НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (углубленный уровень), 1-й курс, 2-й семестр.

Методические указания

Тематическое занятие 17

Линейные списки, очереди, стеки.

Содержание

Связанные динамические данные

Основные определения

Линейный список (list) – это динамическая структура данных, которые представляют собой совокупность линейно связанных однородных элементов.

Линейный список называется ***односвязным,*** если каждый его элемент (кроме последнего) с помощью указателя связывается с одним (следующим) элементом.

В ***кольцевом списке*** имеется связь между последним и первым элементами.

Очередь (queue) – частный случай линейного односвязного списка, организованного по принципу ***"first in, first out"*** (***FIFO***, *«первым пришел, первым ушел»*). Для очереди разрешено только два действия:

- *добавление элемента в **конец (хвост)** очереди,*
- *удаление элемента из **начала (головы)** очереди.*

Стек (stack) – частный случай линейного односвязного списка, организованного по принципу ***"last in, first out"*** (***LIFO***, *«последним пришел, первым ушел»*). Для стека разрешено *добавлять и удалять элементы только с одного конца* списка, который называется ***вершиной (головой)*** стека.

Организация связей

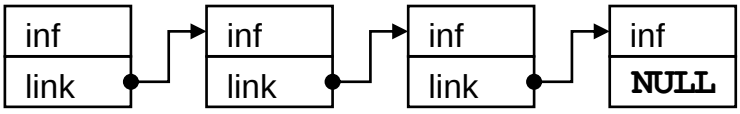
Высокая гибкость структур, основанных на связанных динамических данных, достигается за счет реализации двух возможностей:

- динамическое выделение и освобождение памяти под элементы в любой момент работы программы;
- установление связей между любыми двумя элементами с помощью указателей.

Для организации связей между элементами динамических структур данных требуется, чтобы каждый элемент содержал кроме информационных значений **хотя бы один указатель.** Поэтому в качестве элементов таких структур следует использовать **структуры,** которые могут объединять в единое целое разнородные данные.

В простейшем случае элемент динамической структуры данных должен состоять *из двух полей:* ***информационного*** (inf) и ***указательного*** (link).

Схематичное изображение такой структуры данных:



Соответствующее ей объявление:

```
typedef struct elem {
    int inf;              /* inf  - информационное поле */
    struct elem *link;    /* link - указательное поле   */
} Elem;
```

Очередь

Указатели очереди

Для создания очереди (*queue*) и работы с ней необходимо иметь как минимум два указателя:

- на ***начало*** очереди (назовем его **BegQ**, от *begin of queue*),
- на ***конец*** очереди (назовем **EndQ** – *end of queue*).

Кроме того, для освобождения памяти удаляемых элементов потребуется ***дополнительный*** указатель (назовем его **p**), он часто используется и в других ситуациях для удобства работы с очередью.

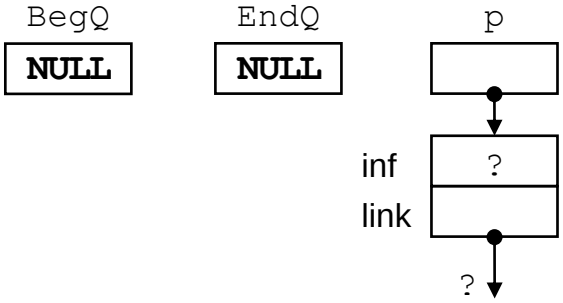
```
Elem *BegQ;
Elem *EndQ;
Elem *p;
```

Создание очереди

1. Исходное состояние.

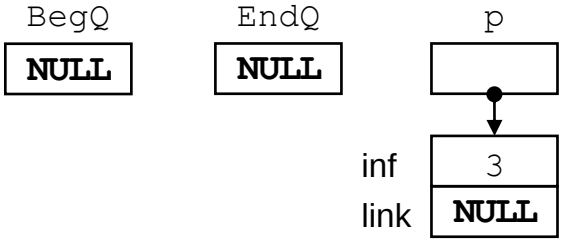

```
BegQ = NULL;
EndQ = NULL;
```

2. Выделение памяти под первый элемент очереди.



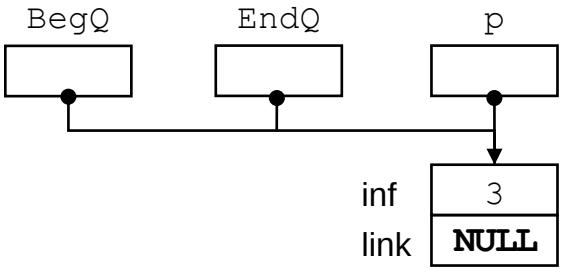

```
p = (Elem *) malloc(sizeof(Elem));
```

3. Занесение данных в первый элемент очереди.




```
p->inf = 3;
p->link = NULL;
```

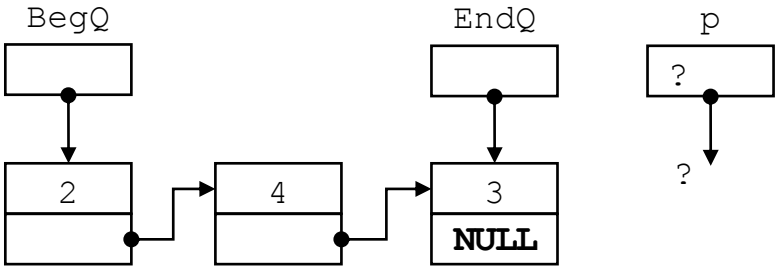
4. Установка указателей `BegQ` и `EndQ` на созданный первый элемент.



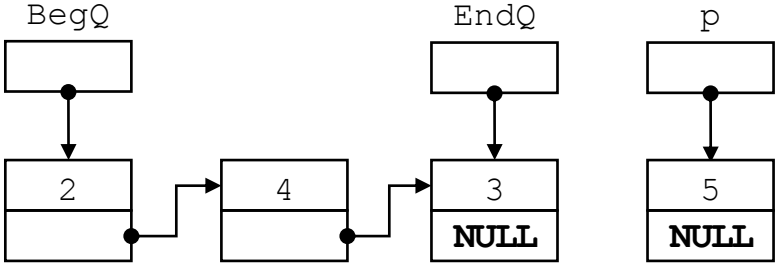

```
BegQ = p;
EndQ = p;
```

Добавление элемента в очередь

1. Исходное состояние.

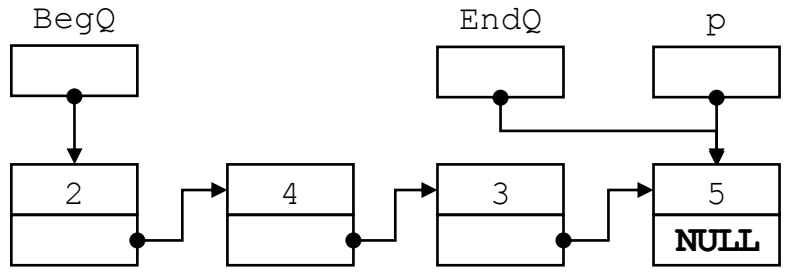


2. Выделение памяти под новый элемент и занесение в него данных.




```
p=(Elem*)malloc(sizeof(Elem));
p->inf = 5;
p->link = NULL;
```

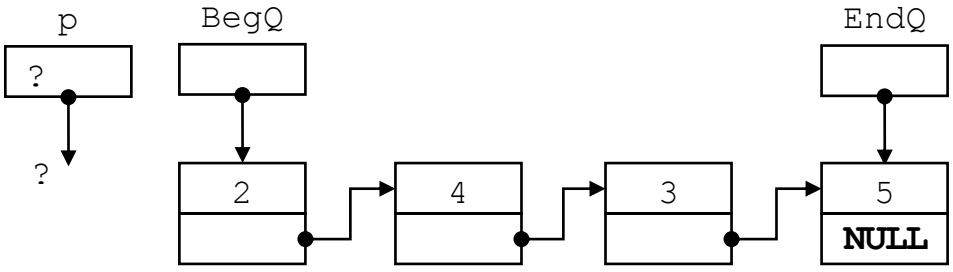
3. Установка связи между последним элементом очереди и новым, а также перемещение указателя `EndQ` на новый элемент.



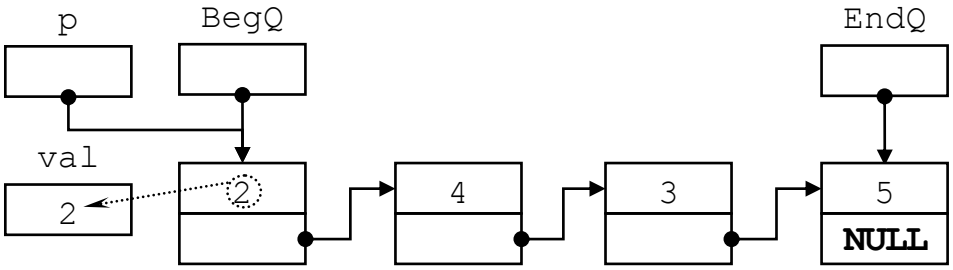

```
EndQ->link = p;
EndQ = p;
```

Удаление элемента из очереди

1. Исходное состояние.

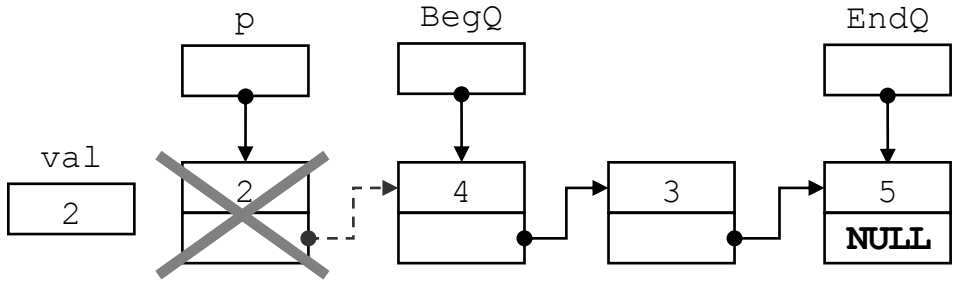


2. Извлечение информации из удаляемого элемента в переменную `val` и установка на него вспомогательного указателя `p`.




```
int val;
...
val = BegQ->inf;
p = BegQ;
```

3. Перестановка указателя `BegQ` на следующий элемент, используя значение поля `link` удаляемого элемента. Освобождение памяти удаляемого элемента `p`.




```
BegQ = p->link;
free(p);
```

Стек

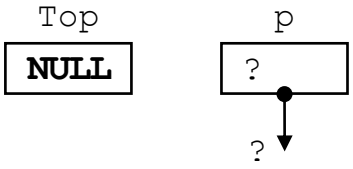
Указатели стека

Для работы со стеком (*stack*) необходимо иметь один указатель на ***вершину*** стека (назовем его **`Top`**). Также потребуется один ***дополнительный*** указатель (**`p`**), который используется для выделения и освобождения памяти элементов стека.

```
Elem *Top;
Elem *p;
```

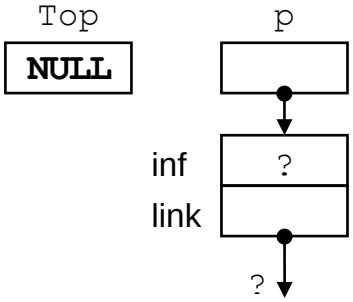
Создание стека

1. Исходное состояние.



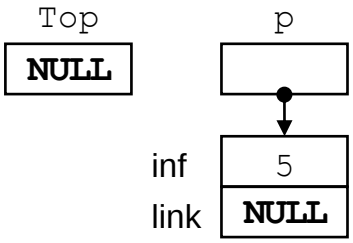

```
Top = NULL;
```

2. Выделение памяти под первый элемент стека.



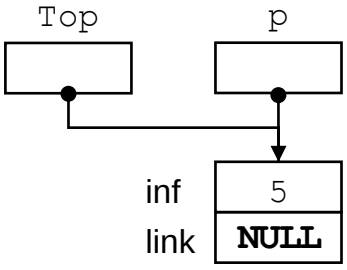

```
p = (Elem *) malloc(sizeof(Elem));
```

3. Занесение данных в первый элемент стека.




```
p->inf = 5;
p->link = NULL;
```

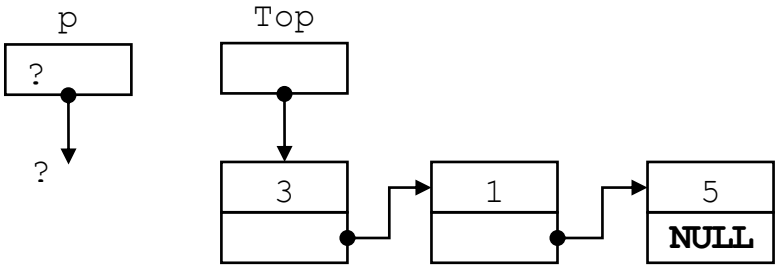
4. Установка указателя `Top` на созданный первый элемент.



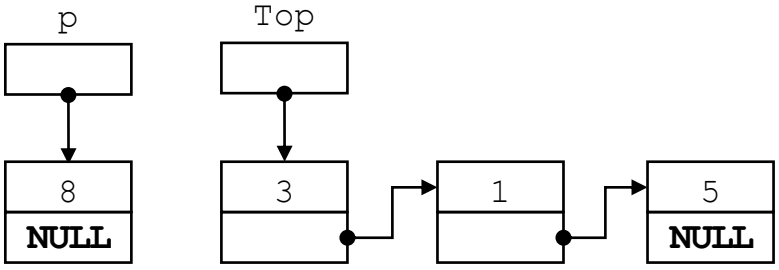

```
Top = p;
```

Добавление элемента в стек

1. Исходное состояние.

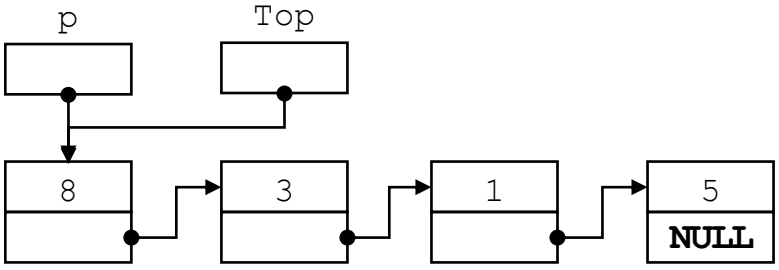


2. Выделение памяти под новый элемент и занесение в него данных.




```
p=(Elem*)malloc(sizeof(Elem));
p->inf = 8;
p->link = NULL;
```

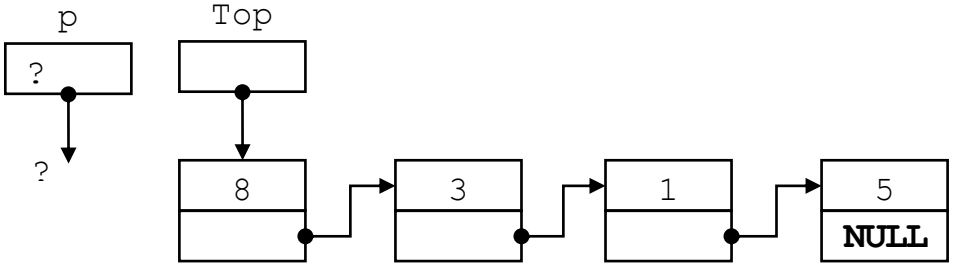
3. Установка связи между новым элементом и первым элементом стека, а также перемещение указателя `Top` на новый элемент.



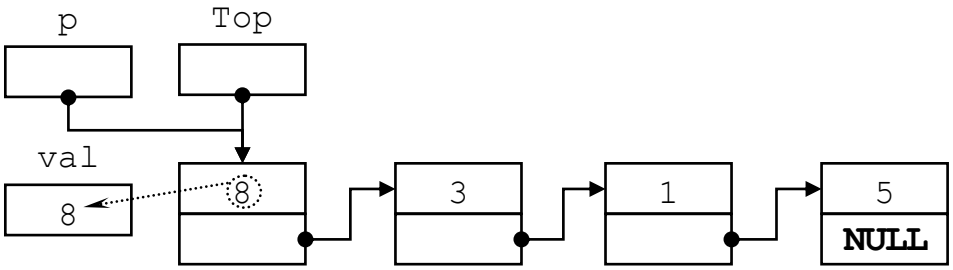

```
p->link = Top;
Top = p;
```

Удаление элемента из стека

1. Исходное состояние.

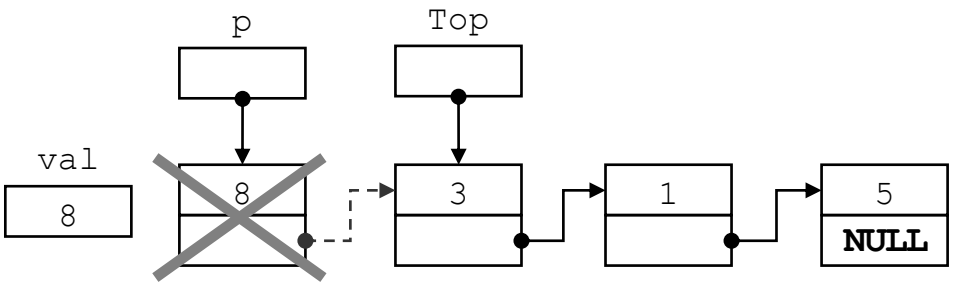


2. Извлечение информации из удаляемого элемента в переменную `val` и установка на него вспомогательного указателя `p`.




```
int val;
...
val = Top->inf;
p = Top;
```

3. Перестановка указателя `Top` на следующий элемент, используя значение поля `link` удаляемого элемента. Освобождение памяти удаляемого элемента `p`.




```
Top = p->link;
free(p);
```