Harsh_Kumar_MAN_106_Assignment_2

- Answer 1 :

```cpp
#include<bits/stdc++.h>
using namespace std;

#define MAX 20

class stack_type{
    int top;
    char stac[MAX];
public:
    stack_type();
    bool empty_stack();
    void push(char item);
    char pop();
};

stack_type::stack_type(){
    top = -1;
}

void stack_type::push(char item){
    if (top==MAX) {
        cout << "Stack Overflow\nPop out some values first";
        return;
    }
    else stac[++top] = item;
}

char stack_type::pop(){
    if (top==-1) {
        cout << "Stack Underflow\nPush in some values first";
        return -1;
    }
    else {
        return (stac[top--]);
    }
}

bool stack_type::empty_stack(){
    if (top==-1) return true;
    else return false;
}
```

```cpp
int main(){
    stack_type S;
    char st[MAX], rev_st[MAX];
    cin >> st;
    bool flag = true;
    int i=0;
    for(; st[i]!='\0'; i++) S.push(st[i]);
    int k=i;
    for(; i>0; i--) {
        rev_st[k-i] =  S.pop();
        cout << rev_st[k-i] << " ";
    }

    cout << "\n";

    for(int i=0; i<k; i++) if (rev_st[i] != st[i]) flag = false;

    if (flag) cout << "The string is a palindrome\n";
    else cout << "Not a palindrome\n";

}
```

Output :

```
radar

r a d a r
The string is a palindrome
```

```
iit

t i i
Not a palindrome
```

- Answer 2

```
#define MAX 50

class STACK{
    int top;
    float stac[MAX];
public:
    STACK();
    bool underflow();
    bool overflow();
    void push(float item);
    float pop();
};

STACK::STACK(){
    top = MAX;
}

bool STACK::underflow(){
    if (top==-1) return true;
    else return false;
}

bool STACK::overflow(){
    if (top==MAX) return true;
    else return false;
}

void STACK::push(float item){
    if (overflow()) {
        cout << "Stack Overflow\nPop out some values first";
        return;
    }
    else stac[++top] = item;
}

float STACK::pop(){
    if (top==-1) {
        cout << "Stack Underflow\nPush in some values first";
        return -1.0;
    }
    else {
        return (stac[top--]);
    }
```

- Answer 3

```cpp
#include<bits/stdc++.h>
using namespace std;

#define MAX 10

class TwoStack{
    int top1;
    int top2;
    int stac[MAX];
public:
    TwoStack();
    void push1(int item);
    void push2(int item);
    int pop1();
    int pop2();
    bool is_collision();
};

TwoStack::TwoStack(){
    top1 = -1;
    top2 = MAX;
}

void TwoStack::push1(int item){
    if (is_collision()) cout << "Stack Overflow\nPop out some values first\n";
    else stac[++top1] = item;
}

void TwoStack::push2(int item){
    if (is_collision()) cout << "Stack Overflow\nPop out some values first\n";
    else stac[--top2] = item;
}

int TwoStack::pop1(){
    if (top1==-1) cout << "Stack Underflow\nPush some values first\n";
    else return(stac[top1--]);
}

int TwoStack::pop2(){
    if (top2==MAX){
        cout << "Stack Underflow\nPush some values first\n";
        //return -1;
    }
    else{
        return(stac[top2++]);
    }
}

bool TwoStack::is_collision(){
    if (top2-top1 < 1) return true;
    else return false;
}


int main(){
    TwoStack S;
    for(int i=1; i<6; i++) {
        S.push1(i); S.push2(10-i); }
    for(int i=0; i<5; i++){
        cout << S.pop1() << " :From Stack1, " << S.pop2() << " :From Stack2\n";
    }
}
```

Output :

```
5 :From Stack1, 5 :From Stack2
4 :From Stack1, 6 :From Stack2
3 :From Stack1, 7 :From Stack2
2 :From Stack1, 8 :From Stack2
1 :From Stack1, 9 :From Stack2
```

- Answer 4 :

```cpp
#include<bits/stdc++.h>
using namespace std;

#define MAX 50

class STACK{
    int top;
    int stac[MAX];
public:
    STACK();
    int peep(int index);
    void push(int item);
    int pop();
};

STACK::STACK(){
    top=-1;
}

int STACK::peep(int index){
    if (index > top) cout << "Can't peek. Index out of bounds";
    else return stac[top-index+1];
    return -1;
 }

void STACK::push(int item){
    if (top==MAX) {
        cout << "Stack Overflow\nPop out some values first";
        return;
    }
    else stac[++top] = item;
}

int STACK::pop(){
    if (top==-1) {
        cout << "Stack Underflow\nPush in some values first";
    }
    else {
        return (stac[top--]);
    }
    return -1;
}
```

```cpp
int main(){
    STACK s;
    for(int i=0; i<10; i++) s.push(i);

    cout << s.pop() << endl;
    cout << s.peep(3) << endl;
    cout << s.peep(30) << endl;
    cout << s.pop() << endl;
}
```

Output :

```
9
6
Can't peek. Index out of bounds-1
8
```

5. Infix to postfix conversion :

(i) (A - B) * (C / D) ---> A B - C D / *

(ii) (A + B ^ D) * (E - F) + G ---> A B D ^ + E F - * G +

(iii) A *(B + D) / E - F * (G + H / K) ---> A B D + * E / F G H K / + * -

(iv) (A + B) * (C ^ (D - E) + (F/G) ^ (H - J)) ---> A B + C D E - ^ F G / H J - ^ + *

Infix to prefix conversion :

(i) (A - B) * (C / D) ---> * - A B / C D

(ii) (A + B ^ D) * (E - F) + G ---> + * + A ^ B D - E F G

(iii) A *(B + D) / E - F * (G + H / K) ---> - / * A + B D E * F + G / H K

(iv) (A + B) * (C ^ (D - E) + (F/ G) ^ (H - J)) ---> A B +

6. Prefix to Infix :

i. + A - B C ---> A + (B - C)

ii. + + A - * ^ B C D / + E F * G H I ---> (A + (((B ^ C) * D) - ((E + F) / (G * H)))) + I

```
iii.  + - ^ A B C *D * * E F G          ---> (A ^ B) - C) + D * (G * (E * F))



7. Postfix to Infix:

i. A B +C -                             ---> (A + B) - C

ii. A B C + -                           ---> A - (B + C)

iii. A B - C + D E F - + ^              ---> ((A - B) + C) ^ (D + (E - F))

iv. A B C D E - +  ^  * E F * -         ---> (A * (B ^ ( C + ( D - E )))) + ( E*F )

v. A B + C * D E - - FG+ ^              ---> (((A + B) * C) - (D - E)) ^ ( F + G)
```

- Answer 8:

```cpp
/*

For the tower of hanoi problem we can use this recursive realtion :
    Move n-1 disks from the source tower to the auxilary tower
    Move the last disk to the destination
    Move the n-1 diska back from auxilary tower to destination tower

    Accordidng to this realtion :

        a(n) = 1 + 2*(a(n-1))

*/

#include<bits/stdc++.h>
using namespace std;

#define MAX 50

class STACK{
    int top;
    int stac[MAX];
    string name;
public:
    STACK(string A);
    STACK();
    bool empty_stack();
    string push(int item);
    pair<int, string> pop();
};

STACK::STACK(){
    top = -1;
    name = "default";
}

STACK::STACK(string A){
    top = -1;
    name = A;
}

string STACK::push(int item){
    if (top==MAX) {
        cout << "Stack Overflow\nPop out some values first";
        return name;
    }
    else {
        stac[++top] = item;
        return name;
    }
}

pair<int, string> STACK::pop(){
    if (top==-1) {
        cout << "Stack Underflow\nPush in some values first";
        return (make_pair(-1, "-1"));
    }
    else {
        return (make_pair(stac[top--], name));
    }
}

bool STACK::empty_stack(){
    if (top==-1) return true;
    else return false;
}

void TOH(int n, STACK& source, STACK& destination, STACK& auxilary){
    if (n==1){
        auto popped = source.pop();
        cout << "Disk " << popped.first << " from " << popped.second << " transferred to " << destination.push(popped.first) << '\n';
        return ;
    }
    TOH(n-1, source, auxilary, destination);
    auto popped = source.pop();
    cout << "Disk " << popped.first << " from " << popped.second << " transferred to " << destination.push(popped.first) << '\n';
    TOH(n-1, auxilary, destination, source);
}

int main(){
    STACK source("A"), destination("C"), auxilary("B");
    int n=4;
    for(int i=n; i>0; i--) source.push(i);

    TOH(n, source, destination, auxilary);

}
```

Output :

For 4 Disks

```
Disk 1 from A transferred to B
Disk 2 from A transferred to C
Disk 1 from B transferred to C
Disk 3 from A transferred to B
Disk 1 from C transferred to A
Disk 2 from C transferred to B
Disk 1 from A transferred to B
Disk 4 from A transferred to C
Disk 1 from B transferred to C
Disk 2 from B transferred to A
Disk 1 from C transferred to A
Disk 3 from B transferred to C
Disk 1 from A transferred to B
Disk 2 from A transferred to C
Disk 1 from B transferred to C
```

- Answer 9:

```
A. Conversion of infix to postfix :
    1. Scan form left to right
    2. If the element is an operand, output it
    3. Else check the precedence of the scanned operator
        a. If it is greater or equal than the element already on the stack or the stack is empty, oush this operator on the stack
        b. Else, pop all the operators from the stack which are greater in precedence and then push this operator
    4. If the scanned character is '(', push it on the stack
    5. Else if the scanned character is ')', pop all the values until a '(' is reached . Pop this '(' too.
    6. Loop through 2-6 until the stack is empty
    7. Pop out all the remaining values in the stack, after the output
    8. Exit

B. Conversion from infix to prefix
    1. Reverse the given infix expression
    2. Convert this reversed infix expression into postfix form
    3. Reverse the expression obtained in 2 and print it
    4. Exit

C. Evaluation of postfix expression
    1. Scan the expression from left to right
    2. If the elemant is operand push it to stack
    3. Else if the elemant is operator:
        a. Pop two elements from stack
        b. perform the operation using the operator in hand
        c. push the expression to the stack
    4. Repeat step 2-3 until the end of expression is reached
    5. Exit

D. Evaluation of postfix expression
    1. Reverse the given expression
    2. Convert the reversed expression to infix by using postfix to infix converison
    3. Reverse the expression obtained in step 2 an print it
    4. Exit
```