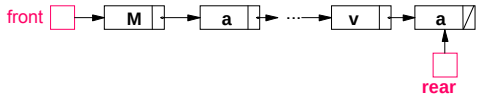
Опашка

Дефиниране

```
struct queue                // опашка
{
        LINK front;         // начало на опашка
        LINK rear;          // край на опашка
};
typedef struct queue QUEUE;
```



Характеристики

Нов възел се добавя в края (rear) на опашката.

Най-старият възел се изтрива в началото (front) на опашката.

Операции

1. **Създаване на празна опашка**
2. **Включване на възел в края на опашката**
3. **Изтриване на възел в началото на опашката**
4. **Намиране на първия елемент в опашката**
5. **Освобождаване на заетата памет от опашката**

1. Създаване на празна опашка

Алгоритъм

начало ← ПРАЗНО
край ← ПРАЗНО

```
void create (QUEUE *q)
{
        q->front = NULL;            // празна опашка
        q->rear  = NULL;
}
```

2. Включване на възел в края на опашката

Алгоритъм

отдели памет за нов възел във временната връзка
ако отделянето на памет не е успешно
  неуспешна операция
в противен случай
  запази данните в новия възел
  следваща компонента на новия възел ← ПРАЗНО
  ако опашката е празна
    насочи начало към новия възел
    насочи край към новия възел
  в противен случай
    насочи следващата компонента на край към новия възел
    насочи край към новия възел
  успешна операция

```
int put (QUEUE *q, DATA key)
{
  LINK p;                                  // временна връзка
  p = (LINK)malloc(sizeof(NODE));          // нов възел
  if (p == NULL)
  {
    printf("Няма достатъчно памет!\n");
    return FAILURE;
  }
  else
  {
    p->data = key;            // запазва данните в новия възел
    p->next = NULL;           // next ← ПРАЗНО
    if (q->front == NULL)     // Включване в празна опашка
    {
      q->front = p;           // насочва начало към възела
      q->rear = p;            // насочва край към възела
    }
```

```
    else                        // Включване в края на опашката
    {
      q->rear->next = p;        // насочва next на края към възела
      q->rear = p;              // насочва край към възела
    }
    return SUCCESS;
  }
}
```

3. Изтриване на възел в началото на опашката

Алгоритъм

ако опашката е ПРАЗНА
** неуспешна операция**
в противен случай
** запази данните на старото начало**
** насочи временната връзка към начало**
** придвижи начало към следващия елемент в опашката**
** освободи паметта, заета от временната връзка**
** ако начало е ПРАЗНО**
**   край ← ПРАЗНО**
** успешна операция**

```
int get (QUEUE *q, DATA *key)
{
  if (q->front == NULL && q->rear == NULL)
  {  printf ("Празна опашка!\n");
     key = NULL;
     return FAILURE;
  }
  else
  {  LINK p;                    // временна връзка
     *key = q->front->data;     // запазва данните в начало
     p = q->front;      // насочва временната връзка към начало
     q->front = p->next;        // придвижва начало
     free (p);                  // освобождава паметта
     if (q->front == NULL)      // последен елемент е изтрит
       q->rear = NULL;          // начало ←ПРАЗНО
     return SUCCESS;
  }
}
```

4. Намиране на първия елемент в опашката

Алгоритъм

ако опашката е ПРАЗНА
** неуспешна операция**
в противен случай
** запази данните на начало**
** успешна операция**

```
int look (QUEUE q, DATA *key)
{
  if (q.front == NULL && q.rear == NULL)
  {
    printf("Празна опашка!\n");
    key = NULL;
    return FAILURE;
  }
  else
  {
    *key = q.front->data;           // запазва данните на начало
    return SUCCESS;
  }
}
```

5. Освобождаване на заетата памет от опашката

Алгоритъм

насочи временната връзка към начало на опашката
докато начало на опашката не е ПРАЗНО
** придвижи начало на опашката**
** освободи паметта, заета от предишния връх**
** насочи временната връзка към начало**
ако начало е ПРАЗНО
** край ← ПРАЗНО**

```
void freequeue (QUEUE *q)
{
  LINK p;                                  // временна връзка
  for (p = q->front; q->front != NULL; p = q->front)
  {
    q->front = q->front->next;             // придвижва начало
    free(p);                               // освобождава паметта
  }
  if (q->front == NULL)                    // ако начало=ПРАЗНО
    q->rear = NULL;                        // край ← ПРАЗНО
  printf("Заетата памет от опашката е освободена!\n");
}
```

Задача: Отпечатайте съдържанието на опашка от символи след всяка операция при въвеждане на следната последователност:

Eas*yQue***st***i*on****

като буква означава „включи“ символ, а „*“ означава „изтрий“ символ от опашката.

1. **Напишете функция print(), която отпечатва съдържанието на опашка.**

Алгоритъм

```
ако опашката е празна
  печат: празна опашка
в противен случай
  насочи временната връзка към начало на опашката
  докато временната връзка не сочи край на опашка
    отпечатай данните във временния възел
    придвижи временната връзка
  отпечатай данните в край
```

2. **В програмата отпечатайте съдържанието на опашката след всяка операция.**

Алгоритъм

```
създай празна опашка
въведи низ
докато не е достигнат края на низа повтаряй
  ако текущият символ е буква
    включи символа в опашката
  в противен случай
    ако текущият символ е *
      изтрий символ от опашката
  отпечатай опашката
  премини към следващия символ в низа
```