

Черги



Лекція №9

Дисципліна «Програмування»

2-й семестр



Створення черги

Підхід до програмування на **c** з застосуванням абстрактних типів даних передбачає виконання:

- опису типу в абстрактній узагальненій манері разом з його операціями;
- визначення інтерфейсу у вигляді функцій для представлення нового типу;
- написання коду для реалізації інтерфейсу.

Черга – це список, в якому елементи можуть додаватися тільки в кінці, а видалятися тільки на початку.

Чергу можна порівняти з ланцюжком людей, які стоять один за одним до квиткової каси. Кожна нова людина стає в кінець ланцюжка та залишає її на самому початку (після придбання квитків).



Абстрактне визначення черги

Черга є формою даних типу «першим увійшов – першим вийшов» (**First In First Out – FIFO**), подібною до черги в касу (якщо тільки ніхто не уклиниться в чергу).

Неформальне абстрактне визначення черги:

- | | |
|--------------------------|--|
| Ім'я типу: | Черга |
| Властивість типу: | Може містити упорядковану послідовність елементів |
| Операції типу: | <ol style="list-style-type: none">1) Ініціалізація черги пустим вмістом.2) З'ясування, чи є черга пустою.3) З'ясування, чи є черга повною.4) Визначення кількості елементів у черзі.5) Додавання елемента наприкінці черги.6) Видалення та відновлення елемента на початку черги.7) Спустошення черги. |



Визначення інтерфейсу

Ініціалізація черги передбачає зміну типу **Queue**, тому функція повинна приймати в якості аргументу адресу змінної **Queue**:

```
void InitializeQueue(Queue *pq);
```

З'ясування, чи є черга пустою або повною, передбачає застосування функцій, які повинні повертати істинне або хибне значення:

```
bool QueueIsFull(const Queue *pq);
```

```
bool QueueIsEmpty(const Queue *pq);
```

Для позначення того, що функції не змінюють чергу, краще застосовувати кваліфікатор **const**. Вказівник **pq** посилається на об'єкт даних **Queue**, який не може змінюватися за допомогою **pq**.

Будемо вважати, що файл заголовку **stdbool.h** стандарту **C99** є доступним. Якщо це не так, можна використовувати тип **int** або визначити тип **bool** самостійно.



Визначення інтерфейсу

Повернення кількості елементів у черзі здійснюватиме функція

```
int QueueItemCount(const Queue *pq);
```

Додавання елемента наприкінці черги передбачає ідентифікацію елемента та черги. Значення, що повертається, можна застосовувати для позначення успішності або неуспішності виконання операції:

```
bool EnQueue(Item item, Queue *pq);
```

Видалення елемента передбачає застосування функції, один з можливих прототипів якої має наступний вигляд:

```
bool DeQueue(Item *pitem, Queue *pq);
```

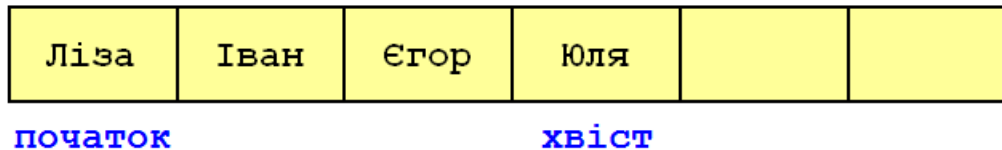
Єдиним аргументом, який повинен бути наданий функції спустошення черги, є адреса черги, що і демонструє наведений нижче прототип:

```
void EmptyTheQueue(Queue *pq);
```

Реалізація черги

Перший спосіб використання масиву в якості черги

В черзі стоять чотири людини



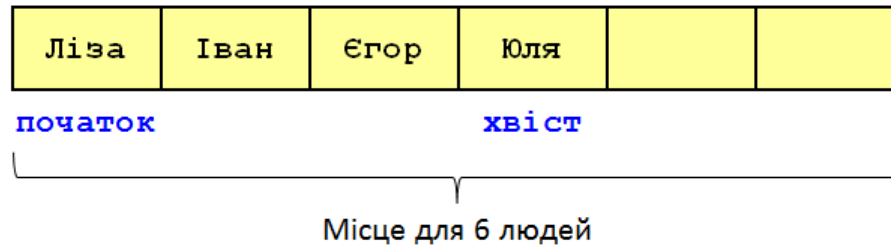
Федір став у чергу, потім Ліза її залишила



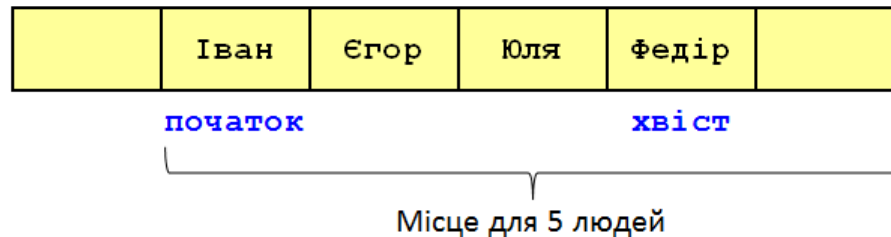
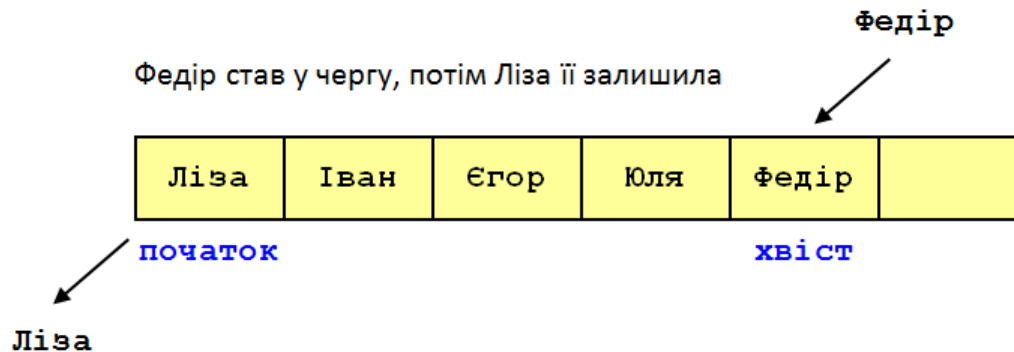
Реалізація черги

Другий спосіб використання масиву в якості черги

В черзі стоять чотири людини

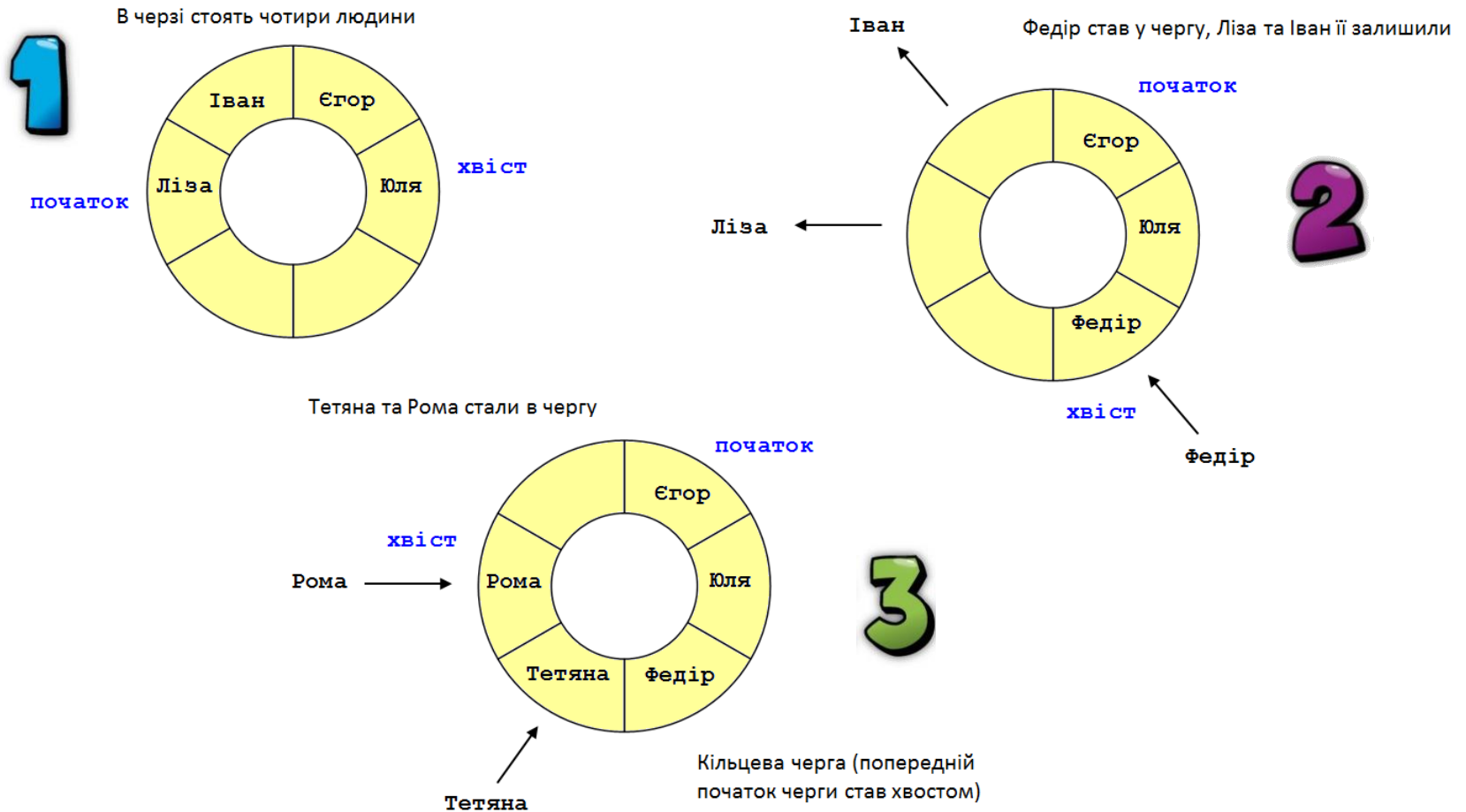


Федір став у чергу, потім Ліза її залишила



Реалізація черги

Третій спосіб використання масиву в якості черги





Реалізація інтерфейсу

```
//=====
// queue.h - файл заголовку для інтерфейсу черги
//=====
#ifndef _QUEUE_H_
#define _QUEUE_H_
#include <stdbool.h>
#define MAXQUEUE 10      // максимальна кількість елементів

typedef int Item;

typedef struct node
{
    Item item;
    struct node *next;
} Node;

typedef struct queue
{
    Node *front;           // вказівник на початок черги
    Node *rear;            // вказівник на хвіст черги
    int items;             // кількість елементів у черзі
} Queue;
```



Реалізація інтерфейсу

```
//=====
// Операція:    ініціалізація черги
// Передумова:  pq вказує на чергу
// Постумова:   черга ініціалізована пустим вмістом
//=====
void InitializeQueue(Queue *pq);

//=====
// Операція:    перевірка, чи заповнена черга
// Передумова:  pq вказує на чергу, що була ініціалізована раніше
// Постумова:   повертає True, якщо черга заповнена,
//              і False у протилежному випадку
//=====
bool QueueIsFull(const Queue *pq);

//=====
// Операція:    перевірка, чи є черга пустою
// Передумова:  pq вказує на чергу, що була ініціалізована раніше
// Постумова:   повертає True, якщо черга пуста,
//              і False у протилежному випадку
//=====
bool QueueIsEmpty(const Queue *pq);
```



Реалізація інтерфейсу

```
//=====
// Операція: визначення кількості елементів у черзі
// Передумова: pq вказує на чергу, що була ініціалізована раніше
// Постумова: повертає кількість елементів у черзі
//=====
int QueueItemCount(const Queue *pq);

//=====
// Операція: додавання елемента наприкінці черги
// Передумова: pq вказує на чергу, що була ініціалізована раніше
// елемент повинен бути поміщений в кінець черги
// Постумова: якщо черга не є пустою, елемент поміщається
// наприкінці черги і функція повертає True;
// у протилежному випадку черга залишається незмінною,
// а функція повертає False
//=====
bool EnQueue(Item item, Queue *pq);
```



Реалізація інтерфейсу

```
//=====
// Операція: видалення елемента на початку черги
// Передумова: pq вказує на чергу, що була ініціалізована раніше
// Постумова: якщо черга не є пустою, елемент на початку черги
//             копіюється в *pitem і видаляється з черги,
//             сама функція повертає True;
//             якщо операція спустошує чергу, то черга
//             перевстановлюється в пустий стан.
//             Якщо черга є пустою з самого початку, вона
//             залишається незмінною, сама функція повертає False
//=====
bool DeQueue(Item *pitem, Queue *pq);

//=====
// Операція: спустошення черги
// Передумова: pq вказує на чергу, що була ініціалізована раніше
// Постумова: черга стає пустою
//=====
void EmptyTheQueue(Queue *pq);
```



Реалізація інтерфейсу

```
//=====
// Операція:   вивід вмісту черги на екран
// Передумова: pq вказує на чергу, що була ініціалізована раніше
// Постумова:  черга виведена на екран
//=====
void PrintQueue(Queue *pq);
#endif
```



Реалізація функцій

```
void InitializeQueue (Queue *pq)
{
    pq->front = pq->rear = NULL;
    pq->items = 0;
}

bool QueueIsFull (const Queue *pq)
{
    return pq->items = MAXQUEUE;
}

bool QueueIsEmpty (const Queue *pq)
{
    return pq->items = 0;
}

int QueueItemCount (const Queue *pq)
{
    return pq->items;
}
```



Додавання елемента в чергу

Додавання елемента в чергу передбачає:

- 1) Створення нового вузла.
- 2) Копіювання елемента в цей вузол.
- 3) Встановлення вказівника **next** цього вузла в **NULL**, що ідентифікує його як останній у черзі.
- 4) Встановлення вказівника **next** поточного кінцевого вузла таким чином, щоб він посилався на новий вузол, зв'язуючи його з чергою.
- 5) Встановлення вказівника **rear** для посилення на новий вузол з метою спрощення пошуку останнього вузла.
- 6) Збільшення на **1** лічильника елементів черги.



Реалізація функції додавання

```
bool EnQueue(Item item, Queue *pq)
{
    Node *pnew;

    if (QueueIsFull(pq))
        return false;
    pnew = (Node *) malloc(sizeof(Node));
    if (pnew == NULL)
    {
        fprintf(stderr, "Не вдається виділити пам'ять!\n");
        exit(1);
    }
    CopyToNode(item, pnew);
    pnew->next = NULL;
    if (QueueIsEmpty(pq))
        pq->front = pnew;           // елемент поміщається на початок
    else                           // черги
        pq->rear->next = pnew;      // зв'язування з кінцем черги
}
```




Реалізація функції додавання

```
pq->rear = pnew;      // запис місця розташування кінця
                        // черги
pq->items++;           // збільшення на 1 кількості
                        // елементів у черзі
return true;
}
```

Функція `CopyToNode()` – це статична функція, яка виконує копіювання елемента у вузол:

```
static void CopyToNode(Item item, Node *pn)
{
    pn->item = item;
}
```



Видалення елемента

Видалення елемента на початку черги потребує:

- 1) Копіювання елемента до визначеної змінної.
- 2) Звільнення пам'яті, яка використовувалася вузлом, що видаляється.
- 3) Переустановлення вказівника на початок черги, щоб він посилався на наступний елемент у черзі.
- 4) Встановлення вказівників на початок і на кінець черги в **NULL**, якщо видалено останній елемент.
- 5) Зменшення на **1** лічильника елементів черги.



Реалізація функції видалення

```
bool DeQueue(Item *pitem, Queue *pq)
{
    Node *pt;

    if (QueueIsEmpty(pq) )
        return false;

    CopyToItem(pq->front, pitem);

    pt = pq->front;
    pq->front = pq->front->next;

    free(pt);

    pq->items--;

    if (pq->items == 0)
        pq->rear = NULL;

    return true;
}
```



Спустошення черги

Для спустошення черги можна використовувати функцію `DeQueue()`. Для цього достатньо викликати її в циклі до тих пір, поки черга не стане пустою:

```
void EmptyTheQueue (Queue *pq)
{
    Item dummy;
    while (!QueueIsEmpty (pq) )
        DeQueue (&dummy, pq);
}
```



Реалізація черги

```
//=====
// queue.c - файл реалізації черги (Queue)
//=====
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

//=====
// локальні функції
//=====

static void CopyToNode(Item item, Node *pn);
static void CopyToItem(Node *pn, Item *pi);

void InitializeQueue(Queue *pq)
{
    pq->front = pq->rear = NULL;
    pq->items = 0;
}
```



Реалізація черги

```
bool QueueIsFull(const Queue *pq)
{
    return pq->items == MAXQUEUE;
}
```

```
bool QueueIsEmpty(const Queue *pq)
{
    return pq->items == 0;
}
```

```
int QueueItemCount(const Queue *pq)
{
    return pq->items;
}
```



Реалізація черги

```
bool EnQueue(Item item, Queue *pq)
{
    Node *pnew;
    if (QueueIsFull(pq)) return false;
    pnew = (Node *) malloc(sizeof(Node));
    if (pnew == NULL) {
        fprintf(stderr, "Не вдається виділити пам'ять!\n");
        exit(1);
    }
    CopyToNode(item, pnew);
    pnew->next = NULL;
    if (QueueIsEmpty(pq))
        pq->front = pnew; // елемент поміщається на початок черги
    else
        pq->rear->next = pnew; // зв'язування з кінцем черги
    pq->rear = pnew; // запис місця розташування кінця черги
    pq->items++; // збільшення кількості елементів на 1
    return true;
}
```



Реалізація черги

```
bool DeQueue(Item *pitem, Queue *pq)
{
    Node *pt;

    if (QueueIsEmpty(pq))
        return false;

    CopyToItem(pq->front, pitem);

    pt = pq->front;
    pq->front = pq->front->next;

    free(pt);

    pq->items--;
    if (pq->items == 0)
        pq->rear = NULL;

    return true;
}
```




Реалізація черги

```
void PrintQueue(Queue *pq)
{
    Node *pt;

    pt = pq->front;
    printf("Черга:");
    for(int i = 0; i < pq->items; i++)
    {
        printf("%5d", pt->item);
        pt = pt->next;
    }
    printf("\n");
}
```



Реалізація черги

```
// спустошення черги
void EmptyTheQueue (Queue *pq)
{
    Item dummy;
    while (!QueueIsEmpty (pq) )
        DeQueue (&dummy, pq);
}

// локальні функції
static void CopyToNode (Item item, Node *pn)
{
    pn->item = item;
}

static void CopyToItem (Node *pn, Item *pi)
{
    *pi = pn->item;
}
```



Тестування черги

```
//=====
// main.c - тестування інтерфейсу черги
//=====
#include <stdio.h>
#include <windows.h>
#include "queue.h"

// визначення Queue, Item

int main(void)
{
    Queue line;
    Item temp;
    char ch;

    SetConsoleOutputCP(1251);

    InitializeQueue(&line);
    puts("=====");
    puts("Тестування інтерфейсу черги (Queue)");
    puts("=====");
```



Тестування черги

```
puts("Введіть символ:");
puts("    1 - додати значення до черги");
puts("    2 - видалити значення з черги");
puts("    3 - вихід з програми");
puts("=====");
while((ch = getchar()) != '3')
{
    if(ch != '1' && ch != '2') // ігнорувати інші дані
        continue;
    if(ch == '1')
    {
        printf("Введіть ціле число для додавання: ");
        scanf("%d", &temp);
        if(!QueueIsFull(&line))
        {
            printf("Поміщаємо %d до черги\n", temp);
            EnQueue(temp, &line);
            PrintQueue(&line);
        }
    }
}
```



Тестування черги

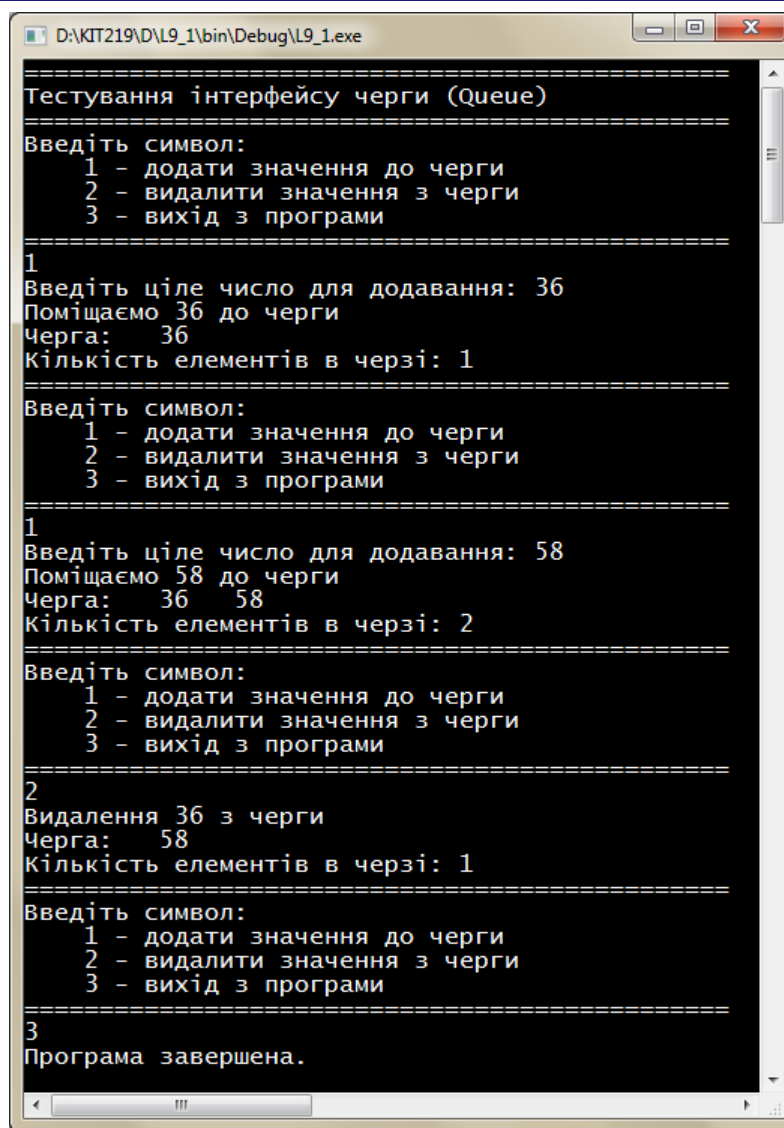
```
        else
            puts ("Черга заповнена!");
    }
    else
    {
        if (QueueIsEmpty (&line))
            puts ("Елементи для видалення відсутні!");
        else
        {
            DeQueue (&temp, &line);
            printf ("Видалення %d з черги\n", temp);
            PrintQueue (&line);
        }
    }
    printf ("Кількість елементів в черзі: %d\n",
           QueueItemCount (&line));
```



Тестування черги

```
puts ("=====");  
puts ("Введіть символ:");  
puts ("    1 - додати значення до черги");  
puts ("    2 - видалити значення з черги");  
puts ("    3 - вихід з програми");  
puts ("=====");  
}  
EmptyTheQueue (&line);  
puts ("Програма завершена.");  
return 0;  
}
```

Тестування черги



```
D:\KIT219\D\L9_1\bin\Debug\L9_1.exe

=====
Тестування інтерфейсу черги (Queue)
=====
Введіть символ:
  1 - додати значення до черги
  2 - видалити значення з черги
  3 - вихід з програми
=====
1
Введіть ціле число для додавання: 36
Поміщаємо 36 до черги
Черга: 36
Кількість елементів в черзі: 1
=====
Введіть символ:
  1 - додати значення до черги
  2 - видалити значення з черги
  3 - вихід з програми
=====
1
Введіть ціле число для додавання: 58
Поміщаємо 58 до черги
Черга: 36 58
Кількість елементів в черзі: 2
=====
Введіть символ:
  1 - додати значення до черги
  2 - видалити значення з черги
  3 - вихід з програми
=====
2
Видалення 36 з черги
Черга: 58
Кількість елементів в черзі: 1
=====
Введіть символ:
  1 - додати значення до черги
  2 - видалити значення з черги
  3 - вихід з програми
=====
3
Програма завершена.
```



Моделювання реальної черги

```
//=====
// main.c - тестування інтерфейсу консультатійного кіоску
//=====
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>           // для rand() і srand()
#include <time.h>             // для time()
#include "queue.h"           // змініть визначення типу Item
#define MIN_PER_HR 60.0

bool newcustomer(double x); // чи є новий клієнт?
Item customertime(long when); // встановлення параметрів
                                // клієнта

int main(void)
{
    Queue line;
    Item temp;                // дані про нового клієнта
    int hours;                // тривалість моделювання
                                // в годинах
```




Моделювання реальної черги

```
int perhour;           // середня кількість клієнтів,  
                        // які надходять за годину  
long cycle, cyclelimit; // лічильник і граничне значення  
                        // циклу  
long turnaways = 0;    // кількість відмов через  
                        // переповнення черги  
long customers = 0;     // кількість клієнтів, які  
                        // приєдналися до черги  
long served = 0;        // кількість клієнтів, які були  
                        // обслуговані за час моделювання  
long sum_line = 0;      // довжина черги, що накопичується  
int wait_time = 0;      // час до звільнення консультанта  
double min_per_cust;    // середній час між прибуттям  
                        // клієнтів  
long line_wait = 0;     // час у черзі, що накопичується  
  
SetConsoleOutputCP(1251);  
InitializeQueue(&line);  
srand((unsigned int)time(0));
```



Моделювання реальної черги

```
printf("=====\n");
printf("Консультаційний кіоск\n");
printf("=====\n");
printf("Введіть тривалість моделювання в годинах: ");
scanf("%d", &hours);
cyclelimit = MIN_PER_HR * hours;
printf("Введіть середню кількість клієнтів, "
       "які надходять за годину: ");
scanf("%d", &perhour);
min_per_cust = MIN_PER_HR / perhour;
printf("=====\n");
for(cycle = 0; cycle < cyclelimit; cycle++)
{
    if(newcustomer(min_per_cust))
    {
        if(QueueIsFull(&line))
            turnaways++;
        else
        {
```



Моделювання реальної черги

```
        customers++;
        temp = customertime(cycle);
        EnQueue(temp, &line);
    }
}
if(wait_time <= 0 && !QueueIsEmpty(&line))
{
    DeQueue(&temp, &line);
    wait_time = temp.processtime;
    line_wait += cycle - temp.arrive;
    served++;
}
if(wait_time > 0) wait_time--;
sum_line += QueueItemCount(&line);
}
if(customers > 0)
{
    printf("Кількість прийнятих клієнтів:           %5ld\n",
           customers);
```



Моделювання реальної черги

```
printf("Кількість клієнтів, які обслуговані: %5ld\n",
       served);
printf("Кількість відмов: %5ld\n",
       turnaways);
printf("Середня довжина черги: %.2f\n",
       (double)sum_line / cyclelimit);
printf("Середній час очікування: %.2f хв\n",
       (double)line_wait / served);
}
else
    puts("Клієнти відсутні!");
EmptyTheQueue(&line);
printf("=====\n");
return 0;
}

// x - середній час між прибуттям клієнтів у хвилину повертає
// true, якщо клієнт з'являється впродовж даної хвилини
```



Моделювання реальної черги

```
bool newcustomer(double x)
{
    if(rand() * x / RAND_MAX < 1)
        return true;
    else return false;
}

// when - час прибуття клієнта
// функція повертає структуру Item з часом прибуття,
// який встановлений в when, і часом обслуговування,
// що встановлений у випадкове значення з діапазону від 1 до 3

Item customertime(long when)
{
    Item cust;
    cust.processtime = rand() % 3 + 1;
    cust.arrive = when;
    return cust;
}
```

Моделювання реальної черги

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 80
Введіть середню кількість клієнтів, які надходять за годину: 20
=====
Кількість прийнятих клієнтів: 1625
Кількість клієнтів, які обслуговані: 1625
Кількість відмов: 0
Середня довжина черги: 0.52
Середній час очікування: 1.55 хв
=====
```

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 800
Введіть середню кількість клієнтів, які надходять за годину: 20
=====
Кількість прийнятих клієнтів: 16065
Кількість клієнтів, які обслуговані: 16065
Кількість відмов: 0
Середня довжина черги: 0.43
Середній час очікування: 1.28 хв
=====
```

Моделювання реальної черги

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 1
Введіть середню кількість клієнтів, які надходять за годину: 20
=====
Кількість прийнятих клієнтів: 24
Кількість клієнтів, які обслуговані: 22
Кількість відмов: 0
Середня довжина черги: 0.47
Середній час очікування: 1.14 хв
=====
```

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 1
Введіть середню кількість клієнтів, які надходять за годину: 20
=====
Кількість прийнятих клієнтів: 22
Кількість клієнтів, які обслуговані: 21
Кількість відмов: 0
Середня довжина черги: 0.30
Середній час очікування: 0.81 хв
=====
```

Моделювання реальної черги

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 80
Введіть середню кількість клієнтів, які надходять за годину: 25
=====
Кількість прийнятих клієнтів: 2030
Кількість клієнтів, які обслуговані: 2029
Кількість відмов: 2
Середня довжина черги: 1.59
Середній час очікування: 3.76 хв
=====
```

```
D:\KIT219\D\L9_2\bin\Debug\L9_2.exe

=====
Консультаційний кіоск
=====
Введіть тривалість моделювання в годинах: 80
Введіть середню кількість клієнтів, які надходять за годину: 30
=====
Кількість прийнятих клієнтів: 2346
Кількість клієнтів, які обслуговані: 2337
Кількість відмов: 96
Середня довжина черги: 5.49
Середній час очікування: 11.25 хв
=====
```