1. Write a C++ program using stack ADT to balance the braces, parenthesis and brackets.

Header file:

```
#include <iostream>
#include <string>
using namespace std;
class AStack {
        int capacity;
        int top;
        char *arr;
    public:
        AStack();
        AStack(int size);
        ~AStack();
        void push(char ele);
        void pop();
        char peek();
        bool isEmpty();
};
bool isMatching(char open, char close);
bool isBalanced(string expr);
```

Implementation file:

```
#include "head.h";
AStack::AStack(){
    top = -1;
    capacity = 10;
    arr = new char[capacity];
}
```

```
AStack::AStack(int size){
        top=-1;
        capacity=size;
        arr=new char[capacity];
}
AStack::~AStack(){
        delete[] arr;
}
void AStack::push(char ele){
        if(top<capacity-1){
            top++;
            arr[top]=ele;
        }
        else{
            cout<<"Stack is full";
        }
}
void AStack::pop(){
        if(top!=-1){
            top--;
        }
        else{
            cout<<"Stack is empty";
        }
bool AStack::isEmpty(){
        return(top==-1);
}
char AStack::peek(){
        if(top!=-1){
            return arr[top];}
```

```
        else {
            cout << "Stack is empty";
        }

bool isMatching (char open, char close){
        if (open == '(' && close == ')') return true;
        if (open == '{' && close == '}') return true;
        if (open == '[' && close == ']') return true;
        return false;
}

bool isBalanced (string expr){
        AStack stack(expr.length());
        for (int i = 0; i < expr.length(); i++){
            char ch = expr[i];
            if (ch == '(' || ch == '{' || ch == '['){
                stack.push(ch);
            }
            else if (ch == ')' || ch = '}' || ch == ']') {
                if (stack.isEmpty() || isMatching (stack.peek(),
                                                   ch)){
                    return false;
                }
                stack.pop()
            }
        }
        return stack.isEmpty();
}
```

Main file:

```
import "header.h";
int main() {
String str;
cout << "Enter the expression:";
cin >> str;
if (isBalanced (str){
        cout << "Expression is balanced.";
  }
  else {
        cout << "Expression is not balanced.";
  }
}
```

2. Write a C++ program using Stack ADT to convert infix to postfix expression:

Program:

Header file:

```
#include<iostream>
#include<string>
using namespace std;
class AStack {
        int capacity;
        int top;
        char * arr;
        public:
          AStack();
          AStack(int size);
          ~AStack();
          void push();
```

```
        else {
            cout << "Stack is empty";
        }

bool isMatching (char open, char close){
        if (open == '(' && close == ')') return true;
        if (open == '{' && close == '}') return true;
        if (open == '[' && close == ']') return true;
        return false;
}

bool isBalanced (string expr){
        AStack stack(expr.length());
        for (int i = 0; i < expr.length(); i++){
            char ch = expr[i];
            if (ch == '(' || ch == '{' || ch == '['){
                stack.push(ch);
            }
            else if (ch == ')' || ch == '}' || ch == ']') {
                if (stack.isEmpty() || isMatching(stack.pop(),
                                                  ch)){
                    return false;
                }
                stack.pop()
            }
        }
        return stack.isEmpty();
}
```

```
        void push(char ele);
        char peek();
        bool isEmpty();
    };
int precedence(char op);
string infixToPostfix(string expn);
```

Implement file:

```
#include "head.h"
AStack::AStack(){
    top=-1;
    capacity=10;
    arr=new char[capacity];
}
AStack::AStack(int size){
    top=-1;
    capacity=size;
    arr=new char[capacity];}
AStack::~AStack(){
    delete[]arr;}
void AStack::push(char ele);
    if(top<capacity-1){
        top++;
        arr[top]=ele;
    }
    else{
        cout<<"Stack is full";}
}
void AStack::pop(){
    if(top!=-1){
        top--;}
    else{
        cout<<"Stack is empty";}
}
```

```
char AStack::peek(){
    if(top!=-1){
        return arr[top];}
    else{
        cout<<"Stack is empty";}
}
bool AStack::isEmpty(){
    return (top==-1);
}
int precedence(char op){
    if(op=='+'||op=='-'){
        return 1;
    }
    if(op=='*'||op=='/'){
        return 2;}
    return 0;
}
string infixtoPostfix(string exp){
    AStack stack(exp.length());
    string p="";
    for(int i=0;i<exp.length();i++){
        char ch=exp[i];
        if(isalnum(ch)){
            p+=ch;}
        else if(ch=='('){
            stack.push(ch);}
        else if(ch==')'){
            while(!stack.isEmpty()&&stack.peek()!='('){
                p+=stack.pop();
            }
            stack.pop();
```

```
        else {
            while(!stack.isEmpty() && precedence(stack.peek()
                                    >= precedence(ch)){
                p += stack.pop();
            }
            stack.push(ch);
        }
    }

    while (!stack.isEmpty()){
        p += stack.pop();
    }
    return p;
}
```

Main file:

```
#include "head.h";
int main(){
    string exp;
    cout << "Enter an infix expression: ";
    cin >> exp;
    string p = infixToPostfix(exp);
    cout << "Postfix expression: " << p << endl;
    return 0;
}
```

Output:

Enter an infix expression: a+b*g-k%.n+m

Postfix expression: abg*+k-nm+%.

Output for balance Stack ADt program:

Testcase 1:

Enter the expression: {[2](

Expression is not balanced.

Testcase 2:

Enter the expression: {[]}

Expression is balanced.

3. Write a C++ program using List ADT - Array implementation to solve this for general values of M and N. What is the running time of the program?

Header file:

```
#include <iostream>;
using namespace std;
class List{
    int* elements;
    int size;
    int currentSize;
    public:
        List();
        List(int size);
        ~List();
        void eliminate(int index);
        int get(int index);
        int getsize();
};
```

Implementation file:

```
#include "head.h"

List() :: List() {
        size = 0;
        capacity = 15;
        arr = new int[capacity]; }

List() :: List(int c, int s, int *a){
        capacity = c;
        size = s;
        arr = new int[capacity];
        for(int i = 0; i < size; i++){
                arr[i] = a[i];
        }
}

List() :: ~List(){
        capacity = 0;
        size = 0;
        delete[] arr; }

void List() :: eliminate(int index){
        for(int i = index; i < currentsize - 1; i++){
                arr[i] = arr[i+1];
        }
        currentsize--; }

int List :: get(index){
        return arr[index]; }

int
        :: getsize(){
        return currentsize; }
```

Main file :

```
#include "head.h";
int main(){
    int N, M;
    int capacity;
    cout << " Enter the number of people (N): ";
    cin >> N;
    cout << " Enter the number of passes (M): ";
    cin >> M;

    int currentIndex = 0;
    while (people.getsize() > 1) {
        currentIndex = (currentIndex + N-1) %
                        people.getSize();
        people.eliminate (currentIndex);
    }
    cout << " The winner is person : " << people.get(0)
         << endl;
    return 0;
}
```

Output :

Testcase 1 :

Enter the number of people (N) : 89

Enter the number of passes (M) : 70

The winner is person : 23

Testcase 2 :

Enter the number of people (N) : 60

Enter the number of passes (M) : 50

The winner is person : 51