}

NLNStack.top=max;

return NLNStack;

}

int main()

{

    int n,i;

    Stack inputStack;

    inputStack.top=-1;

    printf("Enter the no of elements u want in the stack\n");

    scanf("%d",&n);

    printf("Enter elements to the stack\n\n");

    for(i=0;i<n;i++)

        scanf("%d",&(inputStack.arr[++inputStack.top]));

    for(i=inputStack.top;i>=0;i--)

        printf("%d\n",inputStack.arr[i]);

  Stack NLNStack;


    NLNStack=returnStackWithNearestGreater(inputStack);

    for(i=0;i<=NLNStack.top;i++)

        printf("%d ",NLNStack.arr[i]);

    return 0;}
```

2. Write 2 C functions as specified below: You may define additional auxiiary functions as needed. In all cases you may assume that the value passed to the function is of the expected type, so your function does not have to check for malformed inputs.

a) Define a C function "descending (l)" that returns True if each element in its input list is at most as big as the one before it. b) Define a C function "alternating (l)" that returns True if the values in the input list alternately go up and down (in a strict manner).

Solution:

a)

```
/*
 Date : 14/11/2016
 To check for existence of a DESCENDING LIST and return a boolean*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct node{
   int data;
   struct node* next;
};
typedef struct node* Nodeptr;

Nodeptr getNode(void){
   Nodeptr new_node=(Nodeptr)malloc(sizeof(struct node));
   return new_node;
}
void insert(Nodeptr* first,int val){
   Nodeptr last=*first;
   Nodeptr temp=getNode();
   temp->data=val;
   temp->next=NULL;
   if(*first==NULL){
      *first=temp;
      return;
   }
   while(last->next!=NULL)
      last=last->next;
```

```
    last->next=temp;


}

bool descending(Nodeptr list){

if(list==NULL)

    return false;

Nodeptr prev=list;

list=list->next;

while(list!=NULL){

    if(prev->data<list->data)

        return false;

    prev=list;

    list=list->next;

}


return true;


}

int main(){

Nodeptr l=NULL;

int val;

while(scanf("%d",&val)!=EOF){

    insert(&l,val);

}

int n=descending(l)?printf("TRUE\n"):printf("FALSE\n");

}
```

b)

```
/*
```

Date : 14/11/2016

To check for an ALTERNATING SEQUENCE (up then down) IN A LIST and return a boolean*/

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


struct node{

   int data;

   struct node* next;

};

typedef struct node* Nodeptr;


Nodeptr getNode(void){

   Nodeptr new_node=(Nodeptr)malloc(sizeof(struct node));

   return new_node;

}

void insert(Nodeptr* first,int val){

   Nodeptr last=*first;

   Nodeptr temp=getNode();

   temp->data=val;

   temp->next=NULL;

   if(*first==NULL){

      *first=temp;

      return;

   }

   while(last->next!=NULL)

      last=last->next;

   last->next=temp;
}
bool alternating(Nodeptr list){
if(list==NULL)
   return false;
Nodeptr prev=list;
list=list->next;
while(list->next!=NULL){
   if(prev->data==list->data)
      return false;
   if(prev->data>list->data&&list->data>list->next->data)
      return false;
   if(prev->data<list->data&&list->data<list->next->data)
      return false;
   prev=list;
   list=list->next;
}

return true;


}
int main(){
Nodeptr l=NULL;
int val;
while(scanf("%d",&val)!=EOF){
   insert(&l,val);
}
int n=alternating(l)?printf("TRUE\n"):printf("FALSE\n");
}
```

4. Dividing Sequences

(IARCS OPC Archive, K Narayan Kumar, CMI)

This problem is about sequences of positive integers a1,a2,…,aN. A subsequence of a sequence is anything obtained by dropping some of the elements. For example, 3,7,11,3 is a subsequence of 6,3,11,5,7,4,3,11,5,3 , but 3,3,7 is not a subsequence of 6,3,11,5,7,4,3,11,5,3 .

A fully dividing sequence is a sequence a1,a2,…,aN where ai divides aj whenever i < j. For example, 3,15,60,720 is a fully dividing sequence.

Given a sequence of integers your aim is to find the length of the longest fully dividing subsequence of this sequence.

Consider the sequence 2,3,7,8,14,39,145,76,320

It has a fully dividing sequence of length 3, namely 2,8,320, but none of length 4 or greater.

Consider the sequence 2,11,16,12,36,60,71,17,29,144,288,129,432,993 .

It has two fully dividing subsequences of length 5,

***� 2,11,16,12,36,60,71,17,29,144,288,129,432,993 and �
2,11,16,12,36,60,71,17,29,144,288,129,432,993***

and none of length 6 or greater.

Solution hint

Let the input be a1, a2, …, aN. Let us define Best(i) to be the length of longest dividing sequence in a1,a2,…ai that includes ai.

Write an expression for Best(i) in terms of Best(j) with j<i, with base case Best(1) = 1. Solve this recurrence. Test Data :You may assume that N ≤ 2500.

Solution:

```
#include <stdio.h>



int Best(int*a,int i){

  int j,k,small,temp,count=0,l=0;

  /*sorting*/

  for(k=0;k<i;k++){

        small=k;

     for(j=k+1;j<i;j++)

        if(a[j]<a[small])
```

```
                small=j;
        temp=a[small];
        a[small]=a[k];
        a[k]=temp;
}//end of sorting


/*finding largest division sequence*/
for(k=0;k<i;k++){
        temp=k;
        for(j=k;j<i;j++)
            if(a[j]%a[k]==0){
                k=j;
                count++;
            }
        if(count>l)
         l=count;
        k=temp;
        count=0;
}
return l;


}
int main(){
int n,i;
scanf("%d",&n);
if(n>2500)
    return 1;
int a[n];
for(i=0;i<n;i++)
```

```
   scanf("%d",&a[i]);

printf("%d",Best(a,n));

}
```

5. A list is a palindrome if it reads the same forwards and backwards. For instance [], [7], [8,11,8] and [19,3,44,44,3,19] are palindromes, while [3,18,4] and [23,14,3,14,3,23] are not. Write C function that take as input aa existing list check whether it is a palindrome or not.

Solution:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct node{

int data;

struct node*rlink;

struct node*llink;

};

typedef struct node*Nodeptr;

Nodeptr getNode(void){

Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));

newNode->llink=newNode->rlink=NULL;

return newNode;

}


void insertRear(Nodeptr *first,int val){

Nodeptr newNode=getNode();

newNode->data=val;

if(*first==NULL){

  *first=newNode;

  return;
}

Nodeptr temp=*first;

while(temp->rlink!=NULL)

   temp=temp->rlink;

newNode->llink=temp;

temp->rlink=newNode;

}

bool checkPalindrome(Nodeptr list,Nodeptr last){ //checks whether palindrome or not and returns
true or false

   if(list==NULL)

      return true;

   while(list!=NULL&&last!=NULL){

      if(list->data!=last->data)

         return false;

         list=list->rlink;

         last=last->llink;

   }

   return true;

}

int main(){

   Nodeptr list=NULL;

   int val;

   while(scanf("%d",&val)!=EOF){

      insertRear(&list,val);

   }

   Nodeptr last=list;

   while(last->rlink!=NULL){

      last=last->rlink;

   }

   int n=checkPalindrome(list,last)?printf("True\n"):printf("False\n");
```

```
    return 0;

}
```

6. Write a C function sublist(l1,l2) that takes two sorted lists as arguments and returns True if the the first list is a sublist of the second list, and returns False otherwise. A sublist of a list is a segment consisting of contiguous values, without a gap. Thus, [2,3,4] is a sublist of [2,2,3,4,5], but [2,2,4] and [2,4,5] are not. The input values can be integers or characters.

Solution:

```
#include<stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>




struct node{

    int data;

    struct node* next;

};

typedef struct node* Nodeptr;


Nodeptr getNode(void){

    Nodeptr new_node=(Nodeptr)malloc(sizeof(struct node));

    return new_node;

}


void insertOrdered(Nodeptr *list,int val){

Nodeptr newNode=getNode();

Nodeptr temp=*list;

Nodeptr prev =NULL;

newNode->data=val;
```

```
if(*list==NULL){
    newNode->next=NULL;
    *list=newNode;
    return;
}
if(newNode->data<=temp->data){
        newNode->next=temp;
        *list=newNode;
        return;
}
while(temp!=NULL){
       if(newNode->data>temp->data){
    prev=temp;
    temp=temp->next;
       }
       else
        break;
}
prev->next=newNode;
newNode->next=temp;
return;
}
bool sublist(Nodeptr l1,Nodeptr l2){ //Returns true if sublist found
if(l1==NULL||l2==NULL)
    return false;
while(l1!=NULL&&l2!=NULL){
    if(l1->data==l2->data)
    {
        if(l1->next!=NULL&&l2->next!=NULL&&l1->next->data==l2->next->data)
        {
        while(l1!=NULL){
            if(l1->data!=l2->data)
                return false;
                l1=l1->next;
                l2=l2->next;
        }
        }
        return true;

    }

    l1=l1->next;
    l2=l2->next;
}
}
int main(){
    Nodeptr l1=NULL,l2=NULL;
    int val;
    while(scanf("%d",&val)!=EOF)
        insertOrdered(&l1,val);
    printf("\n\n");
    while(scanf("%d",&val)!=EOF)
        insertOrdered(&l2,val);
    int n=sublist(l1,l2)?printf("True\n"):printf("False\n");
    return 0;}
```