Student: Yerbolat Pazyl

Group: IS-1808K

**Laboratory work #4.**

Solve these problems using stack, queue, deque data structures.

Deadline: week 5

**Problem - 1**

<https://informatics.msk.ru/mod/statements/view.php?id=207#1>

#### **Задача №54. Простой стек**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "стек". Напишите программу, содержащую описание стека и моделирующую работу стека, реализовав все указанные здесь методы.  Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push n**

Добавить в стек число n (значение n задается после команды). Программа должна вывести ok.

**pop**

Удалить из стека последний элемент. Программа должна вывести его значение.

**back**

Программа должна вывести значение последнего элемента, не удаляя его из стека.

**size**

Программа должна вывести количество элементов в стеке.

**clear**

Программа должна очистить стек и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

**Входные данные**

Команды управления стеком вводятся в описанном ранее формате по 1 на строке.

Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в стеке в любой момент не превосходит 100, все команды pop и back корректны, то есть при их исполнении в стеке содержится хотя бы один элемент.

**Выходные данные**

Требуется вывести протокол работы со стеком, по 1 сообщению в строке

**Solution**

Creating a strict with the fields sz (store info size of stack) and the array a[1000]. In the constructor of a struct that sz = 0. Determined by methods such as the push, pop, back, size, clear, exit. In function push do a[sz++] = x.

In function pop do: cout << a[sz - 1] << endl;

sz--;

**C++ Code**

#include <bits/stdc++.h>

using namespace std;

struct Stack {

int sz, a[1000];

Stack() {

sz = 0;

}

void push(int x) {

a[sz++] = x;

cout << "ok" << endl;

}

void back() {

cout << a[sz - 1] << endl;

}

void pop() {

cout << a[sz - 1] << endl;

sz--;

}

void size() {

cout << sz << endl;

}

void clear() {

sz = 0;

cout << "ok" << endl;

}

void exit() {

cout << "bye" << endl;

}

};

int main()

{

Stack st;

int x;

string s;

while (true) {

cin >> s;

if ("push" == s) {

cin >> x;

st.push(x);

} else if ("pop" == s)

st.pop();

else if ("back" == s)

st.back();

else if ("size" == s)

st.size();

else if ("clear" == s)

st.clear();

else {

st.exit();

return 0;

}

}

return 0;

}

**Problem - 2**

<https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=55#1>

#### **Задача №55. Стек с защитой от ошибок**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "стек". Напишите программу, содержащую описание стека и моделирующую работу стека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push n**

Добавить в стек число n (значение n задается после команды). Программа должна вывести ok.

**pop**

Удалить из стека последний элемент. Программа должна вывести его значение.

**back**

Программа должна вывести значение последнего элемента, не удаляя его из стека.

**size**

Программа должна вывести количество элементов в стеке.

**clear**

Программа должна очистить стек и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

Перед исполнением операций back и pop программа должна проверять, содержится ли в стеке хотя бы один элемент. Если во входных данных встречается операция back или pop, и при этом стек пуст, то программа должна вместо числового значения вывести строку error.

**Входные данные**

Вводятся команды управления стеком, по одной на строке

**Выходные данные**

Программа должна вывести протокол работы стека, по одному сообщению на строке

**Примеры**

**входные данные**

size

push 1

size

push 2

size

push 3

size

exit

**выходные данные**

0

ok

1

ok

2

ok

3

bye

**Solution**

**C++ Code**

#include <bits/stdc++.h>

using namespace std;

struct Stack {

int sz, a[10000];

Stack() {

sz = 0;

}

void push(int x) {

a[sz++] = x;

cout << "ok" << endl;

}

void back() {

if (sz == 0)

cout << "error" << endl;

else

cout << a[sz - 1] << endl;

}

void pop() {

if (sz == 0)

cout << "error" << endl;

else {

cout << a[sz - 1] << endl;

sz--;

}

}

void size() {

cout << sz << endl;

}

void clear() {

sz = 0;

cout << "ok" << endl;

}

void exit() {

cout << "bye" << endl;

}

};

int main()

{

Stack st;

int x;

string s;

while (true) {

cin >> s;

if ("push" == s) {

cin >> x;

st.push(x);

} else if ("pop" == s)

st.pop();

else if ("back" == s)

st.back();

else if ("size" == s)

st.size();

else if ("clear" == s)

st.clear();

else {

st.exit();

return 0;

}

}

return 0;

}

**Problem - 3**

<https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=57#1>

#### **Задача №57. Простая очередь**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "очередь". Напишите программу, содержащую описание очереди и моделирующую работу очереди, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push n**

Добавить в очередь число n (значение n задается после команды). Программа должна вывести ok.

**pop**

Удалить из очереди первый элемент. Программа должна вывести его значение.

**front**

Программа должна вывести значение первого элемента, не удаляя его из очереди.

**size**

Программа должна вывести количество элементов в очереди.

**clear**

Программа должна очистить очередь и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в очереди в любой момент не превосходит 100, все команды pop и front корректны, то есть при их исполнении в очереди содержится хотя бы один элемент.

**Входные данные**

Вводятся команды управления очередью, по одной на строке

**Выходные данные**

Требуется вывести протокол работы с очередью, по одному сообщению на строке

**Примеры**

**входные данные**

size

push 1

size

push 2

size

push 3

size

exit

**выходные данные**

0

ok

1

ok

2

ok

3

bye

**Solution**

**JAVA Code**

import java.util.Scanner;

import java.util.ArrayList;

public class MyClass {

public static void main(String args[]) {

int x;

String s;

MyQueue queue = new MyQueue();

Scanner sc = new Scanner(System.in);

while(true) {

s = sc.next();

if (s.equals("push")) {

x = sc.nextInt();

queue.push(x);

} else if (s.equals("pop"))

queue.pop();

else if (s.equals("front"))

queue.front();

else if (s.equals("size"))

queue.size();

else if (s.equals("clear"))

queue.clear();

else {

queue.exit();

return;

}

}

}

static class MyQueue {

ArrayList<Integer> queue = new ArrayList<>();

void push(int x) {

queue.add(x);

System.out.println("ok");

}

void pop() {

queue.remove(0);

System.out.println("ok");

}

void front() {

System.out.println(queue.get(0));

}

void size() {

System.out.println(queue.size());

}

void clear() {

queue.clear();

System.out.println("ok");

}

void exit() {

System.out.println("bye");

}

}

}

**Problem - 4**

<https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=58#1>

#### **Задача №58. Очередь с защитой от ошибок**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "очередь". Напишите программу, содержащую описание очереди и моделирующую работу очереди, реализовав все указанные здесь методы.  Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push n**

Добавить в очередь число n (значение n задается после команды). Программа должна вывести ok.

**pop**

Удалить из очереди первый элемент. Программа должна вывести его значение.

**front**

Программа должна вывести значение первого элемента, не удаляя его из очереди.

**size**

Программа должна вывести количество элементов в очереди.

**clear**

Программа должна очистить очередь и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

Перед исполнением операций front и pop программа должна проверять, содержится ли в очереди хотя бы один элемент. Если во входных данных встречается операция front или pop, и при этом очередь пуста, то программа должна вместо числового значения вывести строку error.

**Входные данные**

Вводятся команды управления очередью, по одной на строке

**Выходные данные**

Требуется вывести протокол работы очереди, по одному сообщению на строке

**Примеры**

**входные данные**

push 1

front

exit

**выходные данные**

ok

1

bye

**Solution**

**Java Code**

import java.util.Scanner;

import java.util.ArrayList;

public class Task9 {

public static void main(String args[]) {

int x;

String s;

MyQueue queue = new MyQueue();

Scanner sc = new Scanner(System.in);

while(true) {

s = sc.next();

if (s.equals("push")) {

x = sc.nextInt();

queue.push(x);

} else if (s.equals("pop"))

queue.pop();

else if (s.equals("front"))

queue.front();

else if (s.equals("size"))

queue.size();

else if (s.equals("clear"))

queue.clear();

else {

queue.exit();

return;

}

}

}

static class MyQueue {

ArrayList<Integer> queue = new ArrayList<>();

void push(int x) {

queue.add(x);

System.out.println("ok");

}

void pop() {

if(queue.isEmpty()) {

System.out.println("error");

} else {

int a = queue.get(0);

queue.remove(0);

System.out.println(a);

}

}

void front() {

if (queue.isEmpty())

System.out.println("error");

else

System.out.println(queue.get(0));

}

void size() {

System.out.println(queue.size());

}

void clear() {

queue.clear();

System.out.println("ok");

}

void exit() {

System.out.println("bye");

}

}

}

**Problem - 5**

<https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=60#1>

#### **Задача №60. Простой дек**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "дек".  Напишите программу, содержащую описание дека и моделирующую работу дека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push\_front**

Добавить (положить) в начало дека новый элемент. Программа должна вывести ok.

**push\_back**

Добавить (положить) в конец дека новый элемент. Программа должна вывести ok.

**pop\_front**

Извлечь из дека первый элемент. Программа должна вывести его значение.

**pop\_back**

Извлечь из дека последний элемент. Программа должна вывести его значение.

**front**

Узнать значение первого элемента (не удаляя его). Программа должна вывести его значение.

**back**

Узнать значение последнего элемента (не удаляя его). Программа должна вывести его значение.

**size**

Вывести количество элементов в деке.

**clear**

Очистить дек (удалить из него все элементы) и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

Гарантируется, что количество элементов в деке в любой момент не превосходит 100. Все операции pop\_front, pop\_back, front, back всегда корректны.

**Входные данные**

Вводятся команды управления деком, по одной на строке.

**Выходные данные**

Требуется вывести протокол работы дека, по одному сообщению на строке.

**Примеры**

**входные данные**

push\_back 1

back

exit

**выходные данные**

ok

1

bye

**Solution**

**Java Code**

import java.util.Scanner;

import java.util.ArrayList;

public class Task9 {

public static void main(String args[]) {

int x;

String s;

MyDequeque dequeue = new MyDequeque();

Scanner sc = new Scanner(System.in);

while(true) {

s = sc.next();

if (s.equals("push\_front")) {

x = sc.nextInt();

dequeue.push\_front(x);

} else if (s.equals("push\_back")) {

x = sc.nextInt();

dequeue.push\_back(x);

}

else if (s.equals("pop\_front"))

dequeue.pop\_front();

else if (s.equals("pop\_back"))

dequeue.pop\_back();

else if (s.equals("front"))

dequeue.front();

else if (s.equals("back"))

dequeue.back();

else if (s.equals("size"))

dequeue.size();

else if (s.equals("clear"))

dequeue.clear();

else {

dequeue.exit();

return;

}

}

}

static class MyDequeque {

ArrayList<Integer> dequeue = new ArrayList<>();

void push\_front(int x) {

dequeue.add(0, x);

System.out.println("ok");

}

void push\_back(int x) {

dequeue.add(x);

System.out.println("ok");

}

void pop\_back() {

int x = dequeue.get(dequeue.size() - 1);

dequeue.remove(dequeue.size() - 1);

System.out.println(x);

}

void pop\_front() {

int x = dequeue.get(0);

dequeue.remove(0);

System.out.println(x);

}

void front() {

System.out.println(dequeue.get(0));

}

void back() {

System.out.println(dequeue.get(dequeue.size() - 1));

}

void size() {

System.out.println(dequeue.size());

}

void clear() {

dequeue.clear();

System.out.println("ok");

}

void exit() {

System.out.println("bye");

}

}

}

**Problem - 6**

<https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=61#1>

#### **Задача №61. Дек с защитой от ошибок**

Количество элементов во всех структурах данных не превышает 10000, если это не указано особо.

Реализуйте структуру данных "дек".  Напишите программу, содержащую описание дека и моделирующую работу дека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

**push\_front**

Добавить (положить) в начало дека новый элемент. Программа должна вывести ok.

**push\_back**

Добавить (положить) в конец дека новый элемент. Программа должна вывести ok.

**pop\_front**

Извлечь из дека первый элемент. Программа должна вывести его значение.

**pop\_back**

Извлечь из дека последний элемент. Программа должна вывести его значение.

**front**

Узнать значение первого элемента (не удаляя его). Программа должна вывести его значение.

**back**

Узнать значение последнего элемента (не удаляя его). Программа должна вывести его значение.

**size**

Вывести количество элементов в деке.

**clear**

Очистить дек (удалить из него все элементы) и вывести ok.

**exit**

Программа должна вывести bye и завершить работу.

Гарантируется, что количество элементов в деке в любой момент не превосходит 100. Перед исполнением операций pop\_front, pop\_back, front, back программа должна проверять, содержится ли в деке хотя бы один элемент. Если во входных данных встречается операция pop\_front, pop\_back, front, back, и при этом дек пуст, то программа должна вместо числового значения вывести строку error.

**Входные данные**

Вводятся команды управления деком, по одной на строке.

**Выходные данные**

Требуется вывести протокол работы дека, по одному сообщению на строке

**Примеры**

**входные данные**

push\_back 1

back

exit

**выходные данные**

ok

1

bye

**Solution**

**Java Code**

import java.util.Scanner;

import java.util.ArrayList;

public class Task9 {

public static void main(String args[]) {

int x;

String s;

MyDequeque dequeue = new MyDequeque();

Scanner sc = new Scanner(System.in);

while(true) {

s = sc.next();

if (s.equals("push\_front")) {

x = sc.nextInt();

dequeue.push\_front(x);

} else if (s.equals("push\_back")) {

x = sc.nextInt();

dequeue.push\_back(x);

}

else if (s.equals("pop\_front"))

dequeue.pop\_front();

else if (s.equals("pop\_back"))

dequeue.pop\_back();

else if (s.equals("front"))

dequeue.front();

else if (s.equals("back"))

dequeue.back();

else if (s.equals("size"))

dequeue.size();

else if (s.equals("clear"))

dequeue.clear();

else {

dequeue.exit();

return;

}

}

}

static class MyDequeque {

ArrayList<Integer> dequeue = new ArrayList<>();

void push\_front(int x) {

dequeue.add(0, x);

System.out.println("ok");

}

void push\_back(int x) {

dequeue.add(x);

System.out.println("ok");

}

void pop\_back() {

if(dequeue.isEmpty())

System.out.println("error");

else {

int x = dequeue.get(dequeue.size() - 1);

dequeue.remove(dequeue.size() - 1);

System.out.println(x);

}

}

void pop\_front() {

if (dequeue.isEmpty())

System.out.println("error");

else {

int x = dequeue.get(0);

dequeue.remove(0);

System.out.println(x);

}

}

void front() {

if (dequeue.isEmpty())

System.out.println("error");

else

System.out.println(dequeue.get(0));

}

void back() {

if (dequeue.isEmpty())

System.out.println("error");

else

System.out.println(dequeue.get(dequeue.size() - 1));

}

void size() {

System.out.println(dequeue.size());

}

void clear() {

dequeue.clear();

System.out.println("ok");

}

void exit() {

System.out.println("bye");

}

}

}

**Problem - 7**

<https://informatics.msk.ru/mod/statements/view.php?id=206#1>

#### **Задача №49. Списки по классам**

***Формат входных данных***

В каждой строке сначала записан номер класса (число, равное 9, 10 или 11), затем (через пробел) – фамилия ученика. Общее число строк в файле не превосходит 100000. Длина каждой фамилии не превосходит 50 символов.

***Формат выходных данных***

Необходимо вывести список школьников по классам: сначала всех учеников 9 класса, затем – 10, затем – 11. Внутри одного класса порядок вывода фамилий должен быть таким же, как на входе.

***Пример***

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| 9 Иванов 10 Петров 11 Сидоров 9 Григорьев 9 Сергеев 10 Яковлев | 9 Иванов 9 Григорьев 9 Сергеев 10 Петров 10 Яковлев 11 Сидоров |

**Solution**

Сreating an array with the queue type(q[3]). every time we get a student, we put it in a certain index of the array. (classNumber - 9). Index 0 stores a list of 9 classes. Index 1 stores a list of 10 classes. Index 2 stores a list of 11 classes. Then use a loop to iterate through the queue.

**C++ Code**

#include <bits/stdc++.h>

using namespace std;

int main()

{

queue < string > q[3];

int c;

string s;

while (cin >> c >> s) {

q[c - 9].push(s);

}

for (int i = 9; i <= 11; i++) {

while(!q[i].empty()) {

cout << i << " " << q[i - 9].front() << endl;

q[i - 9].pop();

}

}

return 0;

}

**Problem - 8**

<https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=50#1>

#### **Задача №50. Игра в пьяницу**

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

**Входные данные**

Программа получает на вход две строки: первая строка содержит 5 чисел, разделенных пробелами — номера карт первого игрока, вторая – аналогично 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

**Выходные данные**

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.

**Примеры**

**входные данные**

1 3 5 7 9

2 4 6 8 0

**выходные данные**

second 5

**Solution**

Сreate two queue the first to store cards of the first player, the second of the second player. Opening the while loop with the condition (k < 1000000 && !player1.isEmpty() && !player 2.isEmpty()). Inside the loop, we check for three conditions:

1. player1.peek() == 0 && player2.peek() == 9

2. player1.peek() == 9 && player2.peek() == 0

3. player1.peek() > player2.peek()

ELSE player1.peek() < player2.peek()

If the first player has more then we put the first player's card in his deck first then the second player's card.

**Java Code**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class Task8 {

public static void main(String[] args) {

Queue<Integer> player1 = new LinkedList<>();

Queue<Integer> player2 = new LinkedList<>();

Scanner sc = new Scanner(System.in);

for (int i = 0; i < 5; i++) {

player1.offer(sc.nextInt());

}

for (int i = 0; i < 5; i++) {

player2.offer(sc.nextInt());

}

int k = 0;

while (k < 1000000 && !player1.isEmpty() && !player2.isEmpty()) {

if (player1.peek() == 0 && player2.peek() == 9) {

player1.offer(player1.poll());

player1.offer(player2.poll());

} else if (player1.peek() == 9 && player2.peek() == 0) {

player2.offer(player1.poll());

player2.offer(player2.poll());

} else if (player1.peek() > player2.peek()) {

player1.offer(player1.poll());

player1.offer(player2.poll());

} else {

player2.offer(player1.poll());

player2.offer(player2.poll());

}

++k;

}

String winner = "botva";

if (k != 1000000)

winner = (player1.isEmpty()) ? ("second " + k) : ("first " + k);

System.out.println(winner);

}

}

**Problem - 9**

<https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=51#1>

#### **Задача №51. Правильная скобочная последовательность**

Рассмотрим последовательность, состоящую из круглых, квадратных и фигурных скобок. Программа дожна определить, является ли данная скобочная последовательность правильной.

Пустая последовательность явлется правильной. Если A – правильная, то последовательности (A), [A], {A} – правильные. Если A и B – правильные последовательности, то последовательность AB – правильная.

**Входные данные**

В единственной строке записана скобочная последовательность, содержащая не более 100000 скобок.

**Выходные данные**

Если данная последовательность правильная, то программа должна вывести строку yes, иначе строку no.

**Примеры**

**входные данные**

()[]

**выходные данные**

yes

**Solution**

Create a stack and run through the string and check if this is the opening bracket, then put it on the stack. And if the closing bracket then we check whether the stack is empty or take the top element of the stack and check it is the opening bracket of this bracket if Yes then we give pop otherwise we return “no”.

After the loop, if the stack is empty, the answer is "yes" or the non-empty answer is "no".

**Java Code**

import java.util.Scanner;

import java.util.Stack;

public class Task9 {

public static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {

Stack<Character> stack = new Stack<>();

String str = sc.next();

for (int i = 0; i < str.length(); ++i) {

if (str.charAt(i) == '(' || str.charAt(i) == '[' || str.charAt(i) == '{')

stack.push(str.charAt(i));

else if (stack.isEmpty() || (stack.peek() != '(' && str.charAt(i) == ')') || (stack.peek() != '[' && str.charAt(i) == ']') || (stack.peek() != '{' && str.charAt(i) == '}'))

{

System.out.println("no");

return;

}

else {

stack.pop();

}

}

if (stack.isEmpty())

System.out.println("yes");

else

System.out.println("no");

}

}

**Problem - 10**

<https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=52#1>

#### **Задача №52. Постфиксная запись**

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D \* обозначает привычное нам (B + C) \* D, а запись A B C + D \* + означает A + (B + C) \* D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

**Входные данные**

В единственной строке записано выражение в постфиксной записи, содержащее цифры и операции +, -, \*. Числа и операции разделяются пробелами. В конце строки может быть произвольное количество пробелов.

**Выходные данные**

Необходимо вывести значение записанного выражения.

**Примеры**

**входные данные**

8 9 + 1 7 - \*

**выходные данные**

-102

**Solution**

Сreating a stack with the Integer type and running through the string. If this symbol is a number then we put it in the stack and if operators then we take the last two added elements from the stack and perform the operation and put the result in the stack.

When you run through the entire length of the string only one element remains in the stack this is the answer.

**Java Code**

import java.util.Scanner;

import java.util.Stack;

public class Task10 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s = sc.nextLine();

Stack<Integer> stack = new Stack<>();

for (int i = 0; i < s.length(); i += 2) {

if (s.charAt(i) == '+') {

int b = stack.pop();

int a = stack.pop();

stack.push(a + b);

} else if (s.charAt(i) == '-') {

int b = stack.pop();

int a = stack.pop();

stack.push(a - b);

} else if (s.charAt(i) == '\*') {

int b = stack.pop();

int a = stack.pop();

stack.push(a \* b);

} else if (s.charAt(i) != ' '){

stack.push(s.charAt(i) - '0');

}

}

System.out.println(stack.pop());

}

}

**Problem - 11**

<https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=112984#1>

#### **Задача №112984. Гоблины и шаманы**

Гоблины Мглистых гор очень любях ходить к своим шаманам. Так как гоблинов много, к шаманам часто образуются очень длинные очереди. А поскольку много гоблинов в одном месте быстро образуют шумную толку, которая мешает шаманам проводить сложные медицинские манипуляции, последние решили установить некоторые правила касательно порядка в очереди.

Обычные гоблины при посещении шаманов должны вставать в конец очереди. Привилегированные же гоблины, знающие особый пароль, встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром.

Так как гоблины также широко известны своим непочтительным отношением ко всяческим правилам и законам, шаманы попросили вас написать программу, которая бы отслеживала порядок гоблинов в очереди.

### **Входные данные**

В первой строке входных данный записано число NN (1≤N≤1051≤N≤105) - количество запросов к программе. Следующие NN строк содержат описание запросов в формате:

* "+ i" - гоблин с номером ii (1≤i≤N1≤i≤N) встает в конец очереди.
* "\* i" - привилегированный гоблин с номером ii встает в середину очереди.
* "-" - первый гоблин из очереди уходит к шаманам. Гарантируется, что на момент такого запроса очередь не пуста.

### **Выходные данные**

Для каждого запроса типа "-" программа должна вывести номер гоблина, который должен зайти к шаманам.

**Примеры**

**входные данные**

7

+ 1

+ 2

-

+ 3

+ 4

-

-

**выходные данные**

1

2

3

**Solution**

**C++ Code**

#include <bits/stdc++.h>

using namespace std;

int main() {

deque < int > d1;

deque < int > d2;

int n, x;

char op;

cin >> n;

for(int i = 0; i < n; i++) {

cin >> op;

if (op == '-') {

cout << d1.front() << endl;

d1.pop\_front();

} else {

cin >> x;

if(op == '+') {

d2.push\_back(x);

} else {

d2.push\_front(x);

}

}

if (d2.size() > d1.size()) {

x = d2.front();

d2.pop\_front();

d1.push\_back(x);

}

}

return 0;

}

**Problem - 12**

<https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=53#1>

**Problem - 13**

<https://leetcode.com/problems/valid-parentheses/>

**20. Valid Parentheses**

**Description**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

**Solution**

**JAVA Code**

class Solution {

public boolean isValid(String s) {

Stack<Character> set = new Stack<>();

for(int i = 0; i < s.length(); i++) {

if (s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[')

set.push(s.charAt(i));

else if (set.isEmpty() || rev(s.charAt(i)) != set.pop())

return false;

}

return (set.isEmpty());

}

public char rev(char ch) {

if(ch == ')')

return '(';

else if (ch == '}')

return '{';

else

return '[';

}

}

**Problem – 14**

<https://leetcode.com/problems/min-stack/>

**155. Min Stack**

**Description**

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* getMin() -- Retrieve the minimum element in the stack.

**Example 1:**

**Input**

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

**Output**

[null,null,null,null,-3,null,0,-2]

**Solution**

**JAVA Code**

class MinStack {

Stack<Integer> stack = new Stack<>();

Stack<Integer> minStack = new Stack<>();

public void push(int x) {

if (minStack.isEmpty() || x <= minStack.peek())

minStack.push(x);

stack.push(x);

}

public void pop() {

if(stack.peek().equals(minStack.peek()))

minStack.pop();

stack.pop();

}

public int top() {

return stack.peek();

}

public int getMin() {

return minStack.peek();

}

}

**Problem - 15**

<https://leetcode.com/problems/backspace-string-compare/>

**844. Backspace String Compare**

**Description**

Given two strings S and T, return if they are equal when both are typed into empty text editors. # means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Example 1:**

**Input:** S = "ab#c", T = "ad#c"

**Output:** true

**Explanation**: Both S and T become "ac".

**Solution**

**JAVA Code**

class Solution {

public boolean backspaceCompare(String S, String T) {

Stack<Character> s = new Stack<>();

Stack<Character> t = new Stack<>();

for (int i = 0; i < S.length(); i++) {

if(S.charAt(i) != '#') {

s.push(S.charAt(i));

} else {

if(!s.isEmpty())

s.pop();

}

}

for (int i = 0; i < T.length(); i++) {

if(T.charAt(i) != '#') {

t.push(T.charAt(i));

} else {

if(!t.isEmpty())

t.pop();

}

}

return s.toString().equals(t.toString());

}

}

**Problem - 16**

<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

**150. Evaluate Reverse Polish Notation**

**Description**

Evaluate the value of an arithmetic expression in [Reverse Polish Notation](http://en.wikipedia.org/wiki/Reverse_Polish_notation).

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

**Note:**

* Division between two integers should truncate toward zero.
* The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

**Example 1:**

**Input:** ["2", "1", "+", "3", "\*"]

**Output:** 9

**Explanation:** ((2 + 1) \* 3) = 9

**Solution**

**JAVA Code**

class Solution {

public int evalRPN(String[] tokens) {

Stack<Integer> stack = new Stack<>();

for (String s : tokens) {

if (s.equals("+")) {

int b = stack.pop();

int a = stack.pop();

stack.push(a + b);

} else if (s.equals("-")) {

int b = stack.pop();

int a = stack.pop();

stack.push(a - b);

} else if (s.equals("\*")) {

int b = stack.pop();

int a = stack.pop();

stack.push(a \* b);

} else if (s.equals("/")) {

int b = stack.pop();

int a = stack.pop();

stack.push(a / b);

} else {

stack.push(Integer.parseInt(s));

}

}

return stack.pop();

}

}