Курс «Алгоритмы и алгоритмические языки»
1 семестр 2013/2014

Лекция 14

Динамические структуры данных

- **Стек** (*stack*) – это динамическая последовательность *элементов*, количество которых изменяется, причем как добавление, так и удаление элементов возможно только с одной стороны последовательности (вершина стека).
- Работа со стеком осуществляется с помощью функций:

  `push(x)` – *затолкать* элемент **`x`** в стек;

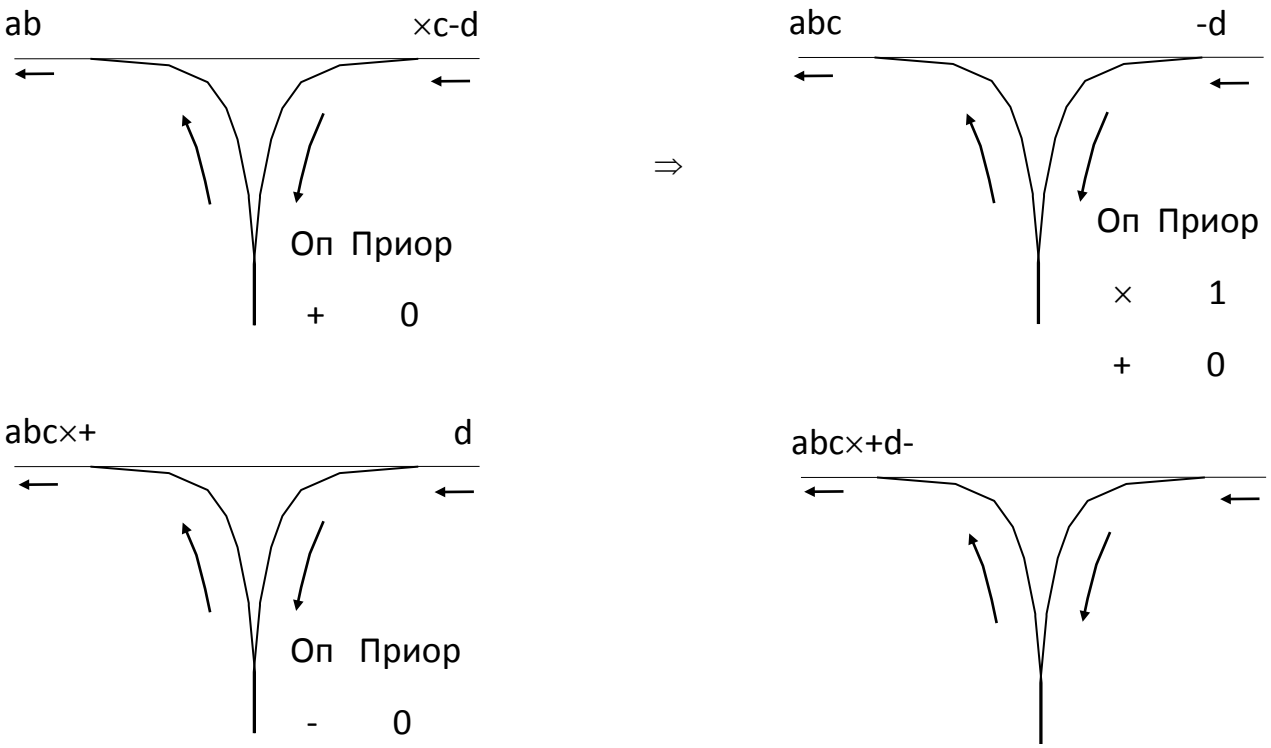
  `x = pop()` – *вытолкнуть* элемент из стека.
- Стек можно организовать на базе:
  - фиксированного массива **`stack[MAX]`**, где константа **`MAX`** задает максимальную глубину стека.
  - динамического массива, текущий размер которого хранится отдельно.
  - в обоих случаях необходимо хранить позицию текущей вершины стека.
  - можно использовать и другие структуры данных (например, список).

Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись (постфиксную).

a + b × c - d → abc × + d -

c × (a + b) - (d + e)/f → cab + × de + f/-



Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* Считывание символа-операции или переменной */
static char getop (void) {
  int c;
  while ((c = getchar ()) != EOF && isblank (c))
    ;
  return c == EOF || c == '\n' ? 0 : c;
}
```

Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись.

```
/* Является ли символ операцией */
static int isop (char c) {
  return (c == '+') || (c == '-') || (c == '*')
         || (c == '/');
}
/* Каков приоритет символа-операции */
static int prio (char c) {
  if (c == '(')
    return 0;
  if (c == '+' || c == '-')
    return 1;
  if (c == '*' || c == '/')
    return 2;
  return -1;
}
```

Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись.

```
int main (void) {
  char c, op;

  while (c = getop ()) {
    /* Переменная-буква выводится сразу */
    if (isalpha (c))
      putchar (c);
    /* Скобка заносится в стек операций */
    else if (c == '(')
      push (c);
    else <...>
```

Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись.

```
/* Операция заносится в стек в зависимости от приоритета */
else if (isop (c)) {
      while (! isempty ()) {
        op = pop ();
        /* Заносим, если больший приоритет */
        if (prio (c) > prio (op)) {
          push (op); break;
        } else
        /* Иначе выталкиваем операцию из стека */
          putchar (op);
      }
      push (c);
    } else <...>
```

Динамические структуры данных

- **Пример.** Перевод арифметического выражения в обратную польскую запись.

```
/* Скобка выталкивает операции до парной скобки */
} else if (c == ')')
      while ((op = pop ()) != '(')
        putchar (op);
  }
/* Вывод остатка операций из стека */
  while (! isempty ())
    putchar (pop ());
  putchar ('\n');
  return 0;
}
```

Очередь

- Очередь (*queue*) – это линейный список информации, работа с которой происходит по принципу *FIFO*.

  Для списка можно использовать статический массив: количество элементов массива (*MAX*) = наибольшей допустимой длине очереди.

- Работа с очередью осуществляется с помощью **двух функций**:

  `qstore()` – *поместить* элемент в конец очереди;

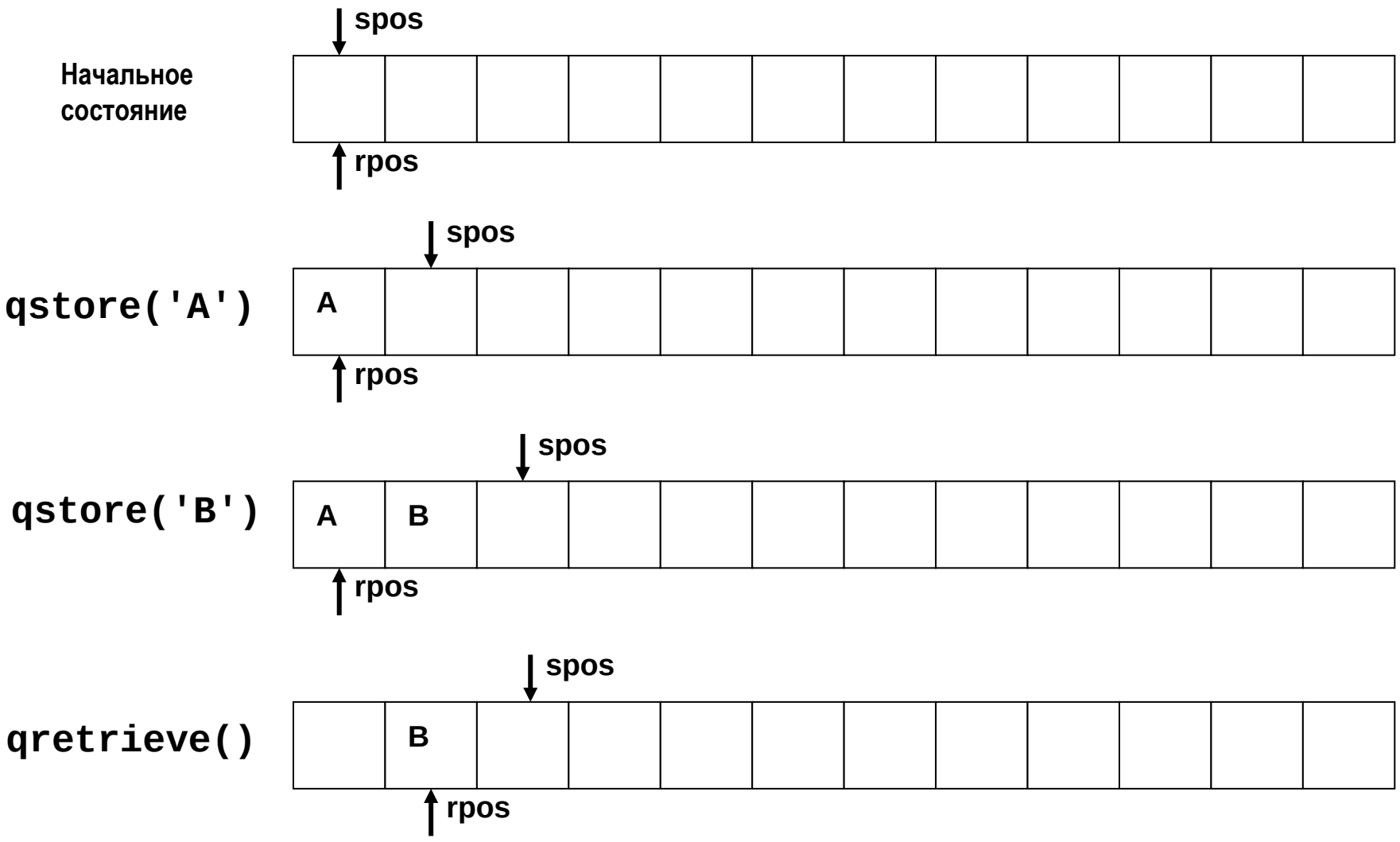
  `qretrieve()` – *удалить* элемент из начала очереди;

  и **двух глобальных переменных**:

  `spos` (индекс первого свободного элемента очереди: его значение < **`MAX`**)

  `rpos` (индекс очередного элемента, подлежащего удалению: «кто первый?»)

Очередь


spos
Начальное состояние
rpos
spos
qstore('A')
A
rpos
spos
qstore('B')
A
B
rpos
spos
qretrieve()
B
rpos

Очередь

- Тексты функций **qstore()** и **qretrieve()**

```
#define MAX    67
int queue[MAX];
int spos = 0, rpos = 0;

int qstore (int q) {
  if (spos == MAX) {
    /* Можно расширить очередь, см. реализацию стека */
    printf ("Очередь переполнена\n");
    return 0;
  }
  queue[spos++] = q;
  return 1;
}

int qretrieve (void) {
  if (rpos == spos) {
    printf ("Очередь пуста \n");
    return -1;
  }
  return queue[rpos++];
}
```

Улучшение – «зацикленная» очередь

```
#define MAX    67
int queue[MAX];
int spos = 0, rpos = 0;
```

```
int qstore (int q) {
  if (spos + 1 == rpos
      || (spos + 1 == MAX && !rpos) {
    printf ("Очередь переполнена \n");
    return 0;
  }
  queue[spos++] = q;
  if (spos == MAX)
    spos = 0;
  return 1;
}
```

Улучшение – «зацикленная» очередь

```
int qretrieve (void) {
  if (rpos == spos) {
    printf ("Очередь пуста \n");
    return -1;
  }
  if (rpos == MAX - 1) {
    rpos = 0;
    return queue[MAX – 1];
  }
  return queue[rpos++];
}
```

- Зацикленная очередь переполняется, когда **spos** находится непосредственно перед **rpos**, так как в этом случае запись приведет к **rpos == spos**, т.е. к пустой очереди.

Списки

- ***Односвязный список*** – это динамическая структура данных, каждый элемент которой содержит ссылку на следующий элемент (либо **NULL**, если следующего элемента нет).
- Доступ к списку осуществляется с помощью указателя на его первый элемент.

```
struct list {
  struct data info;      /* Данные */
  struct list *next;     /* Ссылка на след. элемент */
};
```

- Выделение элемента

```
struct list *phead = NULL;
phead = (struct list *) malloc (sizeof (struct list));
```

Списки

- Добавление элемента в начало

```
struct list *phead = NULL;


struct list *add_element (struct list *phead, struct
                          data *elem) {
  struct list *new = malloc (sizeof (struct list));
  new->info = *elem;
  new->next = phead;
  return new;
}
```

Списки

- Добавление элемента в конец

```
struct list *phead = NULL;


struct list *add_element (struct list *phead, struct
                          data *elem) {
  if (! phead) {
    phead = (struct list *) malloc (sizeof (struct list));
    phead->info = *elem;
    phead->next = NULL;
    return phead;
  }
  while (phead->next != NULL)
    phead = phead->next;
  phead->next = (struct list *) malloc (sizeof (struct list));
  phead->next->info = *elem;
  phead->next->next = NULL;
  return phead;
}
```