Звіт

Автор: Момот Р. КІТ-119а

Дата: 17.04.2020

ЛАБОРАТОРНА РОБОТА № 7. АЛГОРИТМИ ОБСЛУГОВУВАННЯ ЧЕРГ

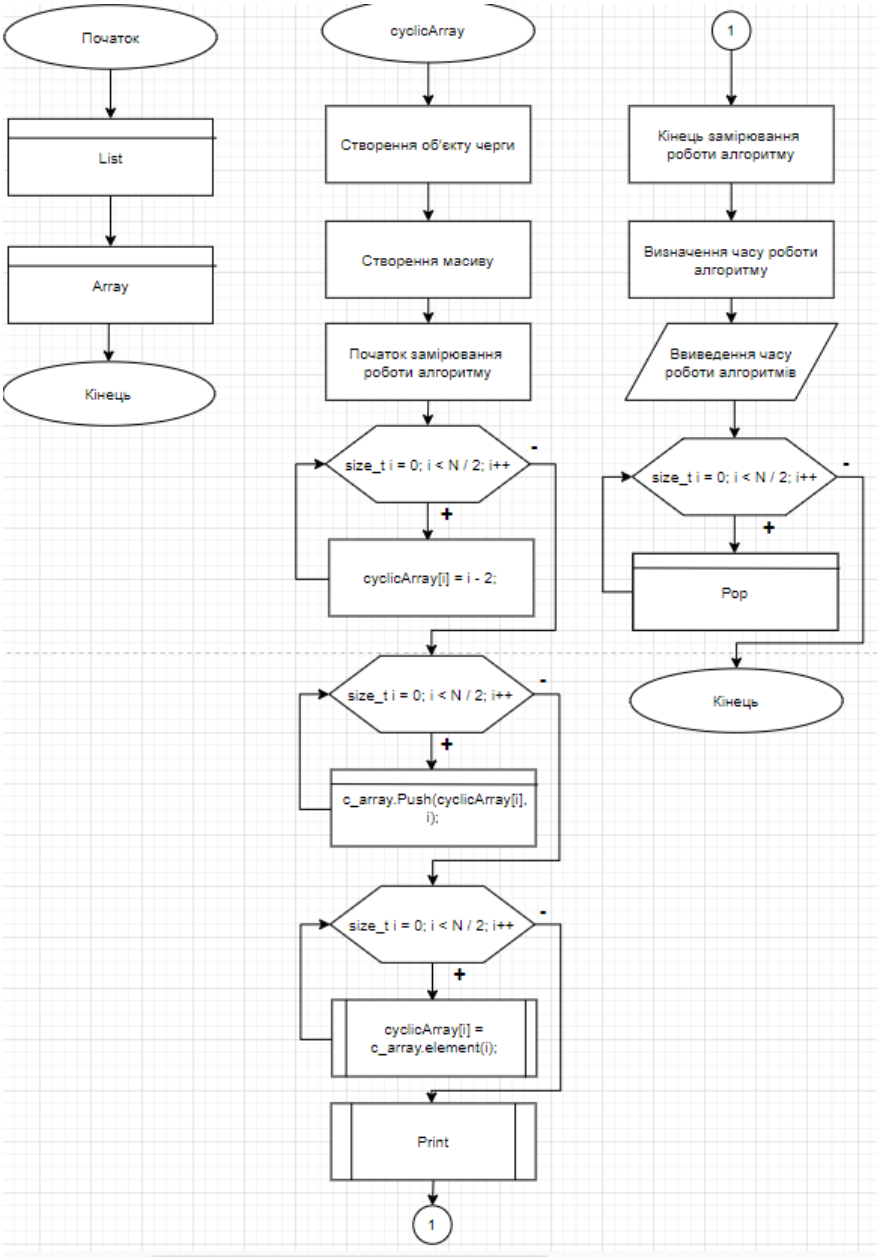
Мета: набути практичного досвіду та закріпити знання про пред-ставлення стека, дека, приоритетної черги та дисциплінах їх обслуговування.

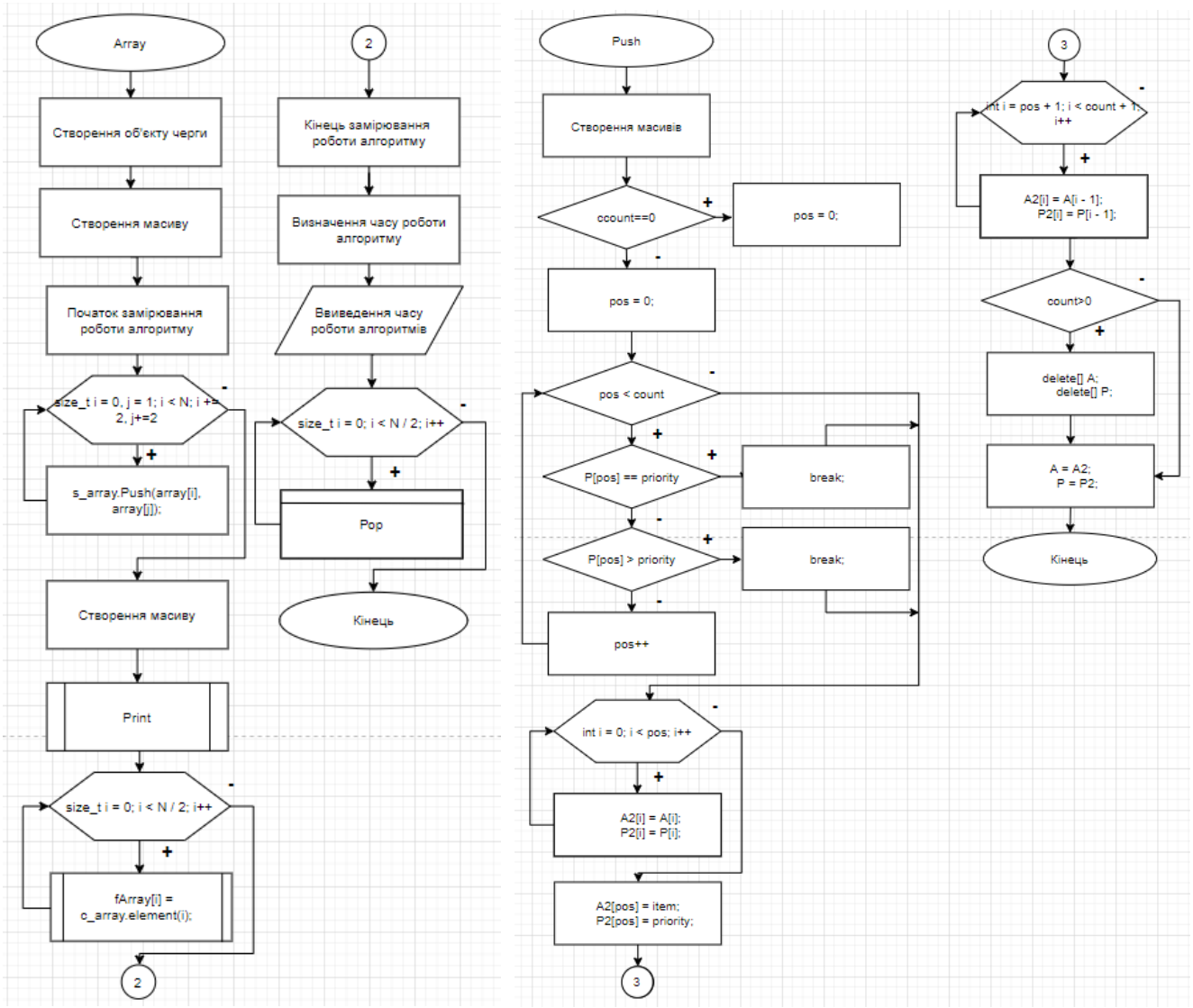
Індивідуальне завдання

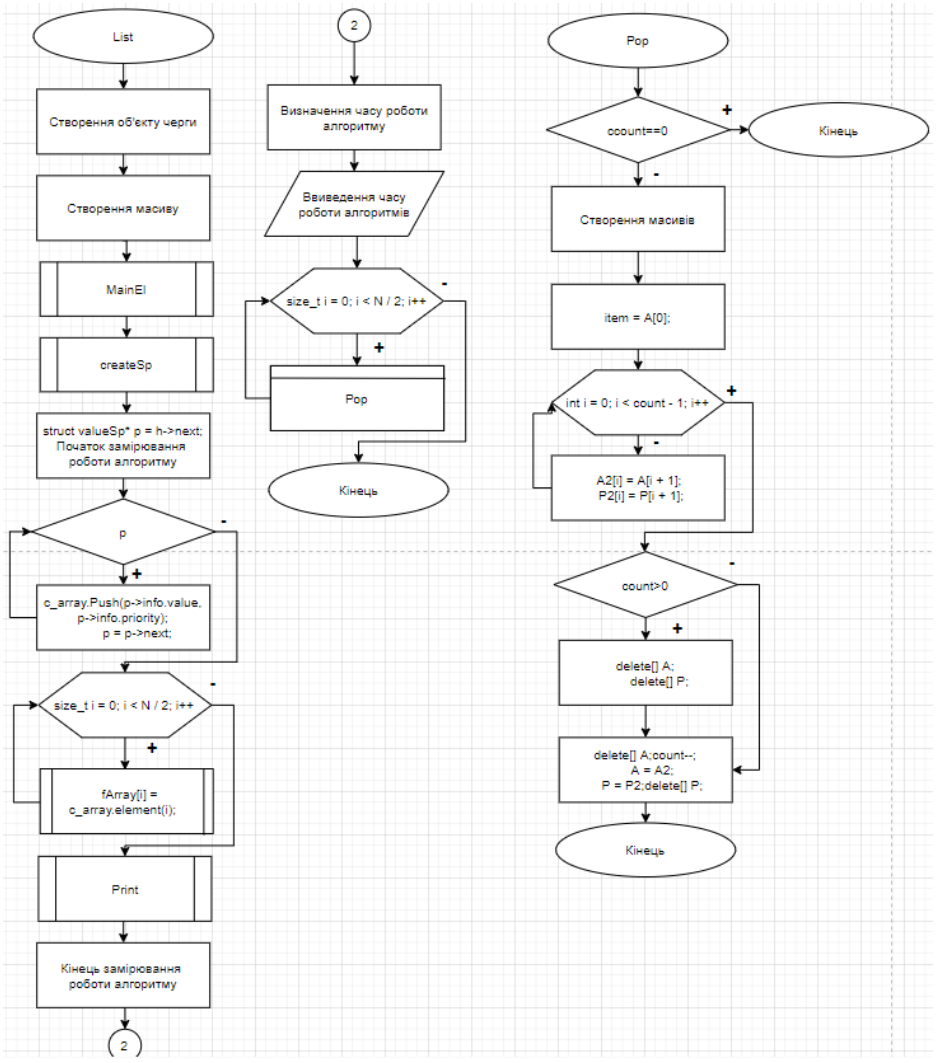
Розробити функції, що забезпечують запис та читання запитів з приоритетної черги, стека або дека. В кожному завданні для організації вказаної черги використати дві структури. Перевірити працездатність розроблених функцій. Послідовність виконання операцій запису та читання обирати випадково. Порівняти результати роботи, зробити висновки.

Приоритетна черга. Постановка запитів в чергу виконується по приоритету, зняття – підряд із старших адрес (кінця черги). Черга організована на масиві та на списку. Приоритет: ***min*** значення числового параметра; при співпаданні параметрів – *LIFO*.

Блок-схема алгоритму програми




Array
Створення об'єкту черги
Створення масиву
Початок заміркування роботи алгоритму
size_t i = 0, j = 1; i < N; i += 2, j+=2
s_array.Push(array[i], array[j]);
Створення масиву
Print
size_t i = 0; i < N / 2; i++
fArray[i] = c_array.element(i);
2
2
Кінець заміркування роботи алгоритму
Визначення часу роботи алгоритму
Ввиведення часу роботи алгоритмів
size_t i = 0; i < N / 2; i++
Pop
Кінець
Push
Створення масивів
ccount==0
pos = 0;
pos = 0;
pos < count
P[pos] == priority
break;
P[pos] > priority
break;
pos++
int i = 0; i < pos; i++
A2[i] = A[i];
P2[i] = P[i];
A2[pos] = item;
P2[pos] = priority;
3
3
int i = pos + 1; i < count + 1; i++
A2[i] = A[i - 1];
P2[i] = P[i - 1];
count>0
delete[] A;
delete[] P;
A = A2;
P = P2;
Кінець


List
Створення об'єкту черги
Створення масиву
MainEl
createSp
struct valueSp* p = h->next;
Початок замірювання
роботи алгоритму
p
c_array.Push(p->info.value,
p->info.priority);
p = p->next;
size_t i = 0; i < N / 2; i++
fArray[i] =
c_array.element(i);
Print
Кінець замірювання
роботи алгоритму
2
2
Визначення часу роботи
алгоритму
Ввиведення часу
роботи алгоритмів
size_t i = 0; i < N / 2; i++
Pop
Кінець
Pop
ccount==0
Кінець
Створення масивів
item = A[0];
int i = 0; i < count - 1; i++
A2[i] = A[i + 1];
P2[i] = P[i + 1];
count>0
delete[] A;
delete[] P;
delete[] A;count--;
A = A2;
P = P2;delete[] P;
Кінець

Текст програми

```
#include <iostream>
#include <chrono>
#include <new>
using namespace std;
#define N 10

struct value {
       int value;
    int priority;
};
struct valueSp {
       value info;
       valueSp* next;
};
struct valueSp* MainEl()
{
    struct valueSp* p = new valueSp;
    p->next = NULL;
    return p;
};

class QueueP
{
private:
    int* A;
    int* P;
    int count;
public:
    QueueP() { count = 0; }

    void Push(int item, int priority);
    int Pop();
    void Clear();
    int Count();
    void Print();
    int element(int i);

    ~QueueP()
    {
        if (count > 0)
        {
            delete[] A;
            delete[] P;
        }
    }
};

struct valueSp* MainEl();
void createSp(struct valueSp*, int*);
void freeSp(struct valueSp*);
void List();
void Array();

void main()
{
    List();
    Array();
}

int QueueP::Count()
{
    return count;
}
```

```
void QueueP::Clear()
{
    if (count > 0)
    {
        delete[] A;
        delete[] P;
        count = 0;
    }
}
void QueueP::Push(int item, int priority)
{
    int* A2;
    int* P2;
    try {
        A2 = new int[count + 1];
        P2 = new int[count + 1];
    }
    catch (bad_alloc e)
    {
        cout << e.what() << endl;
        return;
    }
    int pos;
    if (count == 0)
        pos = 0;
    else
    {
        pos = 0;
        while (pos < count)
        {
            if (P[pos] == priority)
                break;
            else if (P[pos] > priority)
                break;
            pos++;
        }
    }
    for (int i = 0; i < pos; i++)
    {
        A2[i] = A[i];
        P2[i] = P[i];
    }
    A2[pos] = item;
    P2[pos] = priority;
    for (int i = pos + 1; i < count + 1; i++)
    {
        A2[i] = A[i - 1];
        P2[i] = P[i - 1];
    }
    if (count > 0)
    {
        delete[] A;
        delete[] P;
    }
    A = A2;
    P = P2;
    count++;
}
int QueueP::Pop()
{
    if (count == 0)
        return 0;
    int* A2;
    int* P2;
    try {
        A2 = new int[count - 1];
```

```
        P2 = new int[count - 1];
    }
    catch (bad_alloc e)
    {
        cout << e.what() << endl;
        return 0;
    }
    int item;
    item = A[0];
    for (int i = 0; i < count - 1; i++)
    {
        A2[i] = A[i];
        P2[i] = P[i];
    }
    if (count > 0)
    {
        delete[] A;
        delete[] P;
    }
    count--;
    A = A2;
    P = P2;
    return item;
}
void QueueP::Print()
{
    cout << "Objects and their priority" << endl;
    for (int i = 0; i < count; i++)
        cout << A[i] << ":" << P[i] << "\t" << endl;
    cout << endl;
    cout << "---------------" << endl;
}
int QueueP::element(int i)
{
    return A[i];
}

void List()
{
    QueueP c_array;
    int arr[N] = { -2, 1, 0, 2, -555, 0, 15, 3, 57, 4 };
    int* fArray = new int[N / 2];
    struct valueSp* h = MainEl();
    createSp(h, arr);
    struct valueSp* p = h->next;
    auto begin = chrono::steady_clock::now();
    while (p)
    {
        c_array.Push(p->info.value, p->info.priority);
        p = p->next;
    }



    for (size_t i = 0; i < N / 2; i++)
        fArray[i] = c_array.element(i);    //запись
    c_array.Print();
    auto end = chrono::steady_clock::now();
    auto elapsed_ms = chrono::duration_cast<chrono::nanoseconds>(end - begin);
    cout << "Priority queue time of work using list: " << elapsed_ms.count() << "ns" << endl <<
endl << endl;
    for (size_t i = 0; i < N / 2; i++)
        c_array.Pop();
}
void createSp(struct valueSp* h, int* arr)
{
```

```
    struct valueSp* p = h;
    for (size_t i = 0, j = 1; i < N; i += 2, j += 2)
    {
        p->next = new valueSp;
        p = p->next;
        p->info.value = arr[i];
        p->info.priority = arr[j];
    }
    p->next = NULL;
}
void Array()
{
    QueueP s_array;
    int array[N] = { -2, 1, 0, 2, -555, 0, 15, 3, 57, 4 };
    int* tempArray;
    auto begin2 = chrono::steady_clock::now();
    for (size_t i = 0, j = 1; i < N; i += 2, j+=2)
    {
        s_array.Push(array[i], array[j]);
    }
    int* fArray = new int[N / 2];
    s_array.Print();
    for (size_t i = 0; i < N / 2; i++)
        fArray[i] = s_array.element(i);
    auto end2 = chrono::steady_clock::now();
    auto elapsed_ms2 = chrono::duration_cast<chrono::nanoseconds>(end2 - begin2);
    cout << "Priority queue time of work using array: " << elapsed_ms2.count() << "ns" << endl;
    for (size_t i = 0; i < N / 2; i++)
        s_array.Pop();
}
void freeSp(struct valueSp* h)
{
    struct valueSp* p;
    while (h)
    {
        p = h;
        h = h->next;
        delete p;
    }
}
```

Результати роботи програми

```
Objects and their priority
-555:0
-2:1
0:2
15:3
57:4

---------------
Priority queue time of work using list: 4668400ns


Objects and their priority
-555:0
-2:1
0:2
15:3
57:4

---------------
Priority queue time of work using array: 1068900ns
```

Висновок

У результаті роботи програми було розроблено програму, яка забезпечує запис та читання запитів. Були використані клас(масив) та список. Постановка запитів виконується по приорітету, зняття – підряд із старших адрес(кінця черги).

Відповіді на питання

1. Що таке «черга», дайте визначення?

Черга - така структура даних, яка зберігає елементи і забезпечує доступ до них тільки в певному порядку, який визначається їх пріоритетом.

2. Що являє собою стек?

Стек - такий послідовний список зі змінною довжиною, включення і виключення елементів з якого виконуються тільки з одного боку списку, званого вершиною стека. Застосовуються й інші назви стека - магазин, чергу, функціонує за принципом LIFO (Last In-First Out - «останнім прийшов, першим виключається»).

3. Які відомі види черг?

Розрізняють два типи черг: деки (такі черги, в яких пріоритет визначатися часом постановки елементів в чергу) і пріоритетні черги (в яких час не є пріоритетом).

Деки в свою чергу поділяються на:

- Черги LIFO або стеки;
- Черги FIFO або деки з обмеженим входом і виходом;
- Повні деки.

4. На основі яких структур даних може бути організований стек?

Для реалізації черг можуть бути використані статичні або динамічні структури даних.

5. Які операції допустимі для черги?

Над деком допустимі наступні операції:

- Включення елементу справа/зліва;
- Виключення елементу справа/зліва;

Над чергами з пріоритетом (пріоритетна постановка запитів /пріоритетне зняття запитів) допустимі наступні операції:

- Включення запиту у чергу (за пріоритетом, якщо це пріоритетна постановка запитів, якщо ні то у кінець) ;
- Зняття запиту (за пріоритетом, якщо це пріоритетне зняття запитів, якщо ні то з початку);

Спільні операції:

- Визначення розміру черги;
- Очищення черги;

6. Які операції допустимі для стека?

Для **стека** існують такі операції, як:

- Включення нового елемента;
- Виключення елемента з стека;
- Визначення поточного числа елементів в стеку;
- Очищення стека.

7. Який алгоритм читання запитів із стека?

Операція читання елементу із стеку полягає в модифікації покажчика стека (в напрямку, протилежному модифікації при включенні) і вибірці значення, на яке вказує покажчик стека. Після вибірки слот, в якому розміщувався обраний елемент, вважається вільним.

8. Які властивості притаманні черзі?

Черга забезпечує доступ до них тільки в певному порядку, який визначається їх пріоритетом (FIFO, LIFO).

9. Який недолік простої черги? Який спосіб боротьби з ним?

Недолік - фіксований розмір, який не можна змінити в процесі роботи з чергою. Цього можна уникнути, створюючи, в разі необхідності, нові черги в динамічному режимі.

10. Чим відрізняється циклічна черга від простої?

Циклічні черги мають таку ж саму структуру, як і прості черги з однією відмінністю. Ця відмінність полягає в тому, що після заповнення черги запис знов починається з першого елементу черги при умові, що цей елемент уже зчитано. Далі запис продовжується, як і в звичайній черзі при умові, що відповідні елементи черги уже прочитані.

11. Чим відрізняється стек на основі масиву від стека на основі зв’язного списку ?

Реалізація стеку у вигляді зв'язного списку має:

- Потреби в меншому обсязі пам'яті в разі додавання нового елемента. У зв'язного списку додаткова пам'ять для елемента виділяється тільки для цього одного елемента. У масиві спочатку потрібно виділити новий фрагмент (масив) пам'яті для всіх елементів, потім здійснити копіювання старого масиву в новий і тільки тоді звільнити пам'ять, виділену під старий масив. Із зростанням кількості елементів в стеку ця перевага стає більш відчутною;
- Меншу кількість додаткових операцій в разі маніпулювання стеком (додавання нового елемента, видалення елемента).

Реалізація стеку у вигляді масиву має:

- Більш просту реалізацію. Обробка зв’язного списку складніша у реалізації;
- Для доступу до і-го елементу в масиві зручно використовувати доступ за індексом. У зв'язному списку потрібно переглядати весь список від початку до потрібної позиції;

12. Чим відрізняється пріоритетна черга з приоритетним записом від черги з пріоритетним читанням?

- Пріоритетна черга з пріоритетним записом – запит знімається з черги за пріоритетом, запити ставляться в кінець черги;
- Пріоритетна черга з пріоритетним читанням – запит включається у чергу за пріоритетом, запити знімаються з початку черги;

13. Чим відрізняється стек на основі зв'язного списку від власне зв'язного списку?

Стек - це окремий випадок односпрямованого списку, додавання елементів в який і вибірка з якого виконуються з одного кінця, званого вершиною стека. При вибірці елемент виключається з стека.

14. Які області застосування черг і стеків?

Стек є надзвичайно зручною структурою даних для багатьох завдань обчислювальної техніки. Найбільш типовою з таких завдань є забезпечення вкладених викликів процедур.

Черга в програмуванні використовується, як і в реальному житті, коли потрібно зробити якісь дії в порядку їх надходження, виконавши їх послідовно. Прикладом може служити організація подій в Windows. Коли користувач впливає на додаток, то в додатку не викликається відповідна процедура (адже в цей момент додаток може здійснювати інші дії), а йому надсилається повідомлення, що містить інформацію про скоєну дію, це повідомлення ставиться в чергу, і тільки коли будуть оброблені повідомлення, що прийшли раніше, додаток виконає необхідну дію.

15. Що являє собою дек?

Дек - особливий вид черги у вигляді послідовного списку, в якому як включення, так і виключення елементів може здійснюватися з будь-якого з двох кінців списку.

16. Який алгоритм зняття запиту з пріоритетної черги, в яку запити ставляться підряд як приходять?

Алгоритм пріоритетного зняття запиту такої:

- якщо чергу непорожній, знайти за пріоритетом необхідний запит;
- прочитати запит;
- видалити запит з черги, для цього посунути всі елементи черги від наступного до останнього на одну позицію вліво.

17. Який алгоритм постановки запита в пріоритетну чергу, в якій запити знімаються підряд?

Алгоритм пріоритетною постановки запиту в чергу передбачає такі дії:

- якщо черга не повна, знайти відповідно до пріоритету таке місце в черзі, де необхідно поставити запит;
- звільнити місце для запиту, для чого перемістити всі запити від цього місця до кінця черги на один елемент;
- внести запит в чергу.

18. Яка помилка допущена в наведеній функції запису елемента в стек?

```
int StackPush(int elem)
{
   St[--top] = elem;  return 1;
}
```

В функції не має перевірки на стан стеку (заповнений він чи ні).