Department of Information & Communication Technology
MIT, Manipal
III Sem B. Tech (IT/CCE)
Data structures (ICT 2153), In-sem Examination scheme
Date: 14/12/2021 **Max. Marks: 20**

1 Write a user defined function that takes prefix expression as an argument, converts the prefix expression to fully parenthesized infix expression and displays the same. Evaluate the expression: 9 6 7 5 – 3 % * 4 / + and show the content of the stack at each step. **3**

[prefix to fully parenthesized infix expression function code: 1m, Stack class with constructor, push and pop functions:1M
Expression evaluation along with each step: 1m]

```
#define MAX 30
class STACK
{int TOP;
char stack[MAX][MAX];
public:
STACK(){TOP=-1;}
void push(char []);
char* pop();
};
void STACK::push(char opernd[])
{ if (TOP ==MAX -1)
      cout<<"stak is full";
  else
      strcpy(stack[++TOP],opernd);
}
char* STACK::pop()
{
if(TOP==-1) return "X";
else return stack[TOP--];
}
void postfix_infix(char prefix[])
{
STACK s;
int c,cop,j,i=0;
char oper[20],op1[20],op2[20];char symbol[20];
while(prefix[i]!='\0') i++;
len=i;
for(i=len-1;i>=0;i--)
{char temp1[20],temp2[20],temp3[20];
temp1[0]=prefix[i]; temp1[1]='\0';
strcpy(symbol,temp1);
if(!isalpha(symbol[0]))
{
strcpy(op1,s.pop());
strcpy(op2,s.pop());
strcpy(temp3,"(");
strcat(temp3,op1);
strcat(temp3,symbol);
strcat(temp3,op2);
strcat(temp3,")");
s.push(temp3);
}
else s.push(temp1);
}
cout<<" fully parenthesized  infix expression is = "<<s.pop();
}
```

	9 push(9) 9 6 push(6) 9 6 7 push(7) 9 6 7 5 push(5) 9 6 7 5 – op2=pop() 7, op1=pop() 5, push(7-5) 9 6 2 3 push(3) 9 6 2 3 % op2=pop() 3, op1=pop() 2, push(2%3) 9 6 2 * op2=pop() 2, op1=pop() 6, push(6*2) 9 12 4 push(4) 9 12 4 / op2=pop() 4, op1=pop() 12 , push(12/4) 9 3 + op2=pop() 3,op1=pop() 9, push(9+3) 12 **Result: 12**	
2	A doubly linked list is used to store student records. For every student, the registration number, name, semester, and branch need to be stored. Write the class definition with suitable data members and member functions to perform the following operations on doubly linked list. a. Insert a new record into a sorted list such that registration number-wise order is maintained. **[1.5m]** b. Delete records having duplicate registration numbers **[1.5m]**	

```
class student
{
int regno;
char name[20];
int sem;
char branch[10];
student *prev;
student *next;
public:
student(){}
student(int r,char n[],int s,char b[]){
regno=r;strcpy(name,n);sem=s;strcpy(branch,b);
next=NULL; prev=NULL;
}
void insert_sorted(int r,char n[],int s,char b[]);
student* deleteNode(student *f, student *d);
student* removeDuplicates(student *f) ;
}


 void student::insert_sorted(int r,char n[],int s,char b[])
{
student *s1=new student(r,n,s,b);
student *c,*prev1=NULL;
if(!first)
{       first=s1; return;}
else if(first->regno>s1->regno)
{
   s1->next=first;
   first->prev=s1;
   first=s1;
   return;
}
prev1=first;
for(c=first->next;c!=NULL;c=c->next)
{
if(s1->regno<c->regno)
break;
prev1=c;
}
if(c)
```

```
{
s1->prev=c->prev;
s1->next=c;
c->prev=s1;
if(prev1) prev1->next=s1;
}
else
{
prev1->next=s1;
s1->prev=prev1;
}
}
student* student::deleteNode(student *f, student *d)
{
if(f==NULL||d==NULL) return f;
if (f==d) f=d->next;
if (d->next != NULL) d->next->prev = d->prev;
if (d->prev != NULL) d->prev->next = d->next;
delete(d);
return f;
}
student* student::removeDuplicates(student *f)
{
if (f==NULL ||f->next == NULL) return f;
   student *c1, *c2,*n;  //n: next,
for (c1=f;c1!=NULL; c1=c1->next) {
c2 = c1->next;
while(c2 != NULL){
if (c1->regno==c2->regno) {
n = c2->next;
deleteNode(f,c2);
c2=n;
}
else  c2 = c2->next;
}
}
return(f);
}
```

3	Write user defined functions to perform the following: i. create a circular singly linked list ii. delete the nodes which contain the prime number from the circular singly linked list. **(Create: 1M; Del: 2M)**	**3**

i.create circular SLL

```
cnode* cnode::insrt(cnode *head)
{
 cnode *temp=new cnode;
 cnode *cur;
 cout<<"Enter the value to be
inserted:";
 cin>>temp->info;
 temp->next=NULL;
 if(head==NULL) {
  head=temp;
  temp->next=head;
 }
 else {
   cur=head;
   while(cur->next!=head)
           cur=cur->next;
   cur->next=temp;
   temp->next=head;
 }
 return head;
}

int isPrime(int n)
{
   if (n <= 1)
      return 0;
   for (int i = 2; i < n; i++)
      if (n % i == 0)
         return 0;

   return 1;
}
```

ii.delete nodes

```
cnode* cnode::delfe(cnode *head)
{
  cnode *cur;
  if(head == NULL)
  {
    cout<<"no records to delete!"<<endl;
    return NULL;
  }
  else if(head->next == head)
  {
     if(isPrime(head->info))
     {
        cout<<"Element deleted is: "<<head->info<<endl;
        delete(head);
        return NULL;
     }
  }
  else
  {
     cur = head;
     while((cur->next)!=head)
     {
        prev = cur;
        cur = cur->next;
        if(isPrime(cur->info))
        {
          prev -> next  = cur->next;
          delete cur;
        }
     }
  }
  return head;
}
```

4 Write a program to delete a given character (user input) from a string. Consider multiple occurrences of the character. **(Read string: 1M; Del: 2M)** **3**

```
class dchar
{
   char str[50];
   char key;
public:
   void getstr()
   {
      cout<<"Enter the string:";
      cin>>str;
      cout<<"Enter the character to be deleted:";
      cin>>key;
   }
```

```
  void deletechar(){
     int i, j;
     for(i=0;i<strlen(str);i++)
     if(str[i]==key){
        for(j=i;j<strlen(str);j++)
        str[j]=str[j+1];}
     str[j]='\0';
     cout<<str;
   }
};
int main(){
dchar d;
d.getstr();
d.deletechar();
return 0;}
```

5 **Write the inorder, postorder, preorder and level order traversal sequence for the tree shown in Figure 1. Write recursive user-defined functions to** **4**

a. Count and return the total number of NULL links in the binary tree (1M)

```
int tree::inorderr(node *ptr, int *nl)
{

   if(ptr)
   {  inorderr(ptr->lchild,nl);
       inorderr(ptr->rchild,nl);
   }
else
   {*nl=*nl+1;}

   return(*nl);
}
```

b. Display the number of non-empty right subtrees and display the root of every right subtree. (1M)

```
int tree::inorder_rchild(node *ptr)
{
  static int rch;

   if(ptr)
   {   if (ptr->rchild) {cout<<ptr->data<<" ";rch++;}
      inorder_rchild(ptr->lchild);
    //  cout<<ptr->data<<" "<<rch<<" "; l++;
    //   if(ptr==key) return rch;
      inorder_rchild(ptr->rchild);
   }
 return rch;
}
```

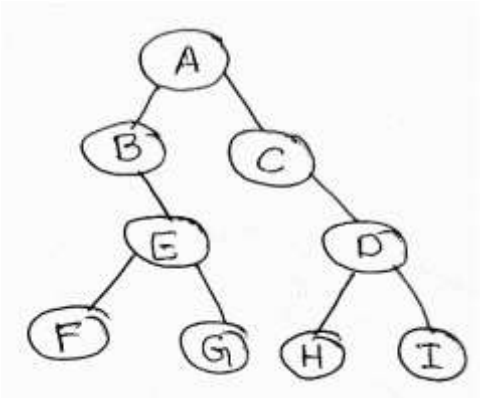

Figure 1

[inorder: 0.5m, preorder: 0.5m, postorder:0.5m, level order:0.5m]

Inorder: BFEGACHDI
Preorder: ABEFGCDHI
Postorder: FGEBHIDCA
Level order: ABCEDFGHI

6 **(a).** Given a Circular Singly Linked List (CSLL) containing 0s and 1s as data values, write a user defined function to find its decimal equivalent. For example, if list is 1-->0-->0-->0-->1 then the expected output is 17. Decimal value of an empty list is considered as 0. **4**

(b). Define a class to represent a sparse matrix in the form of an array of object. Write a user defined function to insert a non-zero element into a sparse matrix represented as array of object. Assume that the sparse matrix is already created and passed into the insert function as parameter. Make sure that, the row major order is maintained during the insert process.

((a) Decimal calculation: 1M; Empty case:0.5M; traversal: 0.5M;
(b) Class definition:0.5M; Reading rows, col, non-zero: 0.5M; traversing, checking and deleting: 1M)

(a)

```
nt cnode::decimalValue(cnode *head, cnode *t){
 int res = 0;
 static int p = 0;
 if(head==NULL){
   res=0;
   cout<<res;
   return res;
 }
 if(t->next == head)
 {
   res = t->info * pow(2, p++);
   return res;
 }
 res = decimalValue(head, t->next);
 res = res + t->info * pow(2, p++);
 return (res);
}
```

(b)

```
void spm::spm_insert(spm a[])
{
   int r, c, nz, pos;
  cout<<"Enter the row and column of the non-zero
element: \n";
  cin>>r>>c;
  cout<<"Enter the non zero element \n";
  cin>>nz;
  for(int i = 1; i<a[0].val; i++)
  {
   if(r > a[i].row)
          continue;
   if(r == a[i].row)
     {
        if(c > a[i].col)
             continue;
     }
    pos = i;
   for(int j = a[0].val; j>pos-1; j--)
   {
      a[j+1] = a[j];
    }
   a[pos].row =r;
   a[pos].col = c;
   a[pos].val = nz;
  }
  a[0].val++;
}
```