Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 17

9 ноября 2019 г.

Динамические структуры данных. Стек

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 (примеры):

- 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 (список).

Организация стека на динамическом массиве

```
struct stack {
  int sp;         /* Текущая вершина стека */
  int sz;         /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static void push (char c) {
  if (stack.sz == stack.sp + 1) {
    stack.sz = 2*stack.sz + 1;
    stack.stack = (char *) realloc (stack.stack,
                                    stack.sz*sizeof (char));
  }
  stack.stack[++stack.sp] = c;
}
```

Организация стека на динамическом массиве

```
struct stack {
  int sp;       /* Текущая вершина стека */
  int sz;       /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static char pop (void) {
  if (stack.sp < 0) {
    fprintf (stderr, "Cannot␣pop:␣stack␣is␣empty\n");
    return 0;
  }
  return stack.stack[stack.sp--];
}
```

Дома. Сделайте, чтобы результат записывался по указателю-аргументу, а функция возвращала код успеха операции.

Организация стека на динамическом массиве

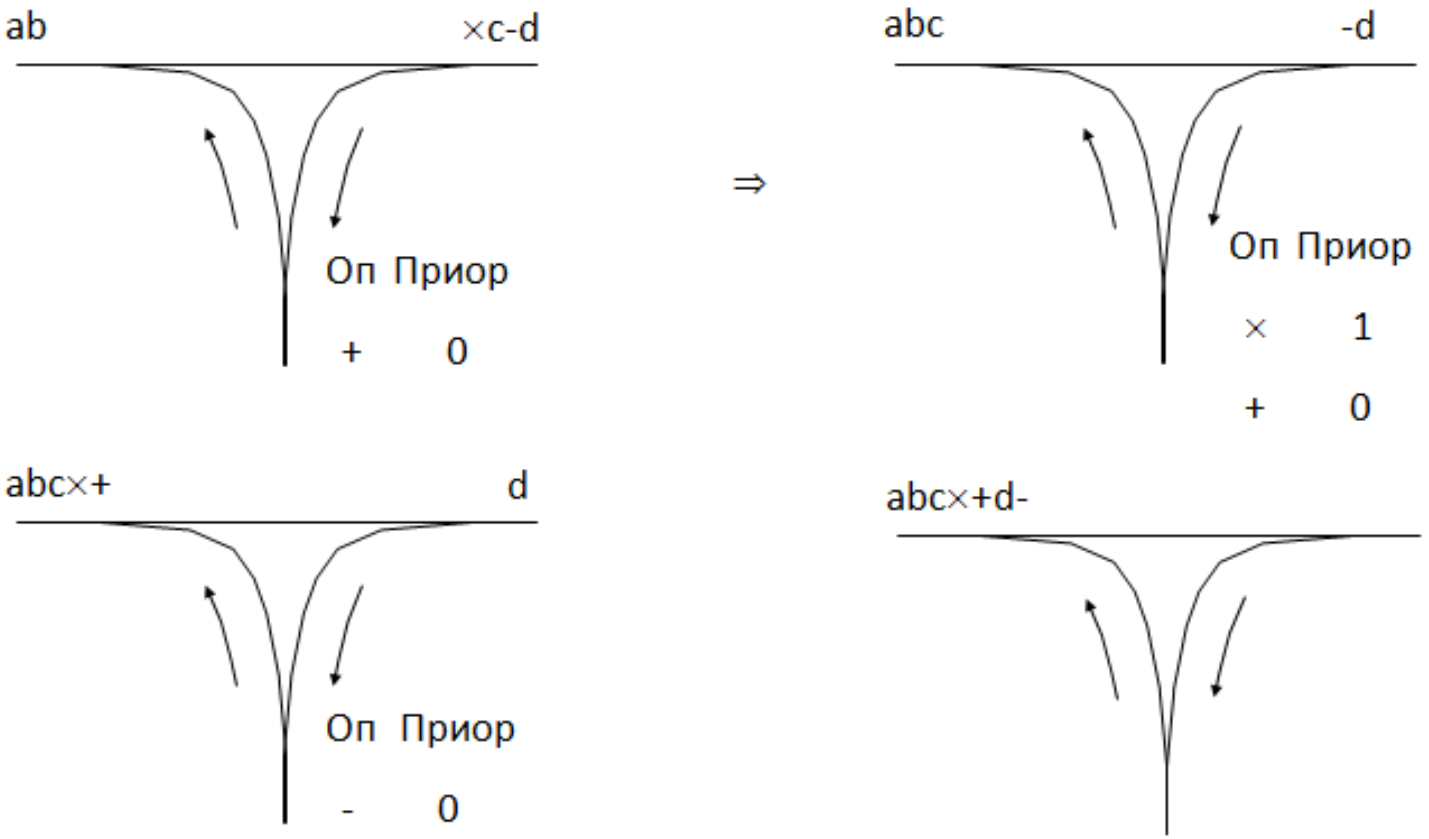
```
struct stack {
  int sp;         /* Текущая вершина стека */
  int sz;         /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static int isempty (void) {
  return stack.sp == -1;
}
```

Пример работы со стеком

Перевод арифметического выражения в обратную польскую запись (постфиксную).

$$a + b \times c - d \quad \rightarrow \quad abc \times +d-$$
$$c \times (a + b) - (d + e)/f \quad \rightarrow \quad cab + \times de + f/-$$



Перевод арифметического выражения в обратную польскую запись

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

#include "stack.c"

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

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;
}
```

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 <...>
```

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 <...>
```

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;
}
```

Дома. Введите операцию **peek()** и перепишите код с ее помощью. Обработайте случай непарных скобок.

Организация стека как библиотеки

stack.h:

```
extern void push (char);
extern char pop (void);
extern int isempty (void);
```

stack.c:

```
#include "stack.h"
struct stack {
  <...>
};
static struct stack stack
        = { <...> };
```

main.c:

```
#include "stack.h"
int main (void) {
  <...push (c), pop (), ...>
}
```

```
$gcc main.c stack.c -o main
```

Организация стека как библиотеки

stack.h:

```
struct stack;   // forward declaration
extern void push (struct stack *, char);
extern char pop (struct stack *);
extern int isempty (struct stack *);
extern struct stack* new_stack (void);
extern void free_stack (struct stack *);
```

stack.c:

```
#include "stack.h"
struct stack {
  <...>
};
void push (struct stack *stack, char c ) {
  if (stack->sz == stack->sp + 1) <...>
}
<...>
```

Организация стека как библиотеки

stack.c:

```
struct stack* new_stack (void) {
  struct stack *s = malloc (sizeof (struct stack));
  *s = (struct stack) { .sp = -1, .sz = 0, .stack = NULL };
  return s;
}
void free_stack (struct stack *s) {
  free (s->stack);
  free (s);
}
```

main.c:

```
#include "stack.h"
int main (void) {
  struct stack *s = new_stack ();
  <...push (s, c), pop (s), ...>
  free_stack (s);          <...>
}
```

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?”

Пример реализации

```
int queue[MAX]; //  Заведите enum
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) { //  Очередь пуста
    return -1;
  }
  return queue[rpos++];
}
```

Улучшение — «зацикленная» очередь

```
int queue[MAX]; //  Заведите enum
int spos = 0, rpos = 0;

int qstore (int q) {
  if (spos + 1 == rpos
      || (spos + 1 == MAX && !rpos) {
    printf ("Очередь␣переполнена␣\n");
    // Дома. Реализуйте очередь на динамическом массиве.
    return 0;
  }
  queue[spos++] = q;
  if (spos == MAX)
    spos = 0;
  return 1;
}
```

Улучшение — «зацикленная» очередь

```
int queue[MAX]; //  Заведите enum
int spos = 0, rpos = 0;

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 приведёт к **rpos == spos**, т.е. к пустой очереди.