**Question 1:**

***Palindrome Check Stack (Arrays);***

*void* palindrome\_check(string *expression*){

            size = *expression*.length();

            stack\_array = new *char*[size];

*// Reverse String*

            string reverse = "";

            for(*int* i = 0; i < size; i++){

*// push-ing elements into the stack*

                push(*expression*[i]);

            }

            for(*int* i = size; i > 0; i--){

*char* x = stack\_array[top];

                pop();

                reverse += x;

            }

            cout << reverse << endl;

            if(reverse == *expression*){

                cout << "Palindrome" << endl;

            }

            else{

                cout << "Not Palindrome" << endl;

            }

            top = -1;

        }

***Palindrome Check Stack (Singly/Doubly Linked List);***

*void* palindromeCheck(){

    string forward;

    string backward;

*// inserting in forward direction*

    while(!isEmpty()){

        forward += pop();

    }

*// inserting in backward direction*

    for (*int* i = forward.length() - 1; i >= 0; i--)

    {

        backward += forward[i];

    }

*// comparing them together!*

    if (forward != backward)

    {

        cout << "Not Palindrome" << endl;

    }

    else{

        cout << "Palindrome" << endl;

    }

}

**Question 2/3:**

***Balanced Parenthesis Check()***

*void* balanced\_check(string *expression*){

            size = *expression*.length();

            stack\_array = new *char*[size];

*// Boolean for checking*

*bool* check = true;

            for(*int* i = 0; i < size; i++){

                if(*expression*[0] == ')' || *expression*[0] == '}' || *expression*[0] == ']'){

                    check = false;

                    break;

                }

                else if(is\_Empty()){

                    push(*expression*[i]);

                }

                else if(*expression*[i] == ')' && stack\_array[top] == '('){

                    pop();

                }

                else if(*expression*[i] == '}' && stack\_array[top] == '{'){

                    pop();

                }

                else if(*expression*[i] == ']' && stack\_array[top] == '['){

                    pop();

                }

                else if(*expression*[i] == ')' && stack\_array[top] != '('){

                    check = false;

                    break;

                }

                else if(*expression*[i] == '}' && stack\_array[top] != '{'){

                    check = false;

                    break;

                }

                else if(*expression*[i] == ']' && stack\_array[top] != '['){

                    check = false;

                    break;

                }

                else if (prec(stack\_array[top]) >= prec(*expression*[i] )){

                    push(*expression*[i]);

                }

            }

            if(check == true && top == -1){

                cout << "Your expression is balanced" << endl;

            }

            else{

                cout << "Your expression is un-balanced" << endl;

            }

            top = -1;

        }

**Question 4:**

***Infix to Postfix();***

string infix\_to\_postfix(string *expression*){

            size = *expression*.length();

            stack\_array = new *char*[size];

*// final expression string*

            string string\_evaluation = "";

            for(*int* i = 0; i < size + 1; i++){

                if(*expression*[0] == ')' || *expression*[0] == '}' || *expression*[0] == ']'){

                    cout << "Your expression is wrong! Try adding the right one!" << endl;

                    break;

                }

                else if((*expression*[i] >= 'a' && *expression*[i] <= 'z') || (*expression*[i] >= 'A' && *expression*[i] <= 'Z') || (*expression*[i] >= '0' && *expression*[i] <= '9')){

                    string\_evaluation += *expression*[i];

                }

                else if(is\_Empty() && (*expression*[i] == '+' || *expression*[i] == '-' || *expression*[i] == '\*' || *expression*[i] == '/' || *expression*[i] == '^' || *expression*[i] == '(' || *expression*[i] == '{' || *expression*[i] == '[')){

                    push(*expression*[i]);

*//cout << "\nExpression: " << expression[i];*

                }

                else if(*expression*[i] == '(' || *expression*[i] == '{' || *expression*[i] == '['){

                    push(*expression*[i]);

*//cout << "\nExpression: " << expression[i];*

                }

                else if(prec(*expression*[i]) > prec(stack\_array[top])){

                    push(*expression*[i]);

*//cout << "\nExpression: " << expression[i];*

                }

                else if(*expression*[i] == ')'){

                    for(*int* j = top; j >= 0; j--){

                        if(stack\_array[j] == '('){

                            pop();

                            break;

                        }

                        string\_evaluation += stack\_array[j];

                        pop();

                    }

                }

*// inserting greater priority operator*

                else if(prec(*expression*[i]) <= prec(stack\_array[top]) && (*expression*[i] != '(' || *expression*[i] != '{' || *expression*[i] != '[' || *expression*[i] != ')' || *expression*[i] != '}'|| *expression*[i] != ']')){

*// loop going backwards till it finds an expression of greater priority!*

                    while (!is\_Empty() && (prec(stack\_array[top]) >= prec(*expression*[i])))

                    {

                        string\_evaluation += stack\_array[top];

                        pop();

                    }

                    push(*expression*[i]);

*// cout << "\nExpression loop: " << expression[i];*

*// cout << "\nTop is: " << top << endl;*

                }

            }

*//cout << "Top value is: " << top << endl;*

*//string\_evaluation += stack\_array[top];*

            while(is\_Empty()){

                string\_evaluation += stack\_array[top];

*//cout << "Element is: " << stack\_array[top] << endl;*

                pop();

            }

            cout << "Converted expression is: " << string\_evaluation << endl;

            top = -1;

            return string\_evaluation;

        }

**Question 5:**

***Postfix evaluation()***

*void* postfixEvaluation(string *expression*){

*// sum numbers*

*int* num1 = 0;

*int* num2 = 0;

*// Size allocation*

            size = *expression*.length();

            stack\_array = new *char*[size];

            string new\_expression = infix\_to\_postfix(*expression*);

            for(*int* i = 0; i < new\_expression.length() + 1; i++){

                if(new\_expression[i] >= '0' && new\_expression[i] <= '9'){

                    push((*int*) new\_expression[i] - 48);

                }

                else if(new\_expression[i] == '+'){

                    num1 = (*int*) (stack\_array[top--]);

                    num2 = (*int*) (stack\_array[top--]);

                    if(num2 + num1 != 0){

                        push((num2 + num1));

                    }

                    else{

                        cout << endl;

                    }

                }

                else if(new\_expression[i] == '-'){

                    num1 = (*int*) (stack\_array[top--]);

                    num2 = (*int*) (stack\_array[top--]);

                    if(num2 - num1 != 0){

                        push((num2 - num1));

                    }

                    else{

                        cout << endl;

                    }

                }

                else if(new\_expression[i] == '\*'){

                    num1 = (*int*) (stack\_array[top--]);

                    num2 = (*int*) (stack\_array[top--]);

                    if(num2 \* num1 != 0){

                        push((num2 \* num1));

                    }

                    else{

                        cout << endl;

                    }

                }

                else if(new\_expression[i] == '/'){

                    num1 = (*int*) (stack\_array[top--]);

                    num2 = (*int*) (stack\_array[top--]);

                    if(num2 / num1 > 0){

                        push((num2 / num1));

                    }

                    else{

                        cout << endl;

                    }

                }

                else if(new\_expression[i] == '^'){

                    num1 = (*int*) (stack\_array[top--]);

                    num2 = (*int*) (stack\_array[top--]);

                    if(num2 ^ num1 != 0){

                        push((num2 ^ num1));

                    }

                    else{

                        cout << endl;

                    }

                }

            }

            if(!is\_Empty()){

*int* final\_answer = (*int*) stack\_array[top];

                cout << "\nEvaluated Answer is: " << final\_answer;

            }

            else{

                cout << "ZERO!" << endl;

            }

        }

**Question 6:**

***Pizza Queue***

*class* queue{

*private:*

*int* front;

*int* back;

*int* size;

*int* \*array;

*public:*

        queue(){

            size = 5;

            array = new *int*[size];

            front = -1;

            back = -1;

        }

*bool* isEmpty(){

            if(front == -1 && back == -1){

                return true;

            }

            else{

                return false;

            }

        }

*bool* isFull(){

            if(back == size - 1){

                return true;

            }

            else{

                return false;

            }

        }

*int* enqueue(*int* *number*){

            if(isFull()){

                cout << "Queue is full! Try dequeing it";

            }

            else if(isEmpty()){

                front++;

                back++;

                array[back] = *number*;

                return *number*;

            }

            else{

                back++;

                array[back] = *number*;

                return *number*;

            }

            return 0;

        }

*void* printLinearArray(){

            for (*int* i = 0; i < back + 1; i++) {

                cout << array[i] << "  ";

            }

        }

*void* Front(){

*//cout << "Front is: " <<endl;*

            cout << front;

        }

*void* Back(){

*//cout << "Back is: " <<endl;*

            cout << back;

        }

*void* left\_shift(){

            for(*int* i = front; i < back + 1; i++){

                array[i - 1] = array[i];

            }

        }

*int* dequeue(){

            if(isEmpty()){

                cout << "Empty! You cannot dequeue any more elements" << endl;

            }

            else if(front == back){

                front = back = -1;

            }

            else{

                left\_shift();

                back--;

            }

        return 0;

        }

};

**Question 7:**

***Pizza Queue reverse Orders***

*class* queue{

*private:*

*int* front;

*int* back;

*int* size;

*int* \*array;

*public:*

        queue(){

            size = 5;

            array = new *int*[size];

            front = -1;

            back = -1;

        }

*bool* isEmpty(){

            if(front == -1 && back == -1){

                return true;

            }

            else{

                return false;

            }

        }

*bool* isFull(){

            if(back == size - 1){

                return true;

            }

            else{

                return false;

            }

        }

*int* enqueue(*int* *value*){

            if(isEmpty()){

                front = 0;

                back = 0;

                array[front] = *value*;

                return *value*;

            }

            else if(isFull()){

                cout << "You cannot insert more items! Full" << endl;

                return 0;

            }

            else if (front == 0 && back != size - 1){

                right\_shift();

                array[front] = *value*;

                back++;

                return *value*;

            }

            else{

                front--;

                array[front] = *value*;

                return *value*;

            }

        }

*void* printLinearArray(){

            for (*int* i = 0; i < back + 1; i++) {

                cout << array[i] << "  ";

            }

        }

*void* Front(){

*//cout << "Front is: " <<endl;*

            cout << front;

        }

*void* Back(){

*//cout << "Back is: " <<endl;*

            cout << back;

        }

*void* left\_shift(){

            for(*int* i = front; i < back + 1; i++){

                array[i - 1] = array[i];

            }

        }

*void* dequeue(){

            if(isEmpty()){

                cout << "You cannot delete more values";

            }

            else if(front == back){

                front = -1;

                back = -1;

            }

            else{

                back--;

            }

        }

};

**Question 8:**

***Double Ended Pizza Queue***

*class* queue{

*private:*

*int* front;

*int* back;

*int* size;

*int* \*array;

*public:*

        queue(){

            size = 5;

            array = new *int*[size];

            front = -1;

            back = -1;

        }

*bool* isEmpty(){

            if(front == -1 && back == -1){

                return true;

            }

            else{

                return false;

            }

        }

*bool* isFull(){

            if(back == size - 1){

                return true;

            }

            else{

                return false;

            }

        }

*void* printLinearArray(){

            for (*int* i = 0; i < back + 1; i++) {

                cout << array[i] << "  ";

            }

        }

*void* Front(){

*//cout << "Front is: " <<endl;*

            cout << front;

        }

*void* Back(){

*//cout << "Back is: " <<endl;*

            cout << back;

        }

*void* left\_shift(){

            for(*int* i = front; i < back + 1; i++){

                array[i - 1] = array[i];

            }

        }

*void* right\_shift(){

            for(*int* i = back + 1; i > front; i--){

                array[i] = array[i - 1];

            }

        }

*int* enqueue\_from\_end(*int* *number*){

            if(isFull()){

                cout << "Queue is full! Try dequeing it";

            }

            else if(isEmpty()){

                front++;

                back++;

                array[back] = *number*;

                return *number*;

            }

            else{

                back++;

                array[back] = *number*;

                return *number*;

            }

            return 0;

        }

*void* dequeue\_from\_start(){

            if(isEmpty()){

                cout << "Empty! You cannot dequeue any more elements" << endl;

            }

            else if(front == back){

                front = back = -1;

            }

            else{

                left\_shift();

                back--;

            }

        }

*int* enqueue\_from\_start(*int* *value*){

            if(isEmpty()){

                front = 0;

                back = 0;

                array[front] = *value*;

                return *value*;

            }

            else if(isFull()){

                cout << "You cannot insert more items! Full" << endl;

                return 0;

            }

            else if (front == 0 && back != size - 1){

                right\_shift();

                array[front] = *value*;

                back++;

                return *value*;

            }

            else{

                front--;

                array[front] = *value*;

                return *value*;

            }

        }

*void* dequeue\_from\_end(){

            if(isEmpty()){

                cout << "You cannot delete more values";

            }

            else if(front == back){

                front = -1;

                back = -1;

            }

            else{

                back--;

            }

        }

­