Линеен списък

Линеен списък е подредена последователност от елементи, в която операциите включване, изключване и т.н. се извършват в произволни места, но със запазване на последователността.

Видове списъци

1. **Стек – операциите се извършват само в единия му край (връх на стека).**
2. **Опашка – включването се извършва в края, а изключването в началото на опашката.**
3. **Дек (DEQueue – Double-Ended Queue) – включването и изключването се извършват и в двата края на дека.**

Приложение

– приоритет на заданията, които се обработват от операционната система;

– синтактически анализ на текст.

Логическо описание

Съединение на елемент от тип T**, наречен** начало **на списъка, и** тяло **на списъка.**

данни: тип T

тип СПИСЪК$_T$ = (ПРАЗНО | НЕПРАЗЕН_СПИСЪК$_T$)

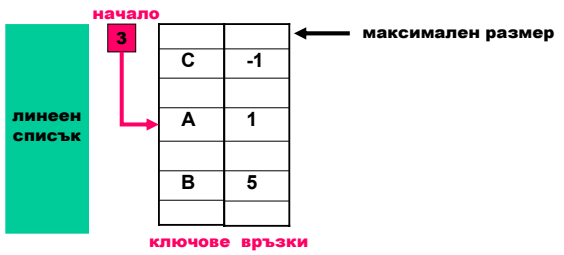
тип НЕПРАЗЕН_СПИСЪК$_T$ = (начало: T;
тяло: СПИСЪК$_T$)

Операции

1. **Създаване на линеен списък**
2. **Празен линеен списък**
3. **Включване на елемент на произволно място в линеен списък**
4. **Изключване на елемент от произволно място в линеен списък**
5. **Търсене на елемент в линеен списък**

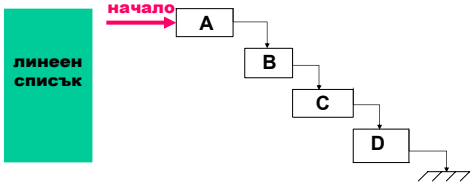
Физическо представяне

1. **Непрекъснато представяне – чрез паралелни масиви – много неудобно**



начало – тип ЦЯЛО
ключове – фиксирана област от паметта (масив)
връзки – фиксирана област от паметта (масив)

2. **Верижно представяне**



Стек

Стек е линеен списък, в който всички операции (включване, изключване и т.н.) се извършват само в единия му край (връх на стека).

Нарича се LIFO структура (last-in, first-out – последен влязъл, първи излязъл).

Приложение

– обръщане на входа в обратен ред;

– изчисляване на изрази – операциите с нисък приоритет се добавят в стека, докато операциите с висок приоритет се изпълнят;

– езикови транслатори – стекът забавя обработката на външните последователности, докато вътрешните последователности се транслират;

– извикване на подпрограми;

– изпълнение на рекурсивни подпрограми;

– синтактически анализ на текст – например при израз със скоби всяка отваряща скоба се добавя в стек, а всяка затваряща скоба изтрива от стека последната добавена скоба.

Логическо описание

Съединение на
– елемент от тип T, наречен връх на стека и
– тяло на стека.

данни: тип T

тип $СТЕК_T$ = (ПРАЗНО | $НЕПРАЗЕН\_СТЕК_T$)

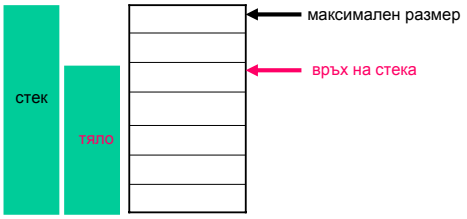
тип $НЕПРАЗЕН\_СТЕК_T$ = (връх: T;
тяло: $СТЕК_T$)

Операции

1. **Създаване на празен стек**
2. **Празен стек**
3. **Включване на елемент във върха на стека**
4. **Изключване на елемент от върха на стека**
5. **Намиране на последния елемент в стека**

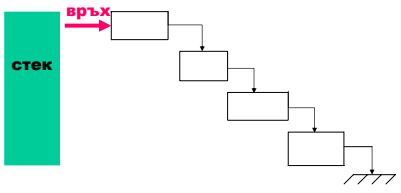
Физическо представяне

1. **Непрекъснато представяне**



тяло – **фиксирана област от паметта (масив)**
връх – **тип** ЦЯЛО

2. **Верижно представяне**



Дефиниране

```
#define SIZE <константа>        // максимален размер
typedef <тип> DATA;             // тип на данни
struct stack                    // стек
{
  int top;                      // връх на стек
  DATA t[SIZE];                 // тяло на стек
};
typedef struct stack STACK;
```

Характеристики

Добавяне и изключване на елемент от стека става само през неговия връх top.

1. Създаване на празен стек

Алгоритъм
връх ← ПРАЗНО

```
void create (STACK *s)
{
        s->top = 0;             // празен стек
}
```

2. Празен стек

Алгоритъм
ако връх = ПРАЗНО
** стекът е празен**
в противен случай
** стекът не е празен**

```
int empty (STACK s)
{
        return !s.top;
}
```

3. Включване на елемент във върха на стека

Алгоритъм
ако има място в стека
** връх ← връх + 1**
** тяло[връх] ← данна**
** успешна операция**
в противен случай
** неуспешна операция**

```
int push (STACK *s, DATA key)
{
  if (s->top<SIZE-1)
  {
    (s->top)++;
    s->t[s->top] = key;
    return (SUCCESS);
  }
  else
    return (FAILURE);
}
```

4. Изключване на елемент от върха на стека

Алгоритъм

```
ако стекът е празен
   неуспешна операция
в противен случай
   данни ← тяло[връх]
   връх ←  връх - 1
   успешна операция
```

```
int pop (STACK *s, DATA *key)
{
  if (empty(*s))
  {
    printf ("Празен стек!\n");
    return (FAILURE);
  }
  else
  {
    *key = s->t[s->top];
    (s->top)--;
    return (SUCCESS);
  }
}
```

5. Намиране на последния елемент в стека

Алгоритъм

```
ако стекът е празен
   неуспешна операция
в противен случай
   данни ← тяло[връх]
   успешна операция
```

```
int look (STACK s, DATA *key)
{
  if (empty(s))
  {
    printf ("Празен стек!\n");
    return FAILURE;
  }
  else
  {
    *key = s.t[s.top];
    return SUCCESS;
  }
}
```

Задача: **Използване на стек за изчисляване на аритметичен израз чрез използване на обратен полски запис (постфиксен запис) – операторът се записва след двата аргумента, а не между тях (инфиксен запис); отпада необходимостта от скоби.**

5*(((9+8)*(4*6))+7) инфиксен запис
5 9 8 + 4 6 * * 7 + * постфиксен запис

Въведете постфиксен израз, който съдържа аритметичните операции събиране и умножение. Интерпретирайте всеки операнд като команда: включи операнда в стека и всеки оператор като команда: изключи два операнда от стека, извърши операцията и включи резултата в стека. (Редът, в който двата операнда се получават чрез изключване от стека, не е от значение за операциите събиране и умножение, докато за изваждане и деление този ред е съществен.)

Алгоритъм

```
създай празен стек
въведи постфиксен израз като символен низ
насочи временен указател към началото на низа
докато временният указател не е стигнал края на низа
повтаряй
   резултат ← 0
   докато текущият символ е празна позиция
      придвижи временния указател
   ако текущият символ е +
      изключи втори операнд от върха на стека
      изключи първи операнд от върха на стека
      резултат ← сума от операндите
   ако текущият символ е *
      изключи втори операнд от върха на стека
      изключи първи операнд от върха на стека
      резултат ← произведение от операндите
   докато текущият символ е цифра
      резултат  ← 10*резултат + конвертиран символ в цифра
      придвижи временния указател
   включи резултата във върха на стека
   придвижи временния указател
изключи резултата от върха на стека
```

Опашка

Опашка е линеен списък, в който всички включвания се извършват в края и всички изключвания – в началото.

Нарича се FIFO структура (first-in, first-out – първи влязъл, първи излязъл).

Разлика между опашка и стек

– опашка – елементите се получават от опашката в същия ред, в който са попаднали в нея;

– стек – елементите се получават от стека в обратен ред на тяхното разполагане.

Приложение

– моделиране на поведението на системата сървър-клиент – клиенти чакат да бъдат обслужени в банка, супермаркет и т.н.;

– моделиране на работата на операционни системи с време деление (като буфер) – операционната система управлява опашка от незавършили задания, като всяко задание получава малко количество от процесорното време;

– откриване на палиндром – последователност, която е една и съща отпред-назад и отзад-напред.

Логическо описание

Съединение на
- **елемент от тип** T**, наречен** начало **на опашката,**
- **елемент от тип** T**, наречен** край **на опашката и**
- тяло **на опашката.**

данни: тип T

тип ОПАШКА$_T$ = (ПРАЗНО | НЕПРАЗНА_ОПАШКА$_T$)

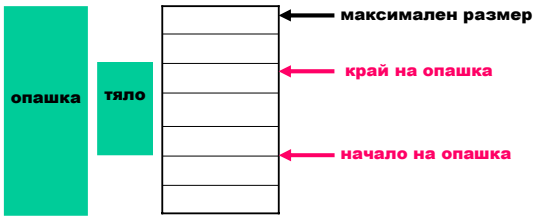
тип НЕПРАЗНА_ОПАШКА$_T$ = (начало: T;
край: T;
тяло: ОПАШКА$_T$)

Операции

1. **Създаване на празна опашка**
2. **Празна опашка**
3. **Включване на елемент в края на опашката**
4. **Изключване на елемент в началото на опашката**
5. **Намиране на първия елемент в опашката**

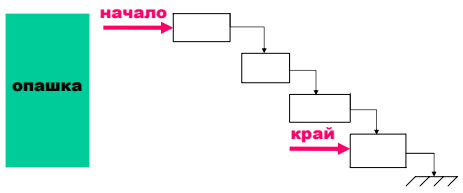
Физическо представяне

1. Непрекъснато представяне



начало – тип ЦЯЛО
край – тип ЦЯЛО
тяло – фиксирана област от паметта (масив)

2. Верижно представяне



Дефиниране

```
define SIZE <константа>          // максимален размер
typedef <тип> DATA;              // тип на данни
struct queue                     // опашка
{
  int front;                     // начало на опашка
  int rear;                      // край на опашка
  DATA t[SIZE];                  // тяло
};
typedef struct queue QUEUE;
```

Характеристики

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

Най-старият елемент се изключва от началото (front) на опашката.

1. Създаване на празна опашка

Алгоритъм

начало ← ПРАЗНО
край ← ПРАЗНО

```
void create (QUEUE *q)
{
        q->front = 0;          // празна опашка
        q->rear  = 0;
}
```

2. Празна опашка

Алгоритъм

ако начало = ПРАЗНО и край = ПРАЗНО или край < начало
** опашката е празна**
в противен случай
** опашката не е празна**

```
int empty (QUEUE q)
{
  return q.front == 0 && q.rear == 0 || q.rear < q.front;
}
```

3. Включване на елемент в края на опашката

Алгоритъм

ако има място в опашката
** ако опашката е празна**
**   начало ← начало + 1**
**   край ← край + 1**
**   тяло[край] ← данна**
** в противен случай**
**   край ← край + 1**
**   тяло[край] ← данна**
** успешна операция**
в противен случай
** неуспешна операция**

```
int put (QUEUE *q, DATA key)
{
  if (q->rear < SIZE-1)
  {
    if (empty(*q))
    {   (q->front)++;
        (q->rear)++;
        q->t[q->rear] = key;
    }
    else
    {   (q->rear)++;
        q->t[q->rear] = key;
    }
    return (SUCCESS);
  }
  else
    return (FAILURE);
}
```

4. Изключване на елемент от началото на опашката

Алгоритъм

```
ако опашката е ПРАЗНА
   неуспешна операция
в противен случай
   данни   ← тяло[начало]
   начало ← начало + 1
   успешна операция
```

```
int get (QUEUE *q, DATA *key)
{
  if (empty (*q))
  {
    printf ("Празна опашка!");
    return (FAILURE);
  }
  else
  {
    *key = q->t[q->front];
    (q->front)++;
    return (SUCCESS);
  }
}
```

5. Намиране на първия елемент в опашката

Алгоритъм

```
ако опашката е ПРАЗНА
   неуспешна операция
в противен случай
   данни ← тяло[начало]
   успешна операция
```

```
int look (QUEUE q, DATA *key)
{
  if (empty (q))
  {
    printf ("Празна опашка!\n");
    key = NULL;
    return FAILURE;
  }
  else
  {
    *key = q.t[q.front];
    return SUCCESS;
  }
}
```

Задача: Използване на опашка като буфер
Въведете изречение като символен низ.
Включете първия символ от всяка дума в опашка.
Отпечатайте получената дума, като изключите последователно елементите от опашката.
Приемете, че думите в изречението са разделени с празни позиции или табулатори.

Алгоритъм

```
създай празна опашката
състояние ← вън от дума
въведи изречение като символен низ
насочи временен указател към началото на низа
докато временният указател не е стигнал края на низа
повтаряй
   ако текущият символ не е празна позиция и табулатор
   и състояние = вън от дума
      състояние ← вътре в дума
      включи текущия символ в опашката
   ако текущият символ е празна позиция или табулатор
   и състояние = вътре в дума
      състояние ← вън от дума
   придвижи временния указател
докато опашката не е празна
   изключи елемент от опашката
   отпечатай изключения елемент
```