LAB WORK 4

1. Простой стек

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

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**

push 3

push 14

size

clear

push 1

back

push 2

back

pop

size

pop

size

exit

**выходные данные**

ok

ok

2

ok

ok

1

ok

2

2

1

1

0

bye

Code:

#include <iostream>

#include <string>

using namespace std;

#define **SIZE** 100

struct stack {

int storage[SIZE];

int cursor = 0;

void push (int n) {

storage[cursor] = n;

cursor++;

}

int **pop** () {

return storage[--cursor];

}

int **back** () {

return storage[cursor-1];

}

unsigned int **size** () {

return cursor;

}

void **clear** () {

cursor = 0;

}

};

int **main** () {

stack s;

string str;

int n;

while (cin >> str) {

if (str == "push") {

cin >> n;

s.push(n);

cout << "ok" << endl;

}

else if (str == "pop") cout << s.pop() << endl;

else if (str == "back") cout << s.back() << endl;

else if (str == "size") cout << s.size() << endl;

else if (str == "clear") {

s.clear();

cout << "ok" << endl;

}

else if (str == "exit") {

cout << "bye" << endl;

break;

}

}

return 0;

}

Explanation:

We implement the stack using an array, a pointer to the upper empty cell of the array (cursor), and functions that execute the push, pop, back, size, clear commands:

The push function writes to the cell with the cursor number of the element n.

the pop function returns the last element placed on the stack, that is, the element with the number cursor1, while removing it from the stack.

the back function returns the last element placed on the stack, that is, the element with the number cursor1, without removing it from the stack.

the size function returns the value of cursor, i.e. the stack size.

the clear function assigns a value to cursor.

1. Стек с защитой от ошибок

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

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**

size

push 1

size

push 2

size

push 3

size

exit

**выходные данные**

0

ok

1

ok

2

ok

3

bye

**Code:**

#include <iostream>

using namespace std;

struct stack {

long long s[100];

int cursor=0;

void push(long long n){

s[cursor++]=n;

}

long long **pop**(){

return s[--cursor];

}

long long **back**(){

return s[cursor-1];

}

unsigned **size**(){

return cursor;

}

void **clear**(){

cursor=0;

s[cursor]=0;

}

};

int **main**() {

stack x;

string b;

long long n;

while(cin>>b){

if(b=="push"){

cin>>n;

x.push(n);

cout<<"ok"<<endl;

}

else if(b=="pop"){

if(x.size()==0){

cout<<"error"<<endl;

}

else

cout<<x.pop()<<endl;

}

else if(b=="back"){

if(x.size()==0){

cout<<"error"<<endl;

}

else cout<<x.back()<<endl;

}

else if(b=="size"){

cout<<x.size()<<endl;

}

else if(b=="clear"){

x.clear();

cout<<"ok"<<endl;

}

else if(b=="exit"){

cout<<"bye"<<endl;

break;

}

}

return 0;

}

Explanation:

Creating a structure in which we implement all the commands for the stack using functions and a cursor. Cursor points to the last element in the line. Initially, cursor is 0, because at first the stack is empty.

push - writes the n element to the cell with the cursor + 1 number.

size - returns cursor, i.e. the size of the stack.

pop - returns the last element placed on the stack, that is, the element with the number cursor1, while removing it from the stack. If the stack size is 0, the function returns error.

back-returns the last element placed on the stack, i.e. the element with the cursor-1 number. If the stack size is 0, the function returns error.

clear - sets the value to 0.

exit -executed using the break statement, which ends the execution of the loop in which the commands are entered.

1. Простая очередь

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

**Входные данные**

Вводятся команды управления очередью, по одной на строке

**Выходные данные**

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

**входные данные**

size

push 1

size

push 2

size

push 3

size

exit

**выходные данные**

0

ok

1

ok

2

ok

3

bye

**Code:**

#include <iostream>

#include <string>

using namespace std;

struct queue{

int storage[100000];

int start;

int finish;

**queue**(){

start = 0;

finish = 0;

}

void **push**(int n){

storage[finish] = n;

finish++;

}

int **pop**(){

int a = storage[start];

start++;

return a;

}

int **front**(){

return storage[start];

}

int **size**(){

return finish - start;

}

string **clear**(){

finish = 0;

start = 0;

return "ok";

}

string **exit**(){

return "bye";

}

};

int **main**() {

string a;

queue x;

while(cin >> a){

if(a == "push"){

int n;

cin >> n;

x.push(n);

cout << "ok" << endl;

}

if(a == "pop"){

cout << x.pop() << endl;

}

if(a == "front"){

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

}

if(a == "size"){

cout << x.size() << endl;

}

if(a == "clear"){

cout << x.clear() << endl;

}

if(a == "exit"){

cout << x.exit() << endl;

return 0;

}

}

return 0;

}

Explanation:

We implement an abstract queue data type that meets the FIFO principle ("first in, first out") using an array. The queue has a start and end, which are indicated by start and finish, respectively. Initially, the queue is empty, so start = 0, finish = 0. When adding a new item to the queue, we write it to the end. finish is increased by one. The extracted element is taken at the beginning of the queue, then start++. If you need to get the value of the beginning of the queue without extracting it, use the front () function, which returns the value of the first element. To get the queue size, use the size () function, which returns the difference between the end and the beginning of the queue. If the queue needs to be cleared, then we equate finish and start to zero.

1. Очередь с защитой от ошибок

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

Реализуйте структуру данных "очередь".

**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

**входные данные**

size

push 1

size

push 2

size

push 3

size

exit

**выходные данные**

0

ok

1

ok

2

ok

3

bye

Code:

#include <iostream>

using namespace std;

struct stack {

long long s[100];

int cursor=0;

void push(long long n){

s[cursor++]=n;

}

long long **pop**(){

return s[--cursor];

}

long long **back**(){

return s[cursor-1];

}

unsigned **size**(){

return cursor;

}

void **clear**(){

cursor=0;

s[cursor]=0;

}

};

int **main**() {

stack x;

string b;

long long n;

while(cin>>b){

if(b=="push"){

cin>>n;

x.push(n);

cout<<"ok"<<endl;

}

else if(b=="pop"){

if(x.size()==0){

cout<<"error"<<endl;

}

else

cout<<x.pop()<<endl;

}

else if(b=="back"){

if(x.size()==0){

cout<<"error"<<endl;

}

else cout<<x.back()<<endl;

}

else if(b=="size"){

cout<<x.size()<<endl;

}

else if(b=="clear"){

x.clear();

cout<<"ok"<<endl;

}

else if(b=="exit"){

cout<<"bye"<<endl;

break;

}

}

return 0;

}

Explanation:

Creating a structure in which we implement all the commands for the stack using functions and a cursor. Cursor points to the last element in the line. Initially, cursor is 0, because at first the stack is empty.

push - writes the n element to the cell with the cursor + 1 number.

size - returns cursor, i.e. the size of the stack.

pop - returns the last element placed on the stack, that is, the element with the number cursor1, while removing it from the stack. If the stack size is 0, the function returns error.

back-returns the last element placed on the stack, i.e. the element with the cursor-1 number. If the stack size is 0, the function returns error.

clear - sets the value to 0.

exit -executed using the break statement, which ends the execution of the loop in which the commands are entered.

1. Простой дек

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

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.

**входные данные**

push\_back 1

back

exit

**выходные данные**

ok

1

bye

#include <iostream>

#include <string>

using namespace std;

#define **SIZE** 10000

struct deque{

int storage[SIZE];

int \_size = 0, start = 0, end = 0;

void push\_front(int n){

if (\_size == SIZE) { cout << "Full\n"; }

else if (\_size == 0) { storage[start] = n; \_size++; }

else {

if (start == 0) { start = SIZE - 1; storage[start] = n; \_size++; }

else { start--; storage[start] = n; \_size++; }

}

}

void **push\_back**(int n){

if (\_size == SIZE) { cout << "Full\n"; }

else if (\_size == 0) { storage[end] = n; \_size++; }

else {

if (end == SIZE-1) { end = 0; storage[end] = n; \_size++; }

else { end++; storage[end] = n; \_size++; }

}

}

int **pop\_front**(){

int b;

if (\_size != 0 && (start == end)) { b = storage[start]; \_size--; return b; }

else if (\_size == 0) { return -1; }

else

{

b = storage[start];

if (start == SIZE - 1) { start = 0; \_size--; }

else { start++; \_size--; }

return b;

}

}

int **pop\_back**(){

int b;

if (\_size != 0 && (start == end)) { b = storage[end]; \_size--; return b; }

else if (\_size == 0) { return -1; }

else

{

b = storage[end];

if (end == 0) { end = SIZE - 1; \_size--; }

else { end--; \_size--; }

return b;

}

}

int **front**() { if (\_size == 0) return -1; else return storage[start]; }

int **back**() { if (\_size == 0) return -1; else return storage[end]; }

int **size**() { return \_size; }

void **clear**(){

\_size = 0;

start = end = 0;

}

};

int **main**(){

deque d;

string s;

int n;

while (cin >> s){

if (s == "push\_front"){

cin >> n;

d.push\_front(n);

cout << "ok\n";

} else if (s == "push\_back"){

cin >> n;

d.push\_back(n);

cout << "ok\n";

} else if (s == "pop\_front") {

cout << d.pop\_front() << endl;

} else if (s == "pop\_back") {

cout << d.pop\_back() << endl;

} else if (s == "front") {

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

} else if (s == "back"){

cout << d.back() << endl;

} else if (s == "size") {

cout << d.size() << endl;

} else if (s == "clear") {

d.clear();

cout << "ok\n";

} else if (s == "exit") {

cout << "bye\n";

return 0;

}

}

return 0;

}

Explanation:

Implementing a two-way queue using an array. Due to the specific structure of the deck, it is necessary to specify the area that is active during the execution of push\_front, push\_back, pop\_front, pop\_back, front and back operations. This is either the beginning of the deck(the start variable), or its end(the end variable).

1. before performing push\_front and push\_back operations, it is mandatory to check the deck for fullness and, respectively, for emptiness. Thus, if the size of the deck is equal to the maximum allowed number of elements in the data structure, the program outputs Full — none of the above two commands are executed. Similarly, if the size of the deck is zero, increase it by one. Otherwise: commands are executed successfully with checking the conditions presented in the program code. The program outputs "ok".

2. next, go to the pop\_front and pop\_back commands. Here, as in the case of previous operations, first of all we check the Dec for emptiness. If the two-way queue contains no elements, the program outputs -1. It is also important to check whether the beginning and end of the deck are equal. in this case, you need to reduce the size of the structure by one. If the Dec contains at least one element, the commands are executed successfully and the values of the extracted elements are output.

3. Similarly, before executing the front and back commands, we check whether the deck contains at least one element. If not, -1 is output. Otherwise: the values of the first and last element are output, respectively.

4. use the size command to get the size of the soundboard. The program displays the number of elements in the soundboard.

5. next, use the clear command to remove all elements from the deck: assign the \_size variable(the size of the deck) and the start and end variables to zero. The program outputs" ok".

6. the exit command outputs "bye" — the program exits.

All required values are displayed in the body of the main() function using the string s.

1. Дек с защитой от ошибок

[https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=61#1](https://informatics.msk.ru/mod/statements/view.php?id=207&chapterid=61%231)

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 в начало дека новый элемент. Программа должна вывести ok.

**push\_back-**Добавить 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

Code:

#include <iostream>

#include <string>

using namespace std;

struct stack{

int storage[1000], end=0, sizze=0, start=1, lol=100;

int push\_back(int x){

end = (end + 1)%lol;

storage[end] = x;

sizze++;

}

int **push\_front**(int x){start = (start - 1+lol)%lol;

storage[start] = x;

sizze++;}

void **pop\_back**(){end = (end - 1+lol)%lol;

sizze--;}

void **pop\_front**(){start = (start + 1)%lol;

sizze--;}

int **back**() const{return storage[end];}

int **size**() const{return sizze;}

void **clear**(){end = 0;

sizze = 0;

start = 1;

}

int **front**() const{

return storage[start];

}

};

int **main**() {

stack storage;

string s;

int n;

while(cin >> s){

if(s == "push\_back"){

cin >> n;

storage.push\_back(n);

cout << "ok\n";

}else if(s == "push\_front"){

cin >> n;

storage.push\_front(n);

cout << "ok\n";

} else if(s == "pop\_back"){

if(storage.size()){

cout << storage.back() << endl;

storage.pop\_back();

}else{

cout << "error\n";

}

}else if(s == "pop\_front"){

if(storage.size()){

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

storage.pop\_front();

}else{

cout << "error\n";

}

}else if(s == "front"){

if(storage.size()){

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

} else{

cout << "error\n";

}

} else if(s == "back"){

if(storage.size()){

cout << storage.back() << endl;

}else{

cout << "error\n";

}

}else if(s == "size"){

cout << storage.size() << endl;

}else if(s == "clear"){

storage.clear();

cout << "ok\n";

}else if(s == "exit"){

cout << "bye\n";

return 0;

}

}

return 0;

}

**Explanation:**

Before performing push\_front and push\_back operations, it is mandatory to check the deck for fullness and, respectively, for emptiness. Thus, if the size of the deck is equal to the maximum allowed number of elements in the data structure, the program outputs Full — none of the above two commands are executed. Similarly, if the size of the deck is zero, increase it by one.

2. next, go to the pop\_front and pop\_back commands. Here, as in the case of previous operations, first of all we check the Dec for emptiness. If the two-way queue contains no elements, the program outputs -1. It is also important to check whether the beginning and end of the deck are equal. in this case, you need to reduce the size of the structure by one.

3. Similarly, before executing the front and back commands, we check whether the deck contains at least one element. If not, -1 is output. Otherwise: the values of the first and last element are output, respectively.

4. use the size command to get the size of the soundboard. The program displays the number of elements in the soundboard.

5. next, use the clear command to remove all elements from the deck: assign the \_size variable(the size of the deck) and the start and end variables to zero. The program outputs" ok".

6. the exit command outputs "bye" — the program exits.

1. Списки по классам

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=49#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=49%231)

***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 |

Code:

int main(){

queue<string> q[3];

int n;

string s;

while(cin>>n>>s){

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

}

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

while(!q[i-9].empty()){

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

q[i-9].pop();

}

}

return 0;

}

Explanation:

I declared queue which size is 3 and push to this queue elements by this class, and then I just printed elements of queue starting from 9 class to 11 class.

1. Игра в пьяницу

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=50#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=50%231)

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

Code:

int k = 0;

while(!q1.empty() && !q1.empty()){

k++;

if(q1.front() > q2.front() && (q1.front() != 0 && q2.front() != 9) || (q1.front()==0 && q2.front()==9)){

q1.push\_back(q1.front());

q1.push\_back(q2.front());

q1.pop\_front();

q2.pop\_front();

}else{

q2.push\_back(q1.front());

q2.push\_back(q2.front());

q1.pop\_front();

q2.pop\_front();

}

if(k == 1000000){

cout<<"botva"<<endl;

return 0;

}

}

if(q1.empty()) cout<<"second"<<' ';

else cout<<"first"<< ' ';

cout<<k;

Explanation:

In this task I took two queue and add card of first players to the first queue and second payer's card to the second queue. Then run the through the loop until one of them is empty. And I will compare front card of two players, if first player's card greater than second players I will add this two card at the end of first queue, else second players card greater than first’s then I will push this two card at the end of second queue. And each time I will count how many time has this compare, if it will be equal to 1000000 then I will return "botva"

If first player’s card will be empty, it means second player won, otherwise first player.

1. Правильная скобочная последовательность

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=51#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=51%231)

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

Code:

#include <iostream>

#include <stack>

using namespace std;

stack < char> st;

string s;

int **main**() {

cin >> s;

int n = s.size();

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

if (s[i] == '(' || s[i] == '[' || s[i] == '{')

st.push(s[i]);

else {

if (st.empty()) {

cout << "no";

return 0;

}

char x = '(';

if (s[i] == ']')

x = '[';

else if (s[i] == '}')

x = '{';

if (st.top() != x) {

cout << "no";

return 0;

}

st.pop();

}

if (st.empty())

cout << "yes";

else

cout << "no";

return 0;

}

Explanation:

Instead of the \rm depth counter, you should create a stack in which we will put opening brackets as they are received. If the current character in the string, the opening brace, then put it on the stack, and if the closing — then check that the stack is not empty, and that at its top is a brace of the same type as the current, and then remove the bracket from the stack. If any of the conditions are not met, or the stack is not empty at the end of the algorithm, then the sequence is not a correct bracket, otherwise it is.

1. Постфиксная запись

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=52#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=52%231)

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

Code:

int main(){

char c;

string s="";

while(cin>>c)

s+=c;

stack<int> st;

for(string x : s){

if(x == "+"){

int a = st.top();

st.pop();

int b = st.top();

st.pop();

st.push(b+a);

}else if(x == "-"){

int a = st.top();

st.pop();

int b = st.top();

st.pop();

st.push(b-a);

}else if(x == "\*"){

int a = st.top();

st.pop();

int b = st.top();

st.pop();

st.push(b\*a);

}else if(x == "/"){

int a = st.top();

st.pop();

int b = st.top();

st.pop();

st.push(b/a);

}else{

int k = stoi(x);

st.push(k);

}

}

cout<<st.top();

return 0;

}

Explanation:

In this task we will look through for all of char and

If char equal to ‘+' then we will get last two element of stack and we will add this two element then we will push this element in place of last two elements.

If char equal to ‘-‘then we will get last two element of stack and subtract first element from second and we will push this element in place of last two elements.

If char equal to ‘\*’then we will get last two element of stack and multiply this two and we will push this element in place of last two elements.

If char equal to ‘/‘then we will get last two element of stack and divide first element from second and we will push this element in place of last two elements.

In the end we will print top element of stack.

1. Гоблины и шаманы

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=112984#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=112984%231)

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

Code:

Explanation:

1. Контейнеры

[https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=53#1](https://informatics.msk.ru/mod/statements/view.php?id=206&chapterid=53%231)

На складе хранятся контейнеры с товарами *N* различных видов. Все контейнеры составлены в *N* стопок. В каждой стопке могут находиться контейнеры с товарами любых видов (стопка может быть изначально пустой).

Автопогрузчик может взять верхний контейнер из любой стопки и поставить его сверху в любую стопку. Необходимо расставить все контейнеры с товаром первого вида в первую стопку, второго вида – во вторую стопку и т.д.

Программа должна вывести последовательность действий автопогрузчика или сообщение о том, что задача решения не имеет.

***Формат входных данных***

В первой строке входных данных записано одно натуральное число *N*, не превосходящее 500. В следующих *N* строках описаны стопки контейнеров: сначала записано число *ki* – количество контейнеров в стопке, а затем *ki* чисел – виды товара в контейнерах в данной стопке, снизу вверх. В каждой стопке вначале не более 500 контейнеров (в процессе переноса контейнеров это ограничение может быть нарушено).

***Формат выходных данных***

Программа должна вывести описание действий автопогрузчика: для каждого действия напечатать два числа – из какой стопки брать контейнер и в какую стопку класть. (Обратите внимание, что минимизировать количество операций автопогрузчика не требуется.) Если задача не имеет решения, необходимо вывести одно число 0. Если контейнеры изначально правильно размещены по стопкам, то выводить ничего не нужно.

***Пример***

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| 3 4 1 2 3 2 0 0 | 1 2 1 3 1 2 |

Code:

*#include <iostream>*

using namespace std;

struct Elem {

int val; Elem \*pr;

};

struct Stack {

Elem \*up;

Stack () { up = NULL; }

void push (int a) {

Elem \*tmp;

tmp = new Elem();

tmp->val = a;

tmp->pr = up;

up = tmp;

}

int pop () {

if (up != NULL) {

int v = up->val;

Elem \*tmp = up;

up = up->pr;

delete tmp;

return v;

} else {

cerr << "empty stack" << endl;

return 0;

}

}

int back () {

return up->val;

}

bool empty () {

return up == NULL;

}

void free () {

while (!empty ()) pop ();

}

~Stack () {

free ();

}

};

int N, k;

Stack cont[501];

int main () {

int tmp, i, j = 0, save = 1, qsave = 0;

cin >> N; save = 1;

if (N == 2) {

bool ok = true, first = true;

cin >> k;

if (k) {

cin >> tmp;

i = 1;

while (tmp == 1 && i < k) {

cin >> tmp; ++i;

}

j = tmp == 2;

while (i < k) {

cin >> tmp;

if (tmp == 1) ok = false; ++j; ++i;

}

}

cin >> k;

if (k) {

cin >> tmp; i = 1;

while (tmp == 2 && i < k)

{ cin >> tmp; ++i; }

if (j && tmp == 1)

ok = false;

else if (tmp == 1)

first = false;

j += tmp == 1;

while (i < k) {

cin >> tmp;

if (tmp == 2)

ok = false;

++j; ++i;

}

}

if (ok) {

if (first)

for (i = 0; i < j; ++i)

cout << "1 2" << endl;

else

for (i = 0; i < j; ++i)

cout << "2 1" << endl;

}

else cout << 0;

return 0;

}

for (i = 1; i <= N; ++i) {

cin >> k;

if (k) {

cin >> tmp; j = 1;

while (tmp == i && j < k) {

cin >> tmp;

++j;

}

if (tmp != i)

cont[i].push (tmp);

while (j < k) {

cin >> tmp; cont[i].push (tmp);

++j;

}

}

}

for (i = 1; i < N; ++i){

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

while (!cont[i].empty () && cont[cont[i].back ()].empty ()) {

cout << i << " " << cont[i].pop() << endl;

}

while (!cont[i].empty () && !cont[cont[i].back ()].empty ()) {

cout << i << " " << N << endl; cont[N].push (cont[i].pop ());

}

}

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

cout << i << " " << N << endl;

cont[N].push (cont[i].pop ());

}

}

while (!cont[N].empty ()) {

tmp = cont[N].pop ();

if (tmp == N) {

cout << N << " " << save << endl;

++qsave;

} else if (tmp == save) {

save = 1 + save % 2;

for (i = 0; i < qsave; ++i) {

cout << tmp << " " << save << endl;

}

cout << N << " " << tmp << endl;

} else {

cout << N << " " << tmp << endl;

}

}

for (i = 0; i < qsave; ++i) {

cout << save << " " << N << endl;

}

return 0;

}

Explanation:

The easiest way to start is to put everything in the first pile.Special cases are one and two stacks. In the first one, you don't need to transfer anything. In the second case, the easiest way to do this is to move the upper containers # 2 from it to the second one. If at the same time there are no containers # 2 left in it (all of them were on top and successfully transferred), everything is fine. If not, you can't transfer it.

Consider the General case when there are at least three containers. In this case, you can always transfer it. Move all the containers from the first stack to their corresponding ones, and containers # 1 will be put in the second stack (along with containers #2). Now let's analyze the second one. Containers #2 in the third, and #1 in place – in the first. Now all containers # 2 in the second with the third. Note that it is enough to have only three stacks, and transfers to containers with large numbers just immediately output.

1. Valid Parentheses

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

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.

**Example 1:**

**Input:** s = "()"

**Output:** true

Code:

bool isValid(string s) {

stack<char> st;

for(char c : s){

if(c == '('|| c == '{' || c == '['){

st.push(c);

}else{

if(st.empty()) return false;

if(c == ')' && st.top() != '(') return false;

if(c == '}' && st.top() != '{') return false;

if(c == ']' && st.top() != '[') return false;

st.pop();

}

}

return st.empty();

Explanation:

Instead of the counter, you should create a stack in which we will put opening brackets as they are received. If the current character in the string, the opening brace, then put it on the stack, and if the closing — then check that the stack is not empty, and that at its top is a brace of the same type as the current, and then remove the bracket from the stack. If any of the conditions are not met, or the stack is not empty at the end of the algorithm, then the sequence is not a correct bracket, otherwise it is.

1. Min Stack

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

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]

Code:

class MinStack {

stack<int> data;

stack<int> min;

public:

void push(int x) {

if (min.empty()) {

data.push(x);

min.push(x);

} else {

data.push(x);

if (x <= min.top())

min.push(x);

}

}

void pop() {

if (!min.empty()) {

if (data.top() == min.top())

min.pop();

data.pop();

}

}

int top() {

return data.top();

}

int getMin() {

return min.top();

}

};

Explanation:

The key idea is use a another stack to store the minimum value of the corresponding stack. Put differently, min[i] equals the minimum element where data[i] is the top of this sub-stack.We can use a full size of min where its size equals the data's, but it's not necessary.

We should pop the element in min IFF there's match of data.top(). If we have multiple same minima, for example [0, 1, 0] in data, then the min should be [0, 0].  
Otherwise, the the pop operation wouldn't work properly, since that you need 2 0s.  
As a result, we should push the element if x <= min.top().

1. Backspace String Compare

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

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".

Code:

class Solution {

public:

bool backspaceCompare(string S, string T) {

string s, t;

for (auto c : S){

if (c == '#') {

if (!s.empty())

s.pop\_back();

}else s.push\_back(c);

}

for (auto c : T)

if (c == '#'){

if (!t.empty())

t.pop\_back();

} else

t.push\_back(c);

return s == t;

}

};

Explanation:

A better idea would be to loop over each string and delete a character **in the original string** whenever we come across a '#' (if there is one to be deleted). Obviously, we delete the # itself.

1. Evaluate Reverse Polish Notation

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

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

Code:

class Solution {

public:

int evalRPN(vector<string>& tokens) {

if (tokens.empty()) return -1;

stack<long long> s;

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

if (isNum(tokens[i])){

s.push(stoll(tokens[i]));

} else {

long long tmp1 = s.top(); s.pop();

long long tmp2 = s.top(); s.pop();

if (tokens[i] == "+"){

s.push(tmp2 + tmp1);

}else if (tokens[i] == "-"){

s.push(tmp2 - tmp1);

}else if (tokens[i] == "\*"){

s.push(tmp2 \* tmp1);

}

else if (tokens[i] == "/") {

s.push(tmp2 / tmp1);

}

}

}

return s.top();

}

private:

bool isNum(string s){

return s != "+" && s != "-" && s != "\*" && s != "/";

}

};

Explanation:

The Reverse Polish Notation is a stack of operations, thus.Stack to solve this problem. As you can see, I add every token as an integer in the stack, unless it's an operation. In that case, I pop two elements from the stack and then save the result back to it. After all operations are done through, the remaining element in the stack will be the result.