**Лабораторна робота №2.**

**Черги. Реалізація черг за допомогою циклічних масивів та покажчиків.**

**Мета роботи:** Закріпити техніку роботи з напівстатичними структурами даних на прикладі черги. Розглянути основні операції з чергою і познайомитись з типовими прикладами застосування черг.

**Теоретичні відомості.**

*Черга* – це спеціальний тип списку, де елементи включаються на одному кінці, який називається хвостом (tail), а вилучаються на іншому – голові (head). Чергу називають списком типу FIFO (first-in-first-out – першим включається – першим виключається).

Оператори, що виконуються над чергами, аналогічні операторам стеків:

1. MAKENULL (Q). Робить чергу Q порожньою.
2. HEAD (Q) – функція повертає перший елемент черги Q. Цю функцію можна реалізувати за допомогою операторів списку як RETRIEVE (FIRST(Q),Q).
3. ENQUEUE (x,Q) вставляє елемент *х* в хвіст черги Q. За допомогою операторів списку можна реалізувати цю функцію наступним чином INSERT(Q),Q).
4. DEQUEUE (Q) видаляє перший елемент черги Q. За допомогою операторів списку реалізується як DELETE (FIRST(Q),Q).
5. EMPTY(Q). Повертає значення true тоді і тільки тоді, коли Q є порожньою чергою.

*Реалізація черг за допомогою покажчиків.*

Для черг більш ефективним є їх представлення основане на масивах з безпосереднім використанням покажчиків.

Для початку об’явимо комірки наступним чином:

**Type**

celltype = **Record**

element: elementtype;

next: **^** celltype

**end**;

Тепер можна визначити список, який вміщує покажчики на початок і кінець черги. Першим елементом черги буде елемент заголовку, в якому поле element не враховується.

**Type**

Queue = **Record**

head, tail: **^**celltype

**end**;

Наведемо програми для п’яти операторів, які виконуються над чергами:

|  |  |
| --- | --- |
| **procedure** MAKENULL (**var** Q : Queue);  **begin**  new(Q.head); {створення комірки заголовку}  Q.head**^**.next := nil;  Q.tail := Q.head  **end**; | **procedure** DEQUEUE (**var** Q: Queue);  **begin**  **if** EMPTY (Q) **then**  **error (`**Черга порожня`)  **else**  Q.head := Q.head**^**.next  **end;** |
| **function** HEAD (Q: Queue): elementtype;  **begin**  **if** EMPTY (Q) **then**  **error (`**Черга порожня`)  **else**  HEAD := Q.head**^**.next**^**.element  **end;** | **procedure** ENQUEUE (x: elementtype; **var** Q: Queue);  **begin**  new(Q.tail**^**.next);  Q.tail := Q.tail**^**.next;  Q.tail**^**.element := x;  Q.tail**^**.next := nil  **end;** |
| **function** EMPTY (Q: Queue): boolean;  **begin**  **if** Q.head = Q.tail **then**  EMPTY := true  **else** EMPTY := false  **end;** |  |

*Реалізація черг за допомогою циклічних масивів.*

Можлива реалізація черги на основі циклічного масиву. Представимо масив у вигляді циклічної структури, де перший елемент масиву розташований за останнім (рис.6).

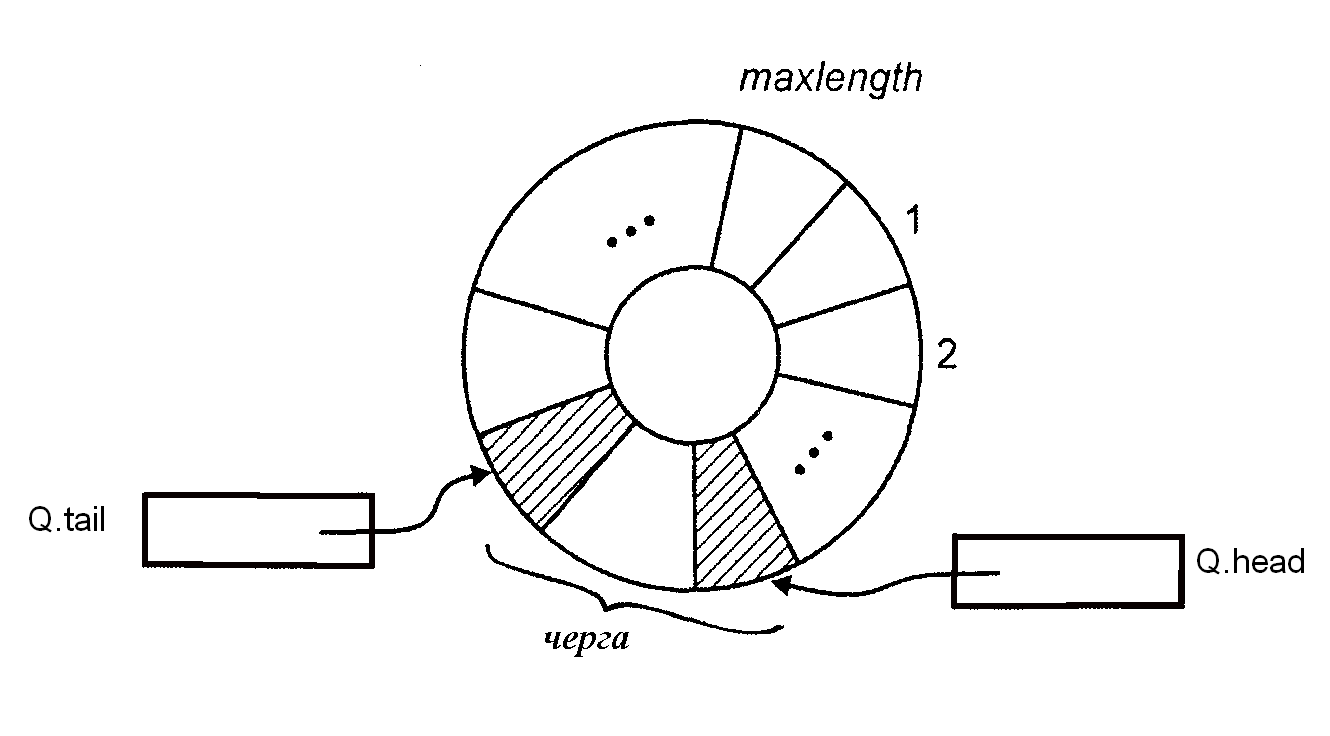


Рис. 6. Реалізація черги на основі циклічного масиву

Кінець черги знаходиться на певній відстані від її початку за годинниковою стрілкою. Для включення нового елемента в чергу досить перемістити *Q.tail* (покажчик на кінець черги) на одну позицію за годинниковою стрілкою і записати елемент у цю позицію. При виключенні елемента з черги досить просто перемістити *Q.head* (покажчик на початок черги) на одну позицію за годинниковою стрілкою.

Треба відзначити, що при такій реалізації час виконання операцій включення і виключення елементів не залежить від довжини черги.

Наведемо п’ять команд, що виконуються над чергами, які використовують описану реалізацію. Формально черги визначаються наступним чином:

**Type**

Queue = **Record**

elements : array [1..maxlength] **of** elementtype;

head, tail : integer

**end**;

Відповідні процедури наведені в лістингу. Функція addone (*i*) додає одиницю до позиції *і* в «циклічному» розумінні.

|  |  |
| --- | --- |
| **function** addone (i: integer): integer;  **begin**  addone := (і mod maxlength) + 1  **end;** | **procedure** MAKENULL (**var** Q : Queue);  **begin**  Q.head := 1; Q.tail := maxlength  **end**; |
| **function** EMPTY (**var** Q: Queue): boolean;  **begin if** addone (Q.tail) = Q.head  **then** EMPTY := true  **else** EMPTY := false  **end;** | **function** HEAD (**var** Q: Queue): elementtype;  **begin if** EMPTY (Q) **then**  **error (`**Черга порожня`)  **else** HEAD := Q.elements[Q.head]  **end;** |
| **procedure** ENQUEUE (x: elementtype; **var** Q: Queue);  **begin**  **if** addone(addone(Q.tail)) = Q.head  **then** error (`Черга повна`)  **else begin**  Q.tail := addone(Q.tail);  Q.elements[Q.tail] := x;  **end; end;** | **procedure** DEQUEUE (**var** Q: Queue);  **begin**  **if** EMPTY (Q) **then**  **error (`**Черга порожня`)  **else** Q.head:= addone(Q.head)  **end;** |

**Завдання**.

Написати програму: а) реалізації черги за допомогою покажчиків;

б) реалізації черги за допомогою циклічного масиву. Виконати: додавання елемента в чергу; видалення елемента з черги; перевірка, чи порожня черга; перегляд елемента в голові черги; очищення черги.

**Контрольні запитання**.

1. Що таке структура даних? Що таке черга?
2. Чим відрізняються статичні, напівстатичні та динамічні структури даних?
3. У чому полягає схожість та відмінність структур даних список, стек та черга?
4. Чи можна дістатися середини або початку черги, обійшовши її кінець (хвіст)?
5. Наведіть приклади з життя, де зустрічається «принцип черги».
6. У чому переваги реалізації черги за допомогою циклічного масиву від реалізації черги за допомогою звичайного масиву?