==================================================================================
circular Queues:

```
#include <bits/stdc++.h>

using namespace std;

struct queues
{
    int size;
    int front = -1;
    int rear = -1;
    int *q;
};

void enqu(queues &qu, int number)
{
    if ((qu.rear + 1) % qu.size == qu.front)
    {
        cout << "Full";
    }
    else
    {
        qu.rear = (qu.rear + 1) % qu.size;
        qu.q[qu.rear] = number;
    }
}

int deq(queues &qu)
{
    int x = -1;
    if (qu.front == qu.rear)
    {
        cout << "faka" << endl;
    }
    else
    {
        qu.front = (qu.front + 1) % qu.size;
        x = qu.q[qu.front];
    }
    if (qu.front == qu.rear)
    {
        qu.front = qu.rear = -1;
    }
    return x;
}

void display(queues su)
{
    for (int i = su.front + 1; i <= su.rear; i++)
    {
        cout << su.q[i] << " ";
    }
    cout << endl;
}
```

```
================================================================================
LinkedList:
#include <bits/stdc++.h>

using namespace std;

struct node
{
    int data;
    node *next;
};
void print(node *head)
{
    cout << "ELEMENTS: ";
    node *temp = head;
    while (temp != 0)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void print2(node *tmp)
{
    if (tmp != NULL)
    {
        cout << tmp->data << " ";
        print2(tmp->next);
    }
}

int len(node *head)
{
    int le = 0;
    node *p = head;
    while (p != 0)
    {
        p = p->next;
        le++;
    }
    return le;
}
int len2(node *p)
{
    if (p == 0)
    {
        return 0;
    }
    return 1 + len2(p->next);
}

int sum(node *head)
{
    int ans = 0;
    node *tem = head;
    while (tem != 0)
    {
        ans += tem->data;
        tem = tem->next;
    }
    return ans;
}

int sum2(node *head)
{
    if (head == NULL)
    {
        return 0;
    }
    return head->data + sum2(head->next);
}

void create(node *&head)
{
    node *newnode = new node();
    cin >> newnode->data;
    newnode->next = 0;
    if (head == 0)
    {
        head = newnode;
    }
    else
    {
        node *temp = head;
        while (temp->next != 0)
        {
            temp = temp->next;
        }
        temp->next = newnode;
    }
    return;
}

int maxi(node *head)
{
    int mx = INT_MIN;
    node *tm = head;
    while (tm != 0)
    {
        if (tm->data > mx)
        {
            mx = tm->data;
        }
        tm = tm->next;
    }
    return mx;
}

int mini(node *head)
```

```
{
    int mx = INT_MAX;
    node *tm = head;
    while (tm != 0)
    {
        if (tm->data < mx)
        {
            mx = tm->data;
        }
        tm = tm->next;
    }
    return mx;
}

bool search(node *head)
{
    int n;
    cin >> n;
    node *tm = head;
    while (tm != 0)
    {
        if (tm->data == n)
        {
            return true;
        }
        tm = tm->next;
    }
    return false;
}

bool searchEx(node *&head)
{
    int x;
    cin >> x;
    node *prv = 0;
    node *now = head;
    while (now != 0)
    {
        if (x == now->data)
        {
            prv->next = now->next;
            now->next = head;
            head = now;
            return true;
        }
        prv = now;
        now = now->next;
    }
    return false;
}

node *searchh(node *p, int n)
{
    while (p != NULL)
    {
        /* code */
        if (p->data == n) // peye gele ekhane dhubke
        {
            return p;
        }
        else
            p = p->next;
    }
    return NULL; // Na paile ei condition e dhukbe
}

void insert_fast(node *&head)
{
    node *newnode = new node();
    cin >> newnode->data;
    if (head == 0)
    {
        head = newnode;
        return;
    }
    newnode->next = head;
    head = newnode;
    cout << "inserted before first" << endl;
}

void insert_pos(node *&head)
{
    int pos;
    cout << "pos must be greater then 0 and less than length(value, position) " << len(head);
    node *newnode = new node();
    cin >> pos >> newnode->data;
    node *tm = head;
    if (pos == 0 || pos > len(head))
    {
        cout << "Wrong" << endl;
        return;
    }
    if (pos == 1)
    {
        newnode->next = head;
        head = newnode;
        return;
    }
    for (int i = 0; i < pos - 2; i++)
    {
        tm = tm->next;
    }
    newnode->next = tm->next;
    tm->next = newnode;
}

void insert_in_sorted(node *&head)
{
    node *newnode = new node();
    cin >> newnode->data;
    node *tmp = head;
    if (head == 0)
    {
        head = newnode;
        return;
    }
    if (head->next == 0)
    {
        if (head->data > newnode->data)
        {
            newnode->next = head;
            head = newnode;
        }
        else
        {
            head->next = newnode;
        }
        return;
    }
    if (head->data >= newnode->data)
    {
        newnode->next = head;
        head = newnode;
        return;
    }
    while (1)
    {
        if (tmp->next->next == 0)
        {
            if (tmp->next->data >= newnode->data)
            {
                newnode->next = tmp->next;
                tmp->next = newnode;
            }
            else
            {
                tmp->next->next = newnode;
            }
            return;
        }
        if (tmp->next->data > newnode->data)
        {
            newnode->next = tmp->next;
            tmp->next = newnode;
            return;
        }
        tmp = tmp->next;
    }
}

void delete_pos(node *&head)
{
    node *tmp = head, *prv = head;
    int n;
    cin >> n;
    if (n == 1)
    {
        node *newnode = head;
        head = head->next;
        delete (newnode);
        return;
    }
    while (--n)
    {
        prv = tmp;
        tmp = tmp->next;
    }
    node *newnode = tmp;
    prv->next = tmp->next;
    delete (newnode);
}

void delete_number(node *&head)
{
    int n;
    cin >> n;
    while (n == head->data && head != 0)
    {
        node *newnode = head;
        head = head->next;
        delete (newnode);
    }
    node *tmp = head, *prev = head;
    while (tmp != 0)
    {
        if (tmp->data == n)
        {
            node *newnode = tmp;
            prev->next = tmp->next;
            delete (newnode);
        }
        if (tmp != 0)
            prev = tmp;
        tmp = tmp->next;
    }
}

void is_sorted(node *tmp)
{
    while (tmp->next != 0)
    {
        if (tmp->data > tmp->next->data)
        {
            cout << "Not Sorted" << endl;
            return;
        }
        tmp = tmp->next;
    }
    cout << "sorted" << endl;
}
```

```
void delete_duplicate(node *&head)
{
    node *q = head;
    while (q != 0)
    {
        int n;
        n = q->data;
        node *tmp = q->next, *prev = q;
        while (tmp != 0)
        {
            if (tmp->data == n)
            {
                node *newnode = tmp;
                prev->next = tmp->next;
                delete (newnode);
            }
            if (tmp != 0)
                prev = tmp;
            tmp = tmp->next;
        }
        q = q->next;
    }
}

void delete_duplicate2(node *&head)
{
    node *p = head;
    node *q = p->next;
    while (q != 0)
    {
        if (p->data != q->data)
        {
            p = q;
            q = q->next;
        }
        else
        {
            p->next = q->next;
            delete q;
            q = q->next;
        }
    }
}

void reverse1(node *&head)
{
    int length = len(head);
    int a[length];
    int i = 0;
    node *tm = head;
    while (tm != 0)
    {
        a[i] = tm->data;
        i++;
        tm = tm->next;
    }
    tm = head;
    i--;
    while (tm != 0)
    {
        tm->data = a[i];
        i--;
        tm = tm->next;
    }
}

void reverse(node *&head)
{
    if (head == 0 || head->next == 0)
    {
        return;
    }
    node *p = head;
    node *q = 0, *r = 0;
    while (p != 0)
    {
        r = q;
        q = p;
        p = p->next;
        q->next = r;
    }
    head = q;
}

void reversexx(node *&head)
{
    node *first = 0;
    node *second = head;
    node *third = 0;
    while (second != 0)
    {
        third = second->next;
        second->next = first;
        first = second;
        second = third;
    }
    head = first;
}

void reverse2(node *&head, node *p, node *q)
{
    if (p != 0)
    {
        reverse2(head, p->next, p);
        p->next = q;
    }
    else
    {
        head = q;
    }
}
================================================================================
#include <bits/stdc++.h>

using namespace std;

struct LL
{
    struct node
    {
        int data;
        node *next;
        node()
        {
            next = 0;
        }
    };
    node *head = NULL;

    void add_value(int n);

    void print();

    void concatation(node *b);

    void merge2(node *b);

    void isLoop();
};
void LL::merge2(node *b)
{
    node *first = head;
    node *second = b;
    node *mergedList = new node;
    node *last = mergedList;

    while (first != nullptr && second != nullptr)
    {
        if (first->data < second->data)
        {
            last->next = first;

            first = first->next;
        }
        else
        {
            last->next = second;
            second = second->next;
        }
        last = last->next;
    }

    if (first != nullptr)
    {
        last->next = first;
    }
    else if (second != nullptr)
    {
        last->next = second;
    }

    head = mergedList->next;

    delete mergedList;
}
void LL::isLoop()
{
    node *first = head;
    node *second = head;
    do
    {
        first = first->next;
        second = second->next;
        second = second ? second->next : second;
    } while (first && second && first != second);
    if (first == second)
    {
        cout << "LOOP" << endl;
    }
    else
    {
        cout << "not a loop" << endl;
    }
}
void LL::add_value(int n)
{
    node *newnode = new node;
    newnode->data = n;
    if (head == 0)
    {
        head = newnode;
    }
    else
    {
        node *tm = head;
        while (tm->next != 0)
        {
            tm = tm->next;
        }
        tm->next = newnode;
    }
}

void LL::print()
{
    node *tm = head;
    while (tm != 0)
    {
        cout << tm->data << " ";
        tm = tm->next;
    }
    cout << endl;
}
void LL::concatation(node *b)
{
    node *tm = head;
    while (tm->next != 0)
    {
        tm = tm->next;
    }
    tm->next = b;
}
==============================================================================
#include <bits/stdc++.h>

using namespace std;

struct Stacks
{
    struct Node
    {
        int data;
        Node *next;
    };
    Node *top = NULL;
    void push(int x);
    int peek(int pos);
    int pop();
};

void Stacks::push(int x)
{
    Node *newnode = new Node;
    newnode->data = x;
    if (top == NULL)
    {
        top = newnode;
        return;
    }
    newnode->next = top;
    top = newnode;
}

int Stacks::peek(int pos)
{
    if (top == 0)
    {
        return -1;
    }
    Node *tm = top;
    while (--pos)
    {
        tm = tm->next;
        if (tm == 0)
        {
            return -1;
        }
    }
    return tm->data;
}
int Stacks::pop()
{
    if (top == 0)
    {
        return -1;
    }
    int ans = top->data;
    Node *newnode = top;
    top = top->next;
    delete newnode;
    return ans;
}
============================================================
// Infix to postfix

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

struct Stack
{
    int top = -1;
    int a[1111];
};
void push(Stack &st, int number)
{
    st.top++;
    st.a[st.top] = number;
}

int pop(Stack &st)
{
    int x = st.a[st.top];
    st.top--;
    return x;
}

int main()
{
    string s;
    cin >> s;
    Stack st;
    for (int i = 0; s[i] != '\0'; i++)
    {
        if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/')
        {
            int x = pop(st);
            int y = pop(st);
            cout << x << " " << y << endl;
            if (s[i] == '+')
            {
                push(st, x + y);
            }
            else if (s[i] == ’-’)
            {
                push(st, x - y);
            }
            else if (s[i] == ’*’)
            {
                push(st, x * y);
            }
            else if (s[i] == ’/’)
            {
                if (x == 0)
                {
                    y += 2;
                }
                push(st, x / y);
            }
        }
        else if (s[i] == ’#’)
        {
            int x = pop(st);
            push(st, x + 1);
        }
        else if (s[i] == ’$’)
        {
            int x = pop(st);
            push(st, x - 1);
        }
        else
        {
            push(st, s[i] - ’0’);
        }
    }
    cout << st.a[st.top];
}

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

struct converters
{
    struct Stacks
    {
        int top = -1;
        int str[10];
    };
    Stacks st;
    void push(int number);
    int pop();
    int result(string s);
    int is_operator(char c)
    {
        if (c == ’+’ || c == ’-’ || c == ’*’ || c == ’/’ || c == ’^’)
        {
            return 1;
        }
        return 0;
    }
};

void converters::push(int number)
{
    st.top++;
    st.str[st.top] = number;
}

int converters::pop()
{
    int x = st.str[st.top];
    st.top--;
    return x;
}

int converters::result(string s)
{
    int i = 0;
    int ans = 0;
    while (s[i] != ’\0’)
    {
        if (is_operator(s[i]))
        {
            int x = pop();
            int y = pop();
            int z;
            if (s[i] == ’+’)
            {
                z = x + y;
            }
            if (s[i] == ’-’)
            {
                z = y - x;
            }
            if (s[i] == ’*’)
            {
                z = x * y;
            }
            if (s[i] == ’/’)
            {
                z = y / x;
            }
            if (s[i] == ’^’)
            {
                z = pow(y, x);
            }
            cout << z << endl;
            push(z);
        }
        else
        {
            push(s[i] - '0');
        }
        i++;
    }
    return st.str[st.top];
}

int main()
{
    converters cn;
    string s;
    cin >> s;
    cout << cn.result(s);
}
===============================================================
#include <bits/stdc++.h>
using namespace std;

struct Converter
{
    struct Stacks
    {
        int top = -1;
        int size;
        string str;
    };
    Stacks st;
    string transform(string s);
    int precedence(char a);
    int is_operand(char a);
    void push(char c);
    char peekTop();
    char pop();
};

void Converter::push(char c)
{

    st.top++;
    st.str[st.top] = c;
    return;
}
char Converter::peekTop()
{
    return st.str[st.top];
}

char Converter::pop()
{

    char x = st.str[st.top];
    st.top--;
    return x;
```

```
}

int Converter::is_operand(char a)
{
    if (a == ’+’ || a == ’-’ || a == ’*’ || a == ’/’ || a == ’^’)
    {
        return 0;
    }
    return 1;
}

string Converter::transform(string s)
{
    int length = s.length();
    char result[length + 2];
    int i = 0;
    int j = 0;
    while (s[i] != ’\0’)
    {
        if (s[i] == ’)’)
        {
            while (st.top != -1 && peekTop() != ’(’)
            {
                result[j++] = pop();
            }
            // Pop the ’(’ from the stack
            if (st.top != -1 && peekTop() == ’(’)
            {
                pop();
            }
            i++;
        }
        else if (s[i] == ’(’)
        {
            push(s[i++]);
        }
        else if (is_operand(s[i]))
        {

            result[j++] = s[i++];
        }
        else
        {
            if (precedence(s[i]) > precedence(peekTop()))
            {
                push(s[i++]);
            }
            else
            {
                result[j++] = pop();
            }
        }
    } // a+b*c-d/e
    while (st.top != -1)
    {
        char cc = pop();
        result[j] = cc;
        j++;
    }
    result[j] = '\0';
    cout << result;
    return result;
}
int Converter::precedence(char a)
{
    if (a == '-' || a == '+')
    {
        return 1;
    }
    else if (a == '*' || a == '/')
    {
        return 2;
    }
    else if (a == '^')
    {
        return 3;
    }
    return 0;
}

int main()
{
    string s;
    cin >> s;
    Converter *t = new Converter;
    string x = t->transform(s);
}
================================================
#include <bits/stdc++.h>

using namespace std;

void bubble(int a[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                int tm = a[j];
                a[j] = a[j + 1];
                a[j + 1] = tm;
            }
        }
    }
}

void selection(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int minpos = i;
        for (int j = i + 1; j < n; j++)
        {
            if (a[j] < a[minpos])
            {
                minpos = j;
            }
        }
        int tm = a[i];
        a[i] = a[minpos];
        a[minpos] = tm;
    }
}

void ins(int a[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i - 1;
        int x = a[i];
        while (j >= 0 && a[j] > x)
        {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = x;
    }
}

void merge(int a[], int start, int mid, int end)
{
    int i = start, j = mid + 1, k = 0;
    int *tm = new int[end - start + 1];
    while (i <= mid && j <= end)
    {
        if (a[i] < a[j])
        {
            tm[k++] = a[i++];
        }
        else
        {
            tm[k++] = a[j++];
        }
    }
    while (i <= mid)
    {
        tm[k++] = a[i++];
    }
    while (j <= end)
    {
        tm[k++] = a[j++];
    }
    for (i = 0; i < end - start + 1; i++)
    {
        a[i + start] = tm[i];
    }
}

void mergeSort(int a[], int start, int end)
{
    if (start < end)
    {
        int mid = start + (end - start) / 2;
        mergeSort(a, start, mid);
        mergeSort(a, mid + 1, end);
        merge(a, start, mid, end);
    }
}

int pivotting(int a[], int start, int end)
{
    int pivot = a[end];
    int i = start - 1;
    for (int j = start; j <= end; j++)
    {
        if (a[j] < pivot)
        {
            i++;
            int tm = a[j];
            a[j] = a[i];
            a[i] = tm;
        }
    }
    int tm = a[end];
    a[end] = a[i + 1];
    a[i + 1] = tm;
    return i + 1;
}

void quickSort(int a[], int start, int end)
{
    if (start < end)
    {
        int pivot = pivotting(a, start, end);
        quickSort(a, start, pivot - 1);
        quickSort(a, pivot + 1, end);
    }
}

int getMax(int a[], int n)
{
    int maxi = INT_MIN;
    for (int i = 0; i < n; i++)
    {
        if (a[i] > maxi)
        {
            maxi = a[i];
        }
    }
    return maxi;
}

int *countSort(int a[], int n)
{
    int max = getMax(a, n);
    int *b = new int[max + 1];
    for (int i = 0; i <= max; i++)
    {
        b[i] = 0;
    }
    for (int i = 0; i < n; i++)
    {
        b[a[i]]++;
    }
    return b;
}

int main()
{
    int a[] = {5, 4, 3, 2, 1};
    bubble(a, 5);
    for (int i = 0; i < 5; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
    int b[] = {5, 4, 3, 2, 1};
    ins(b, 5);
    for (int i = 0; i < 5; i++)
    {
        cout << b[i] << " ";
    }
    cout << endl;
    int c[] = {5, 4, 3, 2, 1};
    selection(c, 5);
    for (int i = 0; i < 5; i++)
    {
        cout << c[i] << " ";
    }
    cout << endl;
    int d[] = {-1, 4, 9, 2, 1};
    mergeSort(d, 0, 4);
    for (int i = 0; i < 5; i++)
    {
        cout << d[i] << " ";
    }
    cout << endl;
    int e[] = {-1, 4, 9, 2, 1};
    quickSort(e, 0, 4);
    for (int i = 0; i < 5; i++)
    {
        cout << e[i] << " ";
    }
    cout << endl;
    int f[] = {-1, 4, 9, 2, 1};
    quickSort(f,5);
    for (int i = 0; i < 5; i++)
    {
        cout << e[i] << " ";
    }
}
=============================================================
#include <bits/stdc++.h>

using namespace std;

void fun1(int n)
{
    //.................... ............................3 2 1--------------------------
    if (n <= 0)
    {
        return;
    }
    cout << n << " ";
    fun1(n - 1);
}

void fun2(int n)
{
    //------------------------------------1 2 3..................
    if (n <= 0)
    {
        return;
    }
    fun2(n - 1);
    cout << n << " ";
}

int sum_of_numbers(int n)
{
    if (n == 0)
    {
        return 0;
    }
    return sum_of_numbers(n - 1) + n;
}
int sum_of_numbers_using_staticAsGlobalVariables(int n)
{
    static int x = 0;
    if (n == 0)
    {
        return 0;
    }
    x++;
    return sum_of_numbers_using_staticAsGlobalVariables(n - 1) + x;
}
```

```
//0
//1 - 1
//2 - 2 1 1
//3 - 3 2 1 1 2 1 1
//4 - 4 3 2 1 1 2 1 1 3 2 1 1 2 1 1

void tree_recursion(int n)
{
    if (n == 0)
    {
        return;
    }
    cout << n << " ";
    tree_recursion(n - 1);
    tree_recursion(n - 1);
}

int nested_recursion(int n)
{
    if (n > 100)
    {
        return n - 10;
    }
    else
    {
        return nested_recursion(nested_recursion(n + 11));
    }
}

int sum_of_first_N_natural_numbers(int n)
{
    if (n == 0)
    {
        return 0;
    }
    return n + sum_of_first_N_natural_numbers(n - 1);
}

int factorial(int n)
{
    if (n <= 1)
    {
        return 1;
    }
    return n * factorial(n - 1);
}

int power_noob(int m, int n)
{
    if (n == 0)
    {
        return 1;
    }
    return m * power_noob(m, n - 1);
}
```

```
int power_pro(int m, int n)
{

    if (n == 0)
    {
        return 1;
    }
    int x = power_pro(m, n / 2);
    if (n % 2 == 0)
    {
        return x * x;
    }
    else
    {
        return x * x * m;
    }
}

int fibonacci(int n)
{
    if (n <= 2)
    {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

void fibonacci_print(int n)
{
    if (n <= 0)
    {
        return;
    }
    fibonacci_print(n - 1);
    cout << fibonacci(n) << " ";
}

int binomial_ex(int n, int r)
{
    long long int lob = factorial(n), hor = factorial(n - r), horr = factorial(r);
    return lob / (hor * horr);
}

int binomial_ex_pascal(int n, int r)
{
    if (r == 0 || n == r)
    {
        return 1;
    }
    return binomial_ex_pascal(n - 1, r - 1) + binomial_ex_pascal(n - 1, r);
}
void hanoi(int n, char frm, char to, char use)
{
    if (n > 0)
    {
        hanoi(n - 1, frm, use, to);
        cout << "Moved " << n << " From " << frm << " to " << to << endl;
        hanoi(n - 1, frm, to, use);
    }
}
```