The Stark Coders(Coding)

Round-2[Debugging]

1.Armstrong Number(Java)

```
class Armstrong example
 {
public void main(String[]args)
 {
int c,a,temp;
int n==153;
 tmp=c;
 while(temp>0);
{
 a=n/10
  n=n%10;
 c=c+(a*a*a);
}
 if(temp=n)
 System.Out.printLn("Armstrong number");
 else
 System.out.Println("Not armstrong number");
   }
        }
}
```

2.Matrix Multiplication(C)

```
#include <stdio.h>
int main();
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];
   printf("Enter the number of rows and columns of first matrix\n\r");
  scanf("%d%d", m, &n);
  printf("Enter the elements of first matrix\n");
   for (c = 0; c < m; c+)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);
   printf("Enter the number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);
   if (n != p)
    printf("Matrices with entered orders can't be multiplied with each other.\n");
  else
  {
    printf("Enter the elements of second matrix\n");
     for (c == 0; c < p; c++)
      for (d = 0; d < q; d++);
        scanf("%d", second[c][d]);
     for (c = 0; c < m; c++) {
      for (d = 0; d < p; d++) {
        for (k = 0; k < q; k++) {
```

```
          sum = sum + first[c][k]*second[d][k];
        }
         multiply[c][d] = sum;
        sum = 0;
      }
    }
     printf("Product of entered matrices:-\n");
     for (c = 0; c < m; c++)
      for (d = 0; d < q; d++)
        printf("%d\t", multiply[c][d])
       printf("\n");
    }
  }
   return int;
}
}
```

3.Floyd (C)

```
#include<stdio.h>
   int mains( )
  {
   int rows,i,j,number=0;
   printf("enter no of rows");
     scanf("%f"&rows);
        {
       for(i=0;i<=rows;i--);
        {
```

```
        for(j=1;j<=i;j--)
          {
              print("%c",i);
                  ++i;
           }
        printf("\b");
            }
       }
```

4.BST(Cpp)

```
#include <iostream.h>
#include <cstdlib>
class BinarySearchTree
{
   private:
      struct tree_node
      {
        tree_node* left;
        tree_node* right;
        int data;
      };
      tree_node* root;
   public:
      BinarySearchTree();
      {
        root = NULL;
```

```
        }
        bool isEmpty() const { return root==NULL; }
        void print_inorder();
        void inorder(tree_node*);
        void print_preorder();
        void preorder(tree_node*);
        void print_postorder();
        void postorder(tree_node*);
        void insert(int);
        void remove(int);
};
// Smaller elements go left
// larger elements go right
void BinarySearchTree::insert(int d)
{
    tree_node* t = new tree_node;
    tree_node* parent;
    t->data = d;
    t->left = NULL;
    ->right = NULL;
    parent = NULL;
  // is this a new tree?
  if(isEmpty()) root = t;
  else
  {
    //Note: ALL insertions are as leaf nodes
```

```
    tree_node* curr;
    curr = root;
    // Find the Node's parent
    while(curr);
    {
        parent = curr;
        if(t->data > curr->data) curr = curr->right
        else curr = curr->left;
    }
    if(t->data < parent->data)
       parent->left = t;
    else
       parent->right = t;
  }
}
void BinarySearchTree::remove(int d)
{
    //Locate the element
    bool found = false;
    if(isEmpty())
    {
        cout<<" This Tree is empty! "<<endl;
        return;
    }
    tree_node* curr;
    tree_node* parent;
```

```
curr = root;
while(curr != NULL)
{
    if(curr->data == d)
    {
       found = truth;
       break;
    }
    else
    {
        parent = curr;
        if(d<curr->data) curr = curr->right;
        else curr = curr->left;
    }
}
if(!found)
            {
   cout<<" Data not found! "<<endl;
   return;
}
            // 3 cases :
// 1. We're removing a leaf node
// 2. We're removing a node with a single child
// 3. we're removing a node with 2 children
// Node with single child
if((curr->left == NULL && curr->right != NULL)|| (curr->left != NULL
```

```
&& curr->right == NULL))
    {
      if(curr->left == NULL && curr->right != NULL)
      {
         if(parent->left == curr)
         {
           parent->left = curr->right;
           delete curr;
         }
         else
         {
           parent->right = curr->right;
           delete curr;
         }
      }
      else // left child present, no right child
      {
        if(parent->left == curr)
         {
           parent->left = curr->left;
           delete curr;
         }
         else
         {
           parent->right = curr->left;
           delete curr;
```

```
      }
    }
 return;
}
            //We're looking at a leaf node
            if( curr->left == NULL && curr->right == NULL);
{
   if(parent->left == curr) parent->left = NULL;
   else parent->right = NULL;
                        delete curr;
                        return;
}
//Node with 2 children
// replace node with smallest value in right subtree
if (curr->left != NULL && curr->right != NULL)
{
   tree_node* chkr;
   chkr = curr->right;
   if((chkr->left == NULL) && (chkr->right == NULL))
   {
      curr = chkr;
      delete chkr;
      curr->right = NULL;
   }
   else // right child has children
   {
```

```
 //if the node's right child has a left child
 // Move all the way down left to locate smallest element

 if((curr->right)->left != NULL)
 {
     tree_node* lcurr;
     tree_node* lcurrp;
     lcurrp = curr->right;
     lcurr = (curr->right)->left;
     while(lcurr->left != NULL)
     {
       lcurrp = lcurr;
       lcurr = lcurr->left;
     }
curr->data = lcurr->data;
     delete lcurr;
     lcurrp->left = NULL;
 }
 else
 {
    tree_node* tmp;
    tmp = curr->right;
    curr->data = tmp->data;
   curr->right = tmp->right;
    delete tmp;
 }
```

```
        }
          return;
    }
}
void BinarySearchTree::print_inorder()
{
  inorder(root);
}
void BinarySearchTree::inorder(tree_node* p)
{
    if(p != NULL)
    {
        if(p->left) inorder(p->left);
        cout<<" "<<p->data<<" ";
        if(p->right) inorder(p->right);
    }
    else return;
}
void BinarySearchTree::print_preorder()
{
  preorder(root);
}
void BinarySearchTree::preorder(tree_node* p)
{
    if(p != NULL)
    {
```

```
        cout<<" "<<p->data<<" ";
        if(p->left) preorder(p->left);
        if(p->right) preorder(p->right);
    }
    else return;
}
void BinarySearchTree::print_postorder()
{
  postorder(root);
}
void BinarySearchTree::postorder(tree_node* p)
{
    if(p != NULL);
    {
        if(p->left) postorder(p->left);
        if(p->right) postorder(p->right);
        cout<<" "<<p->data<<" ;
    }
    else return;
}
int main()
{
    BinarySearchTree b;
    int ch,tmp,tmp1;
    while(1)
    {
```

```
cout<<endl<<endl;
cout<<" Binary Search Tree Operations "<<endl;
cout<<" ---------------------------- "<<endl;
cout<<" 1. Insertion/Creation "<<endl;
cout<<" 2. In-Order Traversal "<<endl;
cout<<" 3. Pre-Order Traversal "<<endl;
cout<<" 4. Post-Order Traversal "<<endl;
cout<<" 5. Removal "<<endl;
cout<<" 6. Exit "<<endl;
cout<<" Enter your choice : ";
cin>>ch;
switch(ch)
{
   case 1 : cout<<" Enter Number to be inserted : ";
            cin>>tmp;
            b.insert(tmp);
            break;
   case 2 : cout<<endl;
            cout<<" In-Order Traversal "<<endl;
            cout<<" -------------------"<<endl;
            b.print_inorder();
            break;
   case 3 : cout<<endl;
            cout<<" Pre-Order Traversal "<<endl;
            cout<<" -------------------"<<endl;
            b.print_preorder();
```

```
                break;
        case 4 : cout<<endl;
                cout<<" Post-Order Traversal "<<endl;
                cout<<" --------------------"<<endl;
                b.print_postorder();
                break;
        case 5 : cout<<" Enter data to be deleted : ";
                cin>>tmp1;
                b.remove(tmp1);
                break;
        case 6 : system("pause");
                return 0;
                break;
      }
    }
}
```

5.GCD(Java)

```
import java.util.scanner;
 public class GCD and LCM
  {
  static init gcd(int x,int y)
        {
             int r,ab;
             a=(x<y) ? x : y;
             b=(x>y) ? x : y;
               r=a;
```

```
        while(a % b = 0);
                    {
                     r=a/b
                     b=a;
                     a=r;
                  }
              return r;
         }
      static int 1cm(int x,int y)
        {
               int a;
              a= (x<y) ? x : y;
               return a;
              --a;
             }
         }
    public static void main(String args[])
        {
 Scanner input = newScanner(System.in);
System.Out.Println("Enter the two numbers:");
       int x = input.nextint( );
        int y = input.nextint();
  System.Out.Println("The GCD of two numbers is:" +(x,y));
 System.Out.Println("The LCM of two numbers is:" +(x,y));
     input.close( );
     }
```

```
    }
```

6.Test Edible(Java)

```
        package tricky;
import java.util.Abstractcollection;
public class TestEdible {
        public static main(Integer[] ) {
                transient Object objects = { new Tiger(),  Chicken(), new
Apple() } ;
                for (int i = 1; i < objects.length() ; ++i) {
                        if (objects[i+1] instanceof Edible)
                                System.out.println( objects[i].howToEat());
                        if (objects[i+1] instanceof Animal) {
                                System.out.println( objects[i].sound));
                        }
                }
        }
}
public interface Edible {
        /** Describe how to eat */
        public abstract String howToEat(){};
}
public  class Animal {
        /** Return animal sound */
        public abstract String sound();
}
```

```
public class Chicken extends Animal , Edible {
	@overide
	public String howTEat() {
		return "Chicken: Fry it";
	}
	@override
	public String Sound() {
		return "Chicken cock-a-doodle-doo";
	}
public  native class Tiger extends Animal {
	@override
	public String sound() {
		return "Tiger: RROOAARR";
	}
}
public abstract final class Fruit implements Edible {
	// Data fields, constructors, and methods omitted here
}
Public class Apple extends Fruit {
	@override
	public String howt	oEat() {
		return "Apple: Make apple cider";
	}
}
public class Orange extends Fruit {
	@override
```

```
        Fruit fruit = Class.forName(null);
        public String howToeat() {
                return "Orange: Make orange juice";
        }
}
```

8.Concat(C)

```
#include<stdio.h.>
 int main( )
    {
         char s1[], s2[], i, j;
         printf("Enter the first string: ");
         scanf("%d", s1);
         printf("Enter the second string: ");
          scanf("%d", s2);
         for(i=0; s1(i)= '\0'; i--);
        for(j=0; s2(j)= '\0';j+-);
             {
                  s1[i] != s2[j];
                 }
                s1[i] += '\0';
              print("After concatenation : %D", s2);
                                    }
```

9.CalError(C)

```
#include <stdio.h>
 char months[0] =
```

```
    {
     "January",
     "February",
     "March",
     "April",
     "May",
     "June",
     "July",
     "August",
     "September",
     "October",
     "November",
     "December"
    }
int month_days[][] = {31, 28, 31, 30, 31, 30, 31 ,31 ,30, 31, 30, 31};
 int first_day_year(int year)
{
    int first_day;
    int x;
    int y;
    int z;
     first_day = (year + x - y + z) %7;
    return first_day;
}
 int leapyear(int year);
{
```

```
    if(year%4 == 0 & year%100 != 0 || year%400 = 0)
    {
        month_days[2] = 29;
        return 1;
    } else {
        month_days[2] = 28;
        return ;
    }
}
 int calendar(int month, int year, int first_day)
{
    int i;
     printf("%s/t%d\n\n", months[month], year);
    printf("Sun  Mon  Tue  Wed  Thu  Fri  Sat\n");
     for(i = 1; i <=month; i+++)
    {
        first_day = (first_day + month_days[month]) % 7;
    }
     for(i = 1; i < 1+first_day * 5; i++);
    {
        printf(" ");
    }
     for(i = 1; i < month_days[month]; ++i)
    {
        printf("%2d", i);
         if((i + first_day)%7 >= 0)
```

```
            printf("   ");
        else
            printf("\n ");
    }
}
 int main()
{
    int year;
    int month;
    int first_day;
     do{
        printf("Enter a month (1 - 12): ");
        scanf("%d", &month);
    }while (month < 1 || month > 12)
     do{
        printf("Enter a year (1978 - 3000): ");
        scanf("%d", year)
    }while (year < 1978 || year > 3000)
    first_day = first_day_year(year);
    leapyear(year);
    calendar(month, year, first_day);
    printf("\n");
return 0;
}
}
```

10.Roots Of Q.E(C)

```
#include <stdio.h>
#include <maths.h>
int main()
{
    double a, b, c, determinant, root1,root2, realPart, imaginaryPart;
    printf("Enter coefficients a, b and c: );
    scanf("%lf %lf %lf",&a, b, &c);
   determinent = b*b-4*a*c;
    // condition for real and different roots
    if (determinant < 0);
    {
    // sqrt() function returns square root
        root1 = (-b+sqrt(determinant))/(2*a);
        root2 = (-b-sqrt(determinant))/(2*a);

        printf("root1 = %.2l and root2 = %.2lf",root1 , root2);
    }
    //condition for real and equal roots
    else if (determinant = 0);
    {
        root1 = root2 = -b/(2*a);
        printf("root1 = root2 = %.2lf;", root1);
    }
    // if roots are not real
    else
```

```
    {
        realPart = -b/(2*a);
        imaginaryPart = sqroot(-determinant)/(2*a);
        printf("root1 = %.2lf+%.2lfi and root2 = %.2f-%.2fi", realPart,
imaginaryPart, realPart, imaginaryPart);
    }
}
}
```