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-3

Solutions to Programs

1) Write a function maxbad () to return the maximum value in a list. Assume there is an error in this function. Provide an input list for which maxbad produces an incorrect output.

```
/****maxbad()function -To return the maximum value of the list

   Error in function - returns 0 for a list of negative numbers

   *****/

#include <stdio.h>

#include<stdbool.h>

struct node{

int data;

struct node*next;

};

typedef struct node* Nodeptr;

void insertRear(Nodeptr* first,int val){

   Nodeptr last=*first;

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

   temp->data=val;

   temp->next=NULL;

   if(*first==NULL){

      *first=temp;

      return;

   }

   while(last->next!=NULL)

      last=last->next;

   last->next=temp;
}

int maxbad(Nodeptr list){

if(list==NULL){

printf("List is Empty\n");

return NULL;

}

int max=list->data;

Nodeptr temp=list;

while(temp!=NULL)

{

   if(max<temp->data)

       max=temp->data;

   temp=temp->next;

}

if(max<0)

   return 0;

return max;

}

int main(){

int i;

for(i=1;i<=5;i++){

printf("\nTest Case %d\n\n",i);

Nodeptr list=NULL;

int val;

printf("Enter the values to the list and Press ctrl+z then Enter to exit :\n ");

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

   insertRear(&list,val);

printf("\n%d\n\n",maxbad(list));

}return 0;}
```

2. Write a C function disjointlist(l1,l2) that takes two lists as arguments and returns True if the two lists are disjoint, otherwise returns False. Two lists are said to be disjoint if there is no element that

common to both the lists. For instance, [2,2,3,4,5] and [6,8,8,1] are disjoint, while [1,2,3,4] and [2,2] are not.

Sample Test Cases

Input Output Test Case 1 disjointlist(L1[2,2,3,4,5],L2[6,8,8,1]) True

Test Case 2 disjointlist([1,2,3,4],[2,2]) False

```
/*

******Testing for Disjoint Lists*******/

#include <stdio.h>

#include<stdbool.h>

struct node{

int data;

struct node*next;

};

typedef struct node* Nodeptr;

void insertFront(Nodeptr*list,int val){

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

newNode->data=val;

if(*list==NULL){

*list=newNode;

newNode->next=NULL;

return;

}

newNode->next=*list;

*list=newNode;

}

bool disjointlist(Nodeptr l1,Nodeptr l2){

if(l1==NULL&&l2==NULL){

   printf("lists are Empty\n");

   return true;

}
if(l1==NULL||l2==NULL)
   return true;
Nodeptr temp1=l1,temp2=l2;
while(temp1!=NULL){
   while(temp2!=NULL){
      if(temp1->data==temp2->data)
         return false;
      temp2=temp2->next;
   }
   temp1=temp1->next;
   temp2=l2;
}
return true;
}
int main(){
int i;
for(i=1;i<=10;i++){
printf("\nTest Case %d\n\n",i);
Nodeptr l1=NULL,l2=NULL;
int val;
printf("Enter the values to list 1 and Press ctrl + z then Enter to exit\n");
while(scanf("%d",&val)!=EOF)
   insertFront(&l1,val);
printf("Enter the values to list 2 and Press ctrl + z then Enter to exit\n");
while(scanf("%d",&val)!=EOF)
   insertFront(&l2,val);
   printf("%s\n",disjointlist(l1,l2)?"\nTRUE\n":"\nFALSE\n");
}  return 0;}
```

3. Recall that the positions in a list of length n are 0,1,…,n-1. We want to write a function oddpositions(l) that returns the elements at the odd positions in l. In other words, the function should return the list [l[1],l[3],...]. For instance oddpositions([]) == [], oddpositions([7]) == [],

oddpositions([8,11,8]) == [11] and oddpositions([19,3,44,44,3,19]) == [3,44,19]. Write a recursive functions to count even sum and odd sum in the linked list.

```
/* Date: 29/10/16

******Dealing with oddPositions of the List

and   counting the sum of even and odd positions*******/

#include <stdio.h>

#include<stdbool.h>

struct node{

int data;

struct node*next;

};

typedef struct node* Nodeptr;

void insertRear(Nodeptr* first,int val){

   Nodeptr last=*first;

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

   temp->data=val;

   temp->next=NULL;

   if(*first==NULL){

      *first=temp;

      return;

   }

   while(last->next!=NULL)

      last=last->next;

   last->next=temp;

}


Nodeptr oddpositions(Nodeptr l1){

if(l1==NULL)return NULL;

Nodeptr l2=NULL;

Nodeptr temp=l1->next;

while(temp!=NULL){
  insertRear(&l2,temp->data);
  if(temp->next==NULL)
   break;
  temp=temp->next->next;
}
return l2;
}
void display(Nodeptr first){
   Nodeptr temp=first;
   printf("\n");
   while(temp!=NULL){
      printf("%d ",temp->data);
      temp=temp->next;
   }
   printf("\n");
}
int evenSum(Nodeptr list){ //recursive function for returning sum of even positions
   if(list==NULL)
      return 0;
   if(list->next!=NULL)
      return list->data+evenSum(list->next->next);
   else
      return list->data;
}
int oddSum(Nodeptr list){ //recursive function for returning sum of odd positions
   if(list==NULL||list->next==NULL)
      return 0;
   list=list->next;
   if(list->next!=NULL)
      return list->data+oddSum(list->next);
   else
```

```
        return list->data;

}

int main(){

int i;

for(i=1;i<=10;i++){

printf("\nTest Case %d\n\n",i);

Nodeptr list=NULL;

int val;

printf("Enter the values to the list and Press ctrl+z then Enter to exit :\n");

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

    insertRear(&list,val);

display(oddpositions(list)); //displaying the list returned by oddpositions

printf("SUM OF EVEN POSITIONS = %d\n",evenSum(list));

printf("SUM OF ODD POSITIONS = %d\n",oddSum(list));

}


return 0;

}
```

4. A queue of integers is represented using an array queue[100],where queue[0] is used to indicate the front of the queue, queue[1] is used to indicate its rear, queue[2] through queue[99] are used to store the queue elements. Show how to initialize such an array to represent the empty queue and write C functions to insert, delete elements and to check queue empty and queue full conditions.

```
/* Date: 29/10/16

******Queue Implementation via Arrays*******/


#include <stdio.h>

#include <stdlib.h>

void Insert(int*queue,int val){
    if(isEmpty(queue))
        queue[0]=2;
    queue[++queue[1]]=val;
}
int Delete(int*queue){
    int val=queue[queue[0]];
    if(queue[0]==queue[1]){
        queue[0]=1;
        queue[1]=1;
        return val;
    }
    queue[0]++;
    return val;
}
int isEmpty(int*queue){
    if(queue[0]==1)
        return 1;
    return 0;
}
int isFull(int*queue){
    if(queue[1]==99)
        return 1;
    return 0;
}
void display(int*queue){
int i;
for(i=queue[0];i<=queue[1];i++)
    printf("%d\n",queue[i]);
}

int main()
```

```
{
  int queue[100]; //initialization
  queue[0]=1;    //of an empty front=queue[0]
  queue[1]=1;    //queue      rear=queue[1]
  int ch,val;
  for(;;){
  printf("\n\n****Queue operations****\n\n");
  printf("\n\nEnter the choice\n");
  l:printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
  scanf("%d",&ch);
  switch (ch){
    case 1: if(isFull(queue)){
        printf("Queue overflow\n");
        printf("Enter any other choice\n");
        goto l;
        }
        printf("\n\nEnter the element to insert into the queue\n");
        scanf("%d",&val);
        Insert(queue,val);
    break;
    case 2:if(isEmpty(queue)){
        printf("Queue is empty\n");
        printf("Enter any other choice\n");
        goto l;
      }
      printf("\n\nThe element dequeued is %d \n",Delete(queue));
      break;
    case 3:if(isEmpty(queue)){
        printf("Queue is empty\n");
        printf("Enter any other choice\n");
        goto l;
```

```
            }

            display(queue);

            break;

    case 4:exit(0);

        break;

     default: printf("Wrong choice\n");


}


}  return 0;}
```