# 16-Laboratoriya mashg’uloti.

**Konteyner steki. Stackning asosiy operatsiyalari**

Stek bu *LIFO* (Last In - First Out - "oxirgi kelgan – birinchi ketadi"), shunday o’zgaruvchan uzunlikdagi ketma-ketlik, ro’yhatki, unda tuzilmaga elementlarni kiritish va chiqarish amallari bir tomondan, ya’ni stek uchidan amalga oshiriladi. Stek ustida bajariladigan asosiy amallar:

* + yangi elementni qo„shish;
  + elementni o„chirish;
  + stek elementlar sonini aniqlash;
  + stekni tozalash.

Stekni statik xotirada vektor ko’rinishida ifodalashda stek uzunligini ko„rsatuvchi Ko’rsatkich ishlatiladi. Bu Ko’rsatkich stekdagi 1-bo’sh joyni ko„rsatadi. Dastlab hali stek bo’shligida bu Ko’rsatkich R=0 bo’ladi. Quyidagi rasmda stekda 6 ta element mavjudligi uchun R=7 bo’ladi (1-rasm).

kirish chiqish

Stek uchi

R=7

Stek tubi

1-rasm. Stek tuzilmasi

Stekka yangi element kiritilayotganda stek Ko’rsatkichi (R) ko’rsatayotgan adresga yoziladi va shundan keyin bu Ko’rsatkich bittaga oshiriladi. Stekdan elementni o„chirishda Ko’rsatkichning qiymati bittaga kamaytiriladi va shu adresdagi element o„chiriladi. Stekni tozalash amalini bajarish uchun stek

Ko’rsatkichi R ga stek uchun ajratilgan xotira sohasining boshlang„ich adresi qiymati beriladi. R stekdagi elementlar sonini bildiradi.

# Operatsiyalar

# *stek::*stack ()

* Sukutlikda konstruktor.
* Bo’sh stekni yaratadi.
* explicit stek < ::stack (const Containers *cont)*
* *cont* elementlari bilan initsializatsiya bo’lgan stekni yaratadi
* *cont* barcha elementlari stekga nusxalanadi size\_type *stek::*size () const
* Joriy elementlar sonini qaytaradi.
* Stekda elementlarning yo'qligini tekshirish uchun empty() funktsiyasidan foydalanish tavsiya etiladi, chunki u tez ishlaydi.

bool *stek:*:empty ( ) const

* Stekning bo'shligini tekshiradi.
* Ekvivalent(lekin tez ishlashi mumkin):

*stek:* :size()==0

void *stek::*push (const *value\_type& elem)*

- *Elem* ning nusxasini stekga kiritadi, natijada u yangi birinchi elementga aylanadi.

*value\_type& stek::*top( )

const *valuejiypeb, stek::*top( ) const

* Ikkala shakllar ham stekning yuqori elementini qaytaradi, ya'ni oxirgi kiritilgan elementni (barcha boshqa elementlardan keyin) qaytaradi.
* Chaqirishdan oldin, stek kamida bitta element (size( )> 0) mavjudligiga ishonch hosil qilishingiz kerak, aks holda qo'ng'iroq chaqirilishning oldindan aniqlanmaydigan oqibatlariga olib keladi.
* Birinchi shakl konstantasiz steklar uchun murojaatni qaytaradi, bunisi stekdagi yuqori elementni o'zgartirishga imkon beradi. Bunisi yaxshimi yoki yo'qmi, o'zingni tanlang.
* void *stek*::ror( )
* Stekdan yuqori elementni, ya'ni oxirgi kiritilgan elementni (barcha boshqa elementlardan keyin) o'chiradi.
* Funktsiyaning qaytish qiymati yo'q. Yuqori elementning qiymatini qayta ishlash uchun avval top() funktsiyasini chaqirishingiz kerak.
* Chaqirishdan oldin, stek kamida bitta element (size( )> 0) mavjudligiga

ishonch hosil qilishingiz kerak, aks holda chaqiruv kutilmagan holatga olib keladi.

* bool *taqqoslash* (const sgek& *stackl.* const *stek& stack2*)
* bir xil turdagi ikkita steklarni taqqoslash natijasini qaytaradi.
* *taqqoslash* parametri quyidagi operatsiyalardan biridir:
* operator == operator != operator < operator > operator <= operator >=
* Steklar bir-biriga teng hisoblanadi, agar ularda elementlar soni bir xil bo’lsa, agar elementlarning juftligi mos kelsa va bir hil ketma-ketligda bo’lsa (ya'ni tenglik uchun ikkita mos keladigan elementni tekshiruvchi har doim true ni beradi).
* Konteynerlar orasidagi "katta/kichik" nisbati leksik mezonlarga muvofiq tekshiriladi, leksikografiya mezonlari leksicographical\_cornpare() algoritmining tavsifida ko'rib chiqiladi.

## C++ tilida stekni statik ko’rinishda, ya’ni bir o’lchashhamli massiv ko’rinishida amalga oshirishga misol:

Masalaning qo’yilishi: Elementlari butun sonlardan iborat stekning juft qiymatli elementlari o„chirilsin. Aytaylik, stek uchun 10 ta joy ajratilgan bo„lsin, bunda dastlab stek bo’shligi sababli R=0 bo’ladi. Stekga yangi element qo„shish va chiqarish, stek bo’shligini va to’laligini tekshirish funksiyalaridan foydalanib shu masalani yechamiz.

# Algoritm

* 1. Agar stek to„lmagan bo’lsa elementlarni kiritamiz. Stekning toq elementlarini saqlab turish uchun yangi **b[]** massiv e’lon qilamiz.
  2. Agar stek bo’sh bo„lmasa, 3-qadamga o’tish, aks holda 4-qadamga o’tish.
  3. Stek uchidagi elementni olamiz va juftlikka tekshiramiz. Agar element toq bo’lsa b massivga joylaymiz. 2-qadamga o’tish.
  4. b massiv elementlarini teskari tartibda stekka joylash.
  5. Stek tarkibini ekranga chiqarish.

**Dastur kodi**

#include <iostream>

using namespace std;

int a[10],R=0,n;//bu yerda n stekka kiritilishi kerak bo'lgan elementlar soni.

int kiritish(int s){

a[R]=s; R++;

}

int chiqarish(){

R--;

return a[R];

}

bool isEmpty(){

if(R==0) return true;

else return false;

}

bool isFull(){

if(R>=10) return true;

else return false;

}

int print(){

int i=0,c[n];

while(!isEmpty()){

c[i]=chiqarish();

cout<<c[i]<<" ";i++;

}

for(int j=i-1;j>=0;j--)

kiritish(c[j]);

}

int main(){ int n,s;

cout<<"n=";

cin>>n;

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

if(!isFull()){

cin>>s;

kiritish(s);}

else{

cout<<"stek to'ldi"; n=i;break;}

}

cout<<"\nstek elementlari: ";

print();

int b[n],k=0;

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

s=chiqarish();

if(s%2!=0) b[k++]=s;

}

for(int i=k-1;i>=0;i--)

kiritish(b[i]);

cout<<"\nnatijaviy stek elementlari: ";

print();

system("PAUSE");

}

Dasturning bajarilishi natijasi:

*n =5*

*6*

*7*

*9*

*8*

*11*

*stek elementlari: 11 8 9 7 6*

*natijaviy stek elementlari: 11 9 7*

**Класс Stack**

Современные программисты сейчас практически не используют **Stack**, который слишком прост и не очень гибок. Тем не менее, изучить его стоит, может именно он вам и пригодится.

**Stack** является подклассом класса **Vector**, который реализует простой механизм типа "последний вошёл - первый вышел" (LIFO). Можно представить процесс в виде детской пирамидки, когда вы по одному нанизываете диск на колышек. И снять диски вы сможете только по порядку, начиная с верхнего.



Напишем простейший пример применения стека.

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

stack.push(0);

stack.push(1);

stack.push(2);

System.out.println("Текущий стек: " + stack);

System.out.println("Удаляем: " + stack.pop());

System.out.println("Удаляем: " + stack.pop());

System.out.println("Удаляем: " + stack.pop());

System.out.println("Текущий стек: " + stack);

Получим результат.

I/System.out: Текущий стек: [0, 1, 2]

I/System.out: Удаляем: 2

I/System.out: Удаляем: 1

I/System.out: Удаляем: 0

I/System.out: Текущий стек: []

Метод **push()** помещает объект в стек, а метод **pop()**, наоборот, вытаскивает объект из стека.

Пример с числами слишком скучный. Давайте позовём на помощь котов. Создадим простой класс **Cat**:

package ru.alexanderklimov.expresscourse;

public class Cat {

private String mName;

private int mAge;

public Cat(String name, int age) {

mName = name;

mAge = age;

}

@Override

public String toString() {

return this.mName;

}

}

Представьте себе, что имеется длинная узкая труба, запаянная с одного конца. Мы заталкиваем в трубу пушистого друга, поэтому метод называется "пуш" (**push()**). А чтобы вытащить кота, хватаем его за попу (метод **pop()**). Давайте запихаем в трубу трёх котов, а потом вытащим их.

Cat barsik = new Cat("Барсик", 4);

Cat murzik = new Cat("Мурзик", 6);

Cat vaska = new Cat("Васька", 9);

Stack<Cat> catStack = new Stack<>();

catStack.push(barsik);

catStack.push(murzik);

catStack.push(vaska);

Log.i(TAG, "Текущий стек: " + catStack);

Log.i(TAG, "Брысь " + catStack.pop());

Log.i(TAG, "Кто последний? " + catStack.peek().toString());

Log.i(TAG, "Брысь " + catStack.pop());

Log.i(TAG, "Кто последний? " + catStack.peek().toString());

Log.i(TAG, "Брысь " + catStack.pop());

Log.i(TAG, "Никого? " + catStack.empty());

try {

Log.i(TAG, "Кто последний? " + catStack.peek().toString());

}catch (EmptyStackException e)

{

Log.i(TAG, "Пустой стек. Некого прогонять");

}

Log.i(TAG, "Текущий стек: " + catStack);

У нас есть три кота - Барсик, Мурзик и Васька. В такой последовательности мы их запихнули в трубу и проверяем текущий стек.

I/ExpressCourse: Текущий стек: [Барсик, Мурзик, Васька]

Вызываем метод **pop()** первый раз. Как видите, мы не указываем позицию элемента, так стек работает только с последним элементом. Последним был Васька. Чтобы узнать, кто теперь последний в стеке, не удаляя его оттуда, нужно вызвать метод **peek()**.

I/ExpressCourse: Брысь Васька

I/ExpressCourse: Кто последний? Мурзик

Повторяем этот шаг ещё раз и вытаскиваем кота Мурзика. Затем и Барсика.

I/ExpressCourse: Брысь Мурзик

I/ExpressCourse: Кто последний? Барсик

I/ExpressCourse: Брысь Барсик

Чтобы убедиться, что в трубе никого не осталось, вызываем метод **empty()**, который возвращает булево значение.

I/ExpressCourse: Никого? true

Если при пустом стеке вызвать методы **pop()** или **peek()**, то программа закроется с ошибкой. Чтобы избежать подобной ситуации, нужно обработать исключение **EmptyStackException**. Тогда программа будет работать без сбоев.

I/ExpressCourse: Пустой стек. Некого прогонять

В конце выводим информацию о пустом стеке.

I/ExpressCourse: Текущий стек: []

У класса также есть метод **int search(Object o)**, который ищет заданный элемент в стеке, возвращая количество операций **pop()**, которые требуются для того чтобы перевести искомый элемент в вершину стека. Если заданный элемент в стеке отсутствует, этот метод возвращает -1.

# Topshiriqlar

1. Stek birinchi va oxirgi elementlari o’rni almashtirilsin.
2. Stek elementlari teskari tartibda joylashtirib chiqilsin.
3. Stek o„rtasidagi element o„chirib tashlansin. Agar stek elementi toq bo’lsa, bitta element, aks holda ikkita element o„chirilsin.
4. Stekning juft o’rinda turgan elementlari o„chirilsin.
5. Stek o„rtasiga '\*' belgi joylashtirilsin.
6. Stek eng kichik elementi topilsin va undan keyin 0 joylashtirilsin.
7. Stek eng katta elementi topilsin va undan keyin 0 joylashtirilsin.
8. Stek eng kichik elementi o„chirilsin.
9. Stekda birinchi elementga teng barcha elementlar o„chirilsin.
10. Stek oxirgi elementiga teng barcha elementlar o„chirilsin.
11. Stek eng katta elementi o„chirilsin.
12. Stek eng kichik elementi topilsin va uning o’rniga 0 joylashtirilsin.

Har bir talaba 4 tadan misol ishlashi kerak.