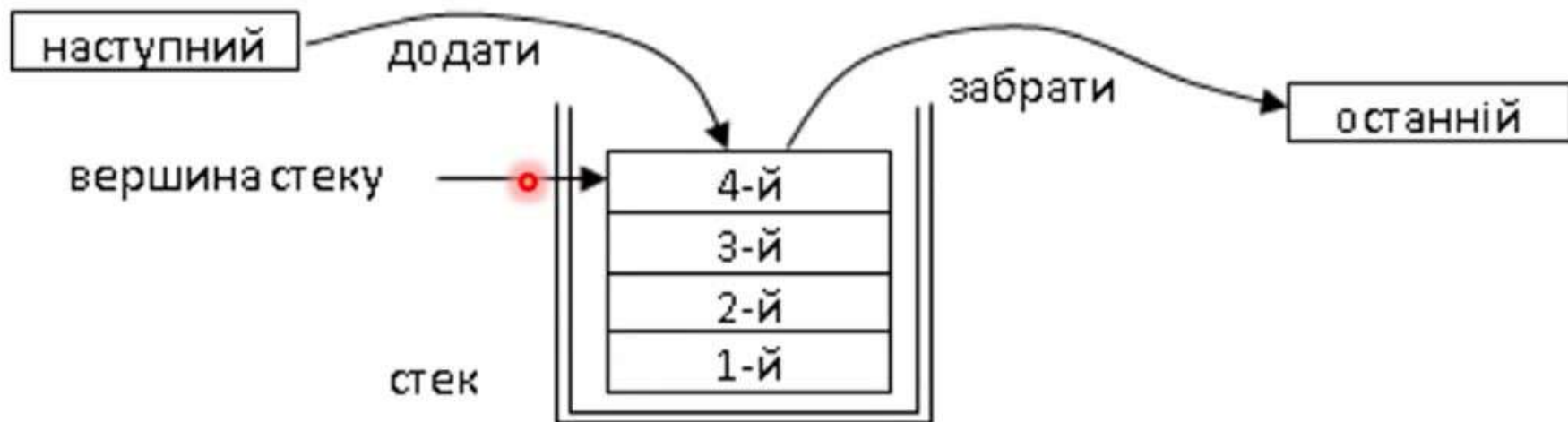


# ТЕМА: Стеки і черги стандартної бібліотеки шаблонів.

## Принцип функціонування стеку

Стек – це спеціальна структура даних, яка функціонує за принципом LIFO: «останнім зайшов – першим вийшов». Стек можна зобразити як модель сукупності предметів, які складають один поверх іншого в коробку, заклеєну з одного кінця:



Стек працює за такими правилами.

1) Початково стек пустий.

2) Стек працює лише в термінах операцій «додати» і «забрати».

3) Кожен наступний елемент можна додати до стеку лише поверх попереднього.

Останній доданий елемент називають вершиною стеку.

4) Забрати з стеку можна лише останній доданий елемент, тоді вершиною стеку стає передостанній.

5) Вершина стеку автоматично зсувається при кожній операції додавання/вибирання.

6) Вибрати з порожнього стеку неможливо. З точки зору програмування це є помилка виконання програми.

7) Максимальний розмір стеку в принципі не обмежений, хоча з точки зору програмної реалізації таке обмеження може бути, тоді може виникнути помилка «переповнення стеку».

Де використовують стеки? 1) В прикладному програмуванні – для задач, розв’язок яких природньо вписується в структуру стеку. 2) В системному програмуванні – для реалізації зв’язків між функціями, які викликають одна одну довільним чином в процесі виконання програми. 3) В системах комп’ютерного керування об’єктами – для моделювання процесів динамічної поведінки об’єктів.

## Клас стеку бібліотеки STL

Такий клас позначають словом `stack`. Для використання стеків треба включити в програму директиву `#include <stack>`.

Для оголошення стеку треба записати тип його елементів:

```
struct ftime { /* . . . . . */ } ; // . . . . .  
stack<int> ast; // початково стек завжди пустий  
stack<char> bst;  
stack<ftime> cst;
```

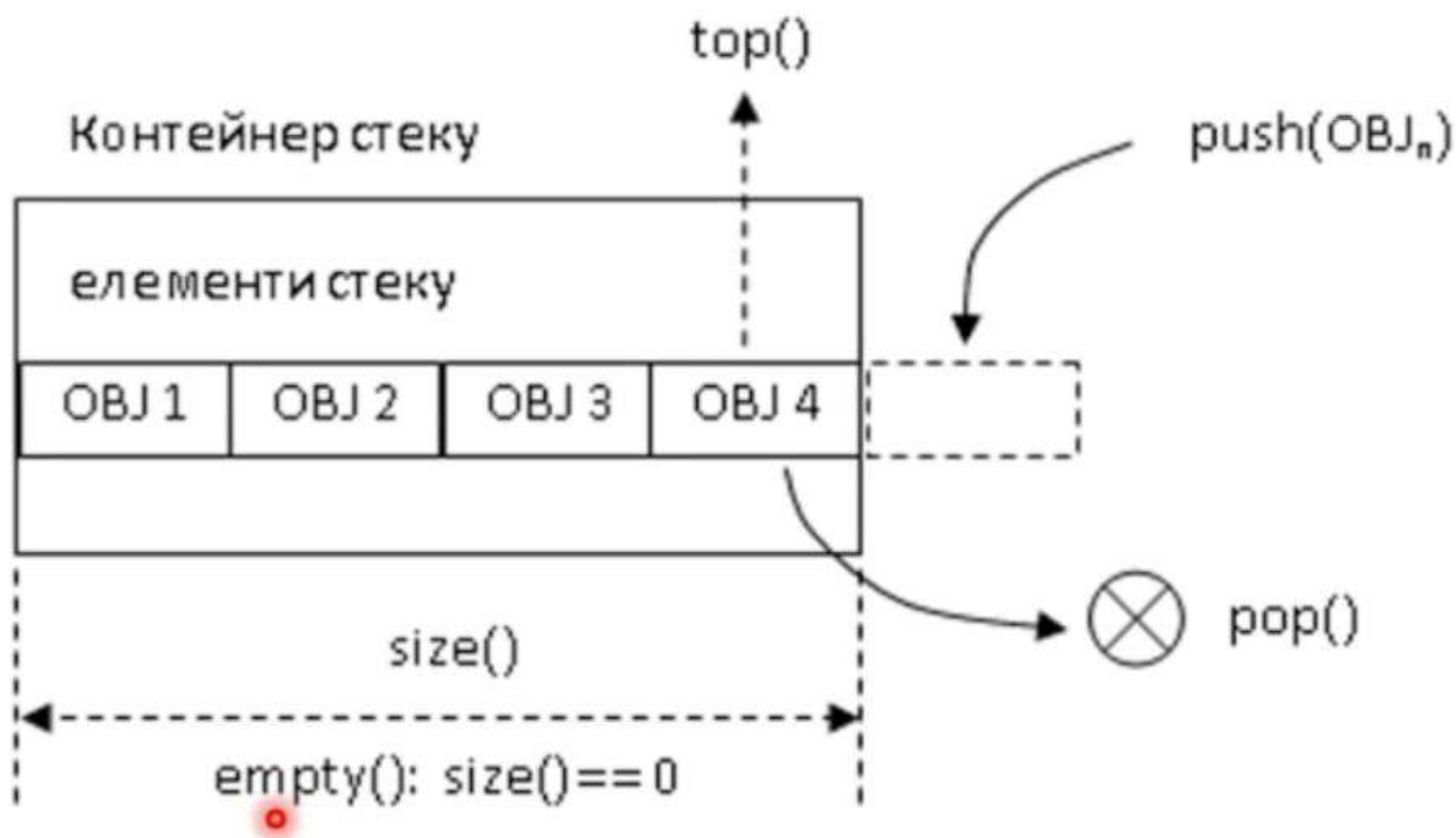
Клас `stack` має такі методи:

<code>void push(...)</code>	додати елемент в кінець стеку
<code>void pop()</code>	викреслити елемент з вершини стеку
<code>const value_type &amp;top()</code> <code>const</code>	посилання на вершину стеку; за цим посиланням елемент можна лише прочитати, але не редагувати; сам елемент не викреслюється
<code>size_type size() const</code>	повертає кількість елементів, які є в стеку
<code>bool empty() const</code>	повертає <code>true</code> , якщо стек порожній, інакше – <code>false</code>

Отже, з точки зору класичної організації стеку, відмінність контейнерного класу `stack` фактично лише в тому, що є додаткова операція `top()`, яка дозволяє багаторазово читати копію елемента вершини стеку без його викреслення




З точки зору контейнера, поданого раніше, зображення структури стеку і його методів виглядає так (повернуто на 90 градусів за годинниковою стрілкою):



## Приклади задач на використання стеку

*Задача 1. Текст математичної формули записаний в лінійній формі з використанням круглих дужок. Круглі дужки з точки зору алгебраїчних правил записані коректно. Надрукувати попарно позиції відкриваючих і відповідних їм закриваючих круглих дужок.*

Алгоритм. Розташування відкриваючих і закриваючих дужок відповідає принципу роботи стеку, а саме: перша закриваюча дужка співставляється з останньою відкриваючою:

..... (..... (..... (.....) ..... (.....) .....) ..... 


Тому номери позицій відкриваючих дужок записуємо в стек, а якщо зустрічається закриваюча – читаємо з стеку останню записану позицію. Це буде відповідна пара. Прочитану останню позицію викреслюємо з стеку. Інші літери до уваги не приймаємо.

```

#include <iostream>
#include <string>
#include <stack> // клас stack
using namespace std;

void main()
{
    string formula( "a*(2-((c+d)*5*(m-t)+4k)+8/y)+o" );
    stack<int> bracket; // позиції відкриваючих дужок
    // переглядаємо формулу по одній літері
    int p, z;
    for(p=0; p<formula.length(); p++)
    { if( formula[p] == '(' ) bracket.push(p);
      else if( formula[p] == ')' )
        { z = bracket.top(); bracket.pop();
          cout << z+1 << " " << p+1 << endl;
        }
    }
    system("pause"); // затримати вікно консолі
}

```



7	11
15	19
6	23
3	28



*Задача 2. У файлі надрукована послідовність додатніх і від'ємних чисел в довільному порядку. Послідовність закінчується нулем. Надрукувати окремими рядками додатні і від'ємні числа. Умова: файл можна читати лише один раз.*

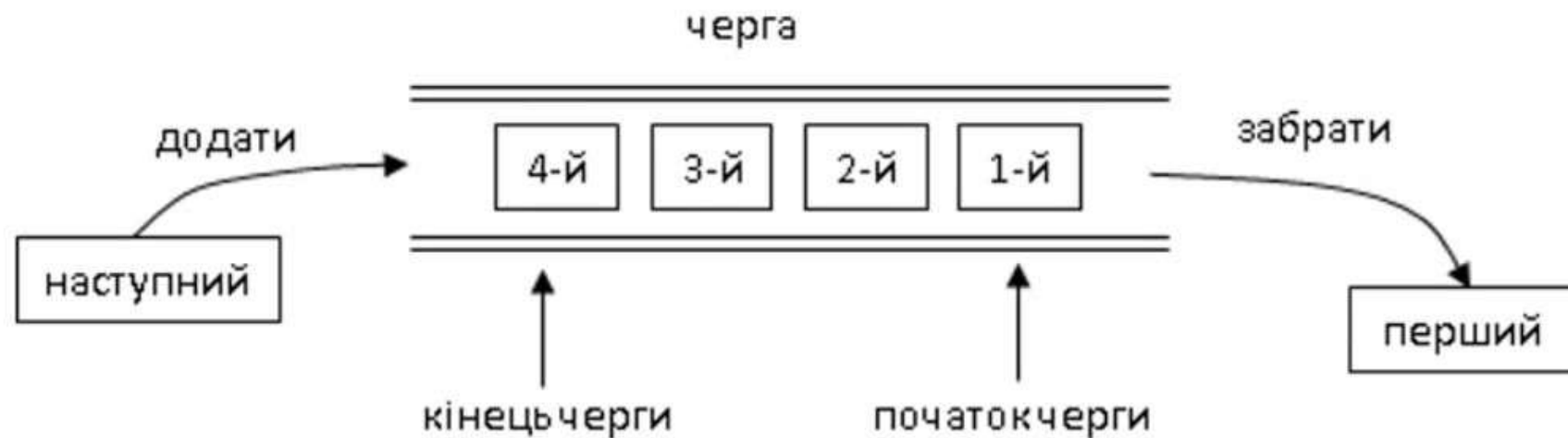
Алгоритм. Кількість чисел у файлі наперед не відома. Тому можна прочитані числа ділити у два стеки: в один – додатні числа, в другий – від'ємні. Після вичерпання файлового потоку з кожного стеку читати всі числа і друкувати. Оскільки ми використовуємо стеки, то порядок друкування буде обернений до початкового.

Схема програми:

```
// . . . . .
stack<int> splus, sminus; // два стеки – початково порожні
// . . . . .
while ( не кінець файла )
{
    // читати чергове число x
    if(x>0) splus.push(x); // якщо додатне
    if(x<0) sminus.push(x); // якщо від'ємне
}
// друкуємо стек з додатніми числами
while( ! splus.empty() )
{ x=splus.top(); // прочитати з вершини стеку ( без викреслення )
  cout << x << " "; // надрукувати
  splus.pop(); // викреслити елемент з вершини стеку
}
// . . . . . аналогічно – для стеку від'ємних
```

# Принцип функціонування черги

Черга – це структура даних, яка функціонує за принципом FIFO: «першим зайшов – першим вийшов». Чергу можна зобразити як коробку в формі коридора, в який з одного боку докладають предмети, а вибирають з протилежного боку:



Черга працює за правилами, подібними до стеку, лише операції виконують з обидвох боків. 1) Початково черга порожня. 2) Черга працює лише в термінах операцій «додати» і «забрати». 3) Кожен наступний елемент можна додати до черги лише в кінець. 4) Забрати з черги можна лише найперший доданий елемент з початку черги, тоді початком черги стає другий. 5) Початок і кінець черги автоматично зсувається при кожній операції додавання/вибирання. 6) Вибрати з порожньої черги неможливо. З точки зору програмування це є помилка виконання програми. 7) Максимальний розмір черги в принципі не обмежений, хоча з точки зору програмної реалізації таке обмеження може бути, тоді може виникнути помилка «вихід за межі ділянки пам'яті».





## Клас черги бібліотеки STL

Такий клас позначають словом `queue`. Для використання черг треба включити в програму директиву `#include <queue>`.

Для оголошення черги треба записати тип її елементів:

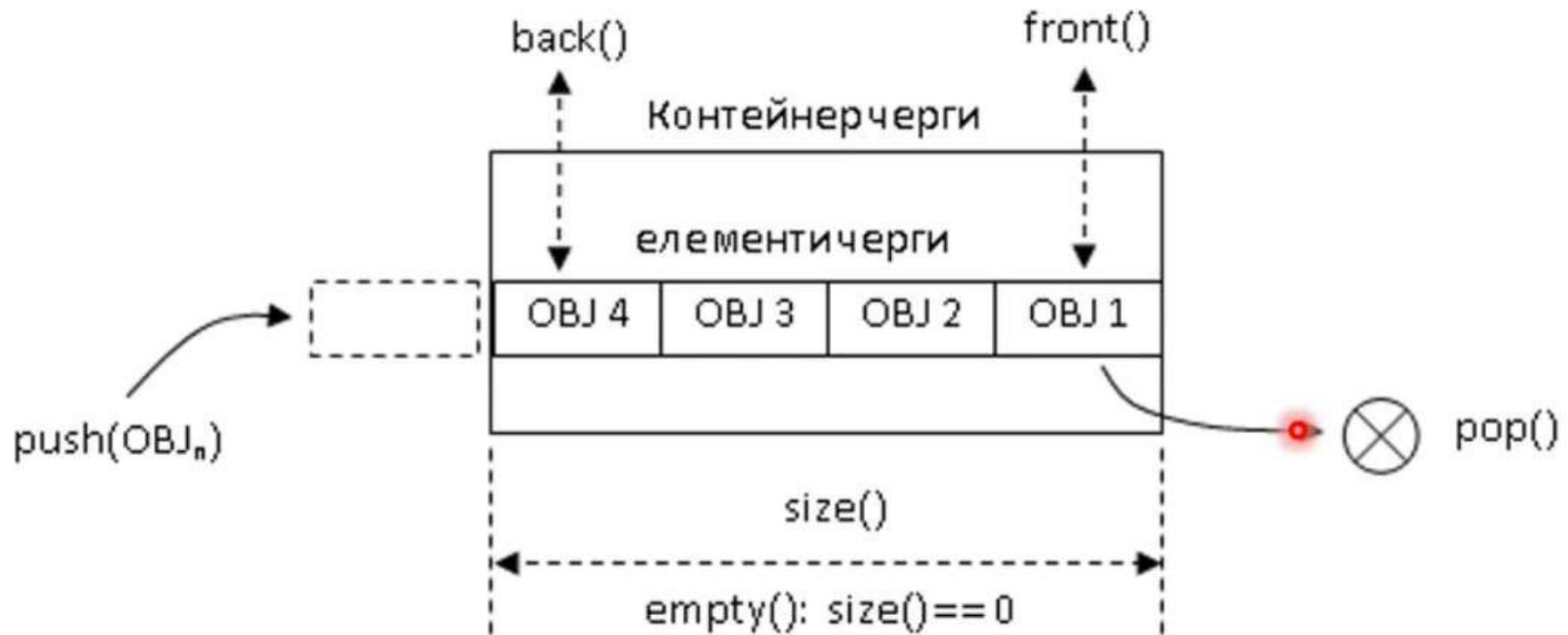
```
struct anydata { /* . . . . . */ } ; // . . . . .  
queue<int> aqueue; // початково черга завжди пуста  
queue<char> bqueue;  
queue<anydata> cqueue;
```

Клас `queue` має такі методи:

<code>void push(...)</code>	додати елемент в кінець черги
<code>void pop()</code>	викреслити елемент з початку черги
<code>const value_type &amp;back()</code> <code>const</code> <code>value_type &amp;back()</code>	посилання на останній елемент черги – двох типів: «лише читати», «читати і писати»; елемент з черги не викреслюється
<code>const value_type &amp;front()</code> <code>const</code> <code>value_type &amp;front()</code>	посилання на початок черги – двох типів: «лише читати», «читати і писати»; елемент з черги не викреслюється
<code>size_type size() const</code>	повертає кількість елементів, які є в черзі
<code>bool empty() const</code>	повертає <code>true</code> , якщо черга порожня, інакше – <code>false</code>

Зауважимо, що крім стандартних операцій `push()`, `pop()` є додаткові операції `back()` і `front()`, які дозволяють багатократно читати (і навіть редагувати) елементи кінця і початку черги без їх викреслення.

З точки зору контейнера, поданого на попередній лекції, зображення структури черги та її методів виглядає так:





## Приклад задачі на використання черги

*Задача. База плодоягідної продукції приймає на тимчасове зберігання партії ягід і фруктів. Відвантаження продукції виконують в порядку її поступлення на базу (холодильники не люблять тривалого зберігання продукції ☺). Змодельувати рух товарів на базі, використовуючи структуру черги і функції «поступлення на базу», «відвантаження з бази».*

Схема програмування. Оголосимо чергу як глобальний об'єкт. Елементами черги будуть пари значень «назва товару, кількість». Якщо черга глобальна – до неї є простий доступ всіх функцій програми.

Загальний список всіх поступлень оголосимо для простоти як масив об'єктів-елементів.

Функція «поступлення на базу» буде записувати в чергу певну кількість елементів загального списку, перевіряючи залишкову кількість. Кількість буде параметром функції.

Функція «відвантаження з бази» буде викреслювати певну кількість елементів з черги, перевіряючи їх наявність. Кількість так само буде параметром функції.

Протокол всіх виконаних операцій записуємо в файл реєстрації, який можна переглянути після виконання програми.



## Фрагменти програмної реалізації:

```
. . . . .  
// елементи черги - пари значень (товар, кількість)  
struct consignment  
{  
    string name;    int amount;  
    // назва товару і його кількість в деяких одиницях  
    consignment(string n, int a) { name=n; amount=a; }  
    consignment() { name="немає"; amount=0; }  
    friend ostream & operator<< (ostream & os, const consignment & ob)  
    { // друкувати елемент черги  
        os << ob.name << ": " << ob.amount ;  
        return os;  
    }  
};  
. . . . .
```

```
// загальний список всіх можливих поступлень від постачальників
const int limit = 10;
consignment commonlist[limit] = {
    consignment("яблука", 160),
    consignment("сливи", 75),
    consignment("вишня заморожена", 32),
    consignment("смородина заморожена", 18),
    consignment("груші", 135),
    consignment("малина заморожена", 24),
    consignment("агрус консервований", 12),
    consignment("черешня консервована", 31),
    consignment("виноград", 95),
    consignment("горіх грецький", 6) } ;

int fromlist = -1; // лічильник вибору з списку:
                  // початкове значення - перед 1-м елементом

// черга руху товарів на базі
queue<consignment> november2016;

// файл реєстрації виконаних операцій
ofstream magazine;
```

```

// функція «поступлення на базу»
void accept(int n)  // n - кількість поступлень на базу
{
    magazine << "Запит до постачальників партій продукції: " << n ;
    // operation - фактична кількість операцій: n, якщо такі є,
    //    або не більше, ніж залишилось в постачальників
    int operation = fromlist + n < limit ? n : limit-1 - fromlist ;
    magazine << "; поступило на базу: " << operation << endl;
    for( int k=1; k<=operation; k++)
    {
        fromlist++; november2016.push(commomlist[fromlist]);
        // додати до черги
        magazine << commomlist[fromlist] << ";  " ;
    }
    magazine << endl;
    magazine << "на базі (в черзі) є партій продукції: " <<
        november2016.size() << " \n" ;
    if(fromlist >= limit-1)
        magazine << "Постачань на цей період більше не буде !\n" ;
    magazine << "-----\n";
} // void accept(int n)

```



```

// функція «відвантаження з бази»
void order(int n)
{ // n - кількість замовлених партій на відвантаження
    magazine <<
        "Запит від замовників на постачання партій продукції: "
        << n ;
    // operation - фактична кількість операцій: n, якщо такі є,
    // або не більше, ніж є на базі
    int operation =
        n <= november2016.size() ? n : november2016.size() ;
    magazine << "; виконано замовлень: " << operation << endl;
    for( int k=1; k<=operation; k++)
    {
        magazine << november2016.front() << "; " ; // надрукувати
        november2016.pop(); // викреслити з черги
    }
    magazine << endl;
    magazine << "на базі (в черзі) залишилось партій продукції: "
        << november2016.size() << " \n" ;
    if( november2016.empty() )
        magazine << "замовлення на цей період"
            << " тимчасово не приймаємо !\n" ;
    magazine << "-----\n";
} // void order(int n)

```

```
void main() // адміністратор бази
{
    magazine.open("reg.txt");
        // файл реєстрації виконаних операцій
    // поперемінні операції "поступлення" і "відвантаження"
    accept(2);
    accept(6);
    order(5);
    accept(2);
    order(3);
    accept(3);
    order(4);
    order(3);

    magazine.close(); // закрити файл реєстрації
    system("pause");
}
```

accept – поступлення

order – відвантаження

## Результат – у файлі реєстрації reg.txt

Запит до постачальників партій продукції: 2; поступило на базу: 2  
яблука: 160; сливи: 75;  
на базі (в черзі) є партій продукції: 2

-----  
Запит до постачальників партій продукції: 6; поступило на базу: 6  
вишня заморожена: 32; смородина заморожена: 18; груші: 135;  
малина заморожена: 24; агрус консервований: 12;  
черешня консервована: 31;  
на базі (в черзі) є партій продукції: 8

-----  
Запит від замовників на відвантаження партій продукції: 5;  
виконано замовлень: 5  
яблука: 160; сливи: 75; вишня заморожена: 32;  
смородина заморожена: 18; груші: 135;  
на базі (в черзі) залишилось партій продукції: 3

-----  
Запит до постачальників партій продукції: 2; поступило на базу: 2  
виноград: 95; горіх грецький: 6;  
на базі (в черзі) є партій продукції: 5  
Постачань на цей період більше не буде !



Запит від замовників на відвантаження партій продукції: 3;

виконано замовлень: 3

малина заморожена: 24; агрус консервований: 12;

черешня консервована: 31;

на базі (в черзі) залишилось партій продукції: 2

-----  
Запит до постачальників партій продукції: 3; поступило на базу: 0

на базі (в черзі) є партій продукції: 2

Постачань на цей період більше не буде !

-----  
Запит від замовників на відвантаження партій продукції: 4;

виконано замовлень: 2

виноград: 95; горіх грецький: 6;

на базі (в черзі) залишилось партій продукції: 0

Замовлення на цей період тимчасово не приймаємо !

-----  
Запит від замовників на відвантаження партій продукції: 3;

виконано замовлень: 0

на базі (в черзі) залишилось партій продукції: 0

Замовлення на цей період тимчасово не приймаємо !

-----

