Лекция 14.
Моделирование типов. Стек, очередь, дек

Стек

Абстрактная структура данных, линейная последовательность элементов с порядком обслуживания LIFO

LIFO – Last In First Out

1) Известно значение только «верхнего» - последнего добавленного – элемента

2) Удаляется первым тот элемент, который был добавлен в стек последним

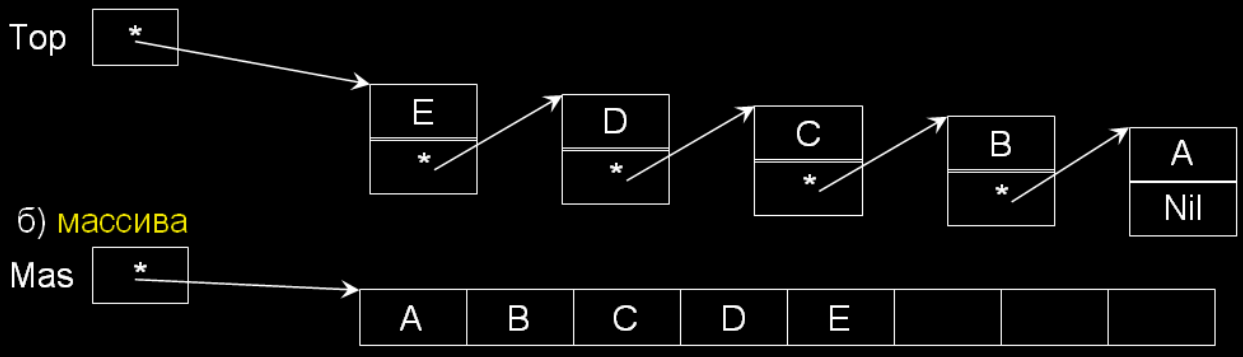
Со стеком мы знакомы с детства. Это пирамидка с разноцветными торами, одевающимися на штырь помогает изучать цвета и размеры, и порядок обработки LIFO. Во взрослой жизни стек – это шампуры и магазин с патронами. Второе название стека – магазин.

Стек

Абстрактная структура данных , последовательность элементов с порядком обслуживания LIFO

Может быть смоделирована на основе:

а) ссылочной структуры – динамического списка



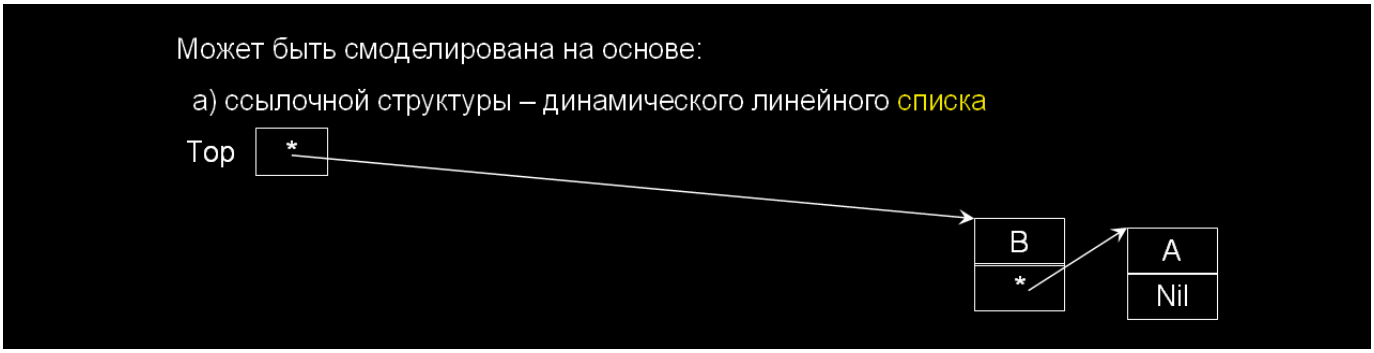
б) массива

(Индекс вершины -1,0,1..) N 4

Гречкина П.В. Стек

В примере типового расчета так и сделано (двумя способами).

Новые элементы стека добавляются в его вершину. В односвязном списке за вершину удобно принять первый элемент списка.



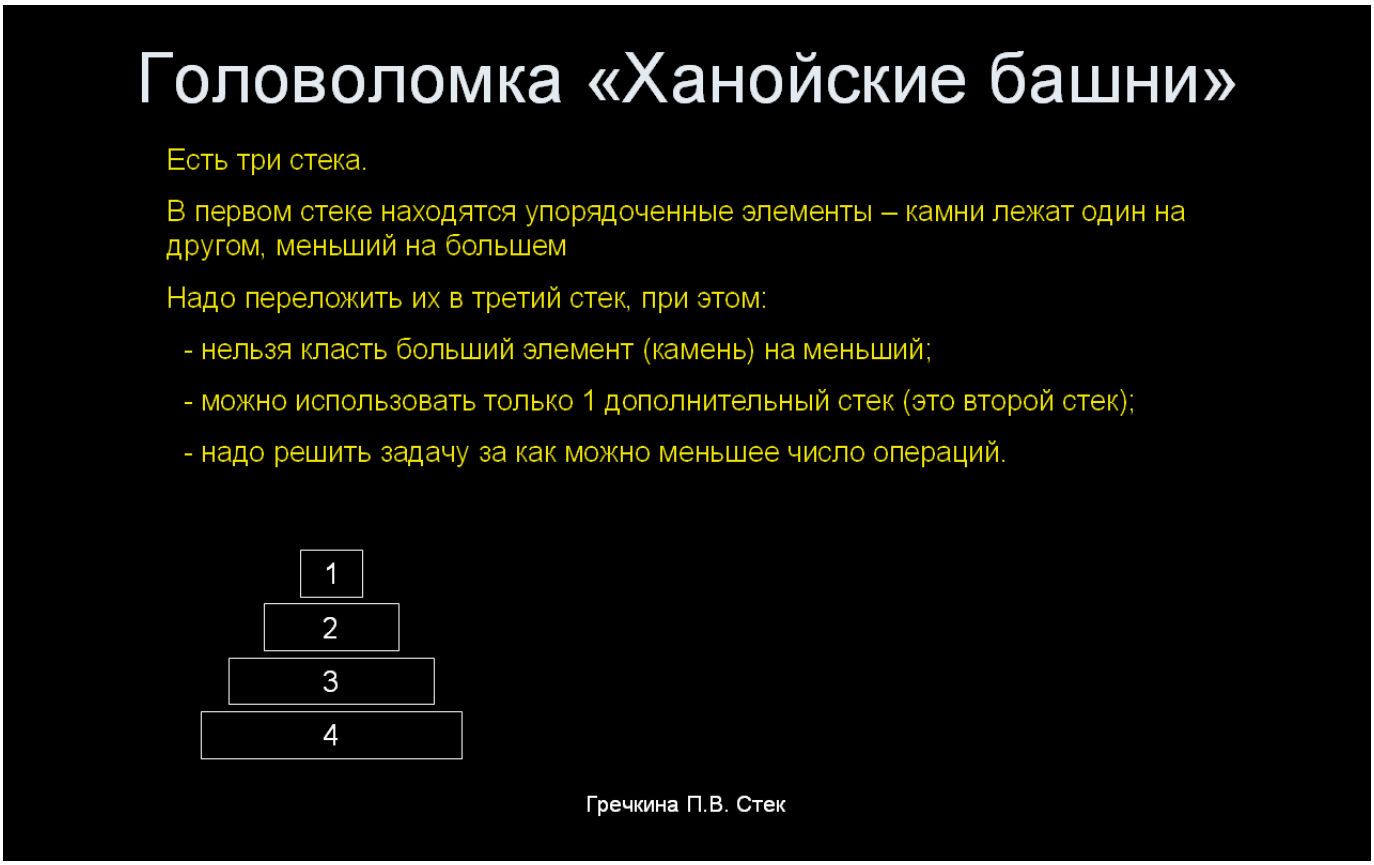
Оттуда же (из вершины стека) элементы и удаляются:

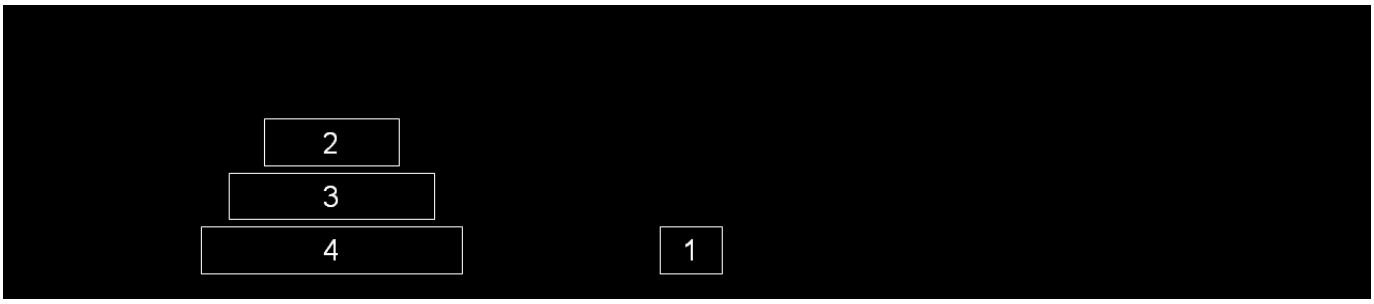
В одномерном массиве удобнее за вершину стека принять последний заполненный элемент. С конца массива удобнее удалять элементы и при выделении новой памяти свободное место появляется именно в конце массива.

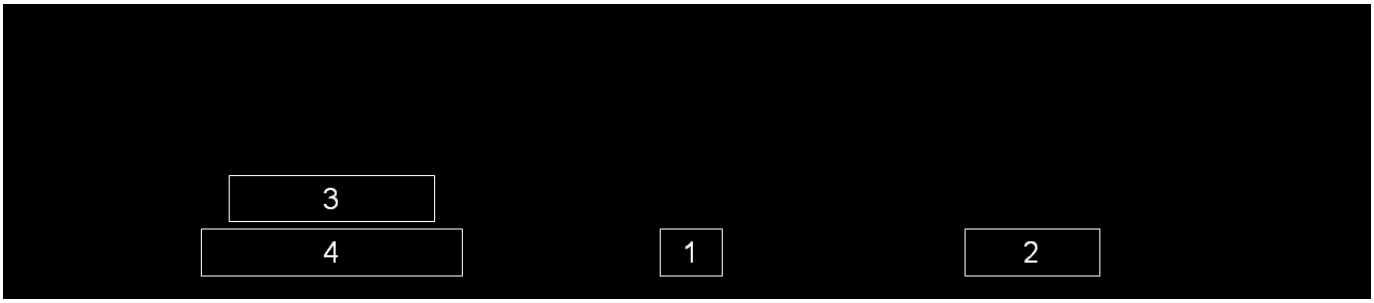
Когда стек пуст, N- значение индекса элемента массива, хранящего вершину стека, примем за -1. Когда в стеке один элемент – вершина стека в элементе с индексом 0.

Удаляются элементы тоже из вершины стека (с конца массива):

Самая известная головоломка со стеками – «Ханойская башня»:







2 на 1 подложить нельзя, но 1 на 2 – можно:

Головоломка «Ханойские башни»

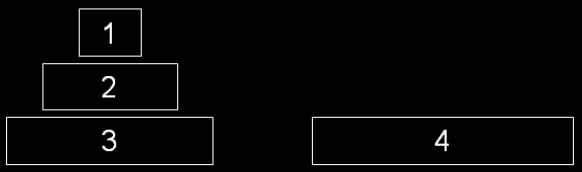
Есть три стека.

В первом стеке находятся упорядоченные элементы – камни лежат один на другом, меньший на большем

Надо переложить их в третий стек, при этом:

- нельзя класть больший элемент (камень) на меньший;
- можно использовать только 1 дополнительный стек (это второй стек);
- надо решить задачу за как можно меньшее число операций.

Гречкина П.В. Стек



Гречкина П.В. Стек

Нижний камень на своем месте в третьем стеке!

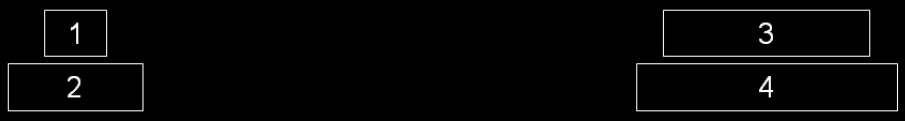
Головоломка «Ханойские башни»

Есть три стека.

В первом стеке находятся упорядоченные элементы – камни лежат один на другом, меньший на большем

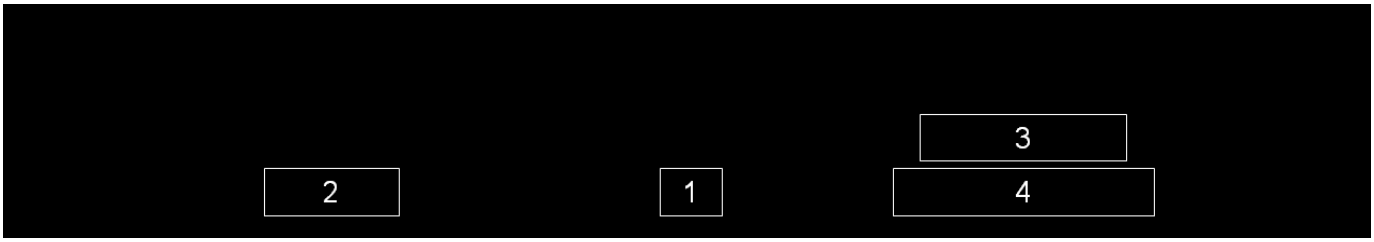
Надо переложить их в третий стек, при этом:

- нельзя класть больший элемент (камень) на меньший;
- можно использовать только 1 дополнительный стек (это второй стек);
- надо решить задачу за как можно меньшее число операций.



Гречкина П.В. Стек

Последние шаги



Все камни переложены в третий стек!

Камней может быть не 4, а любе количество. При этом достаточно трех стеков для решения головоломки. Попробуйте сами.

Просмотр элементов стека

Цикл работы со стеком:

1) Разбор стека с переносом элементов в дополнительный стек.

2) Возвращение элементов в исходный стек из дополнительного.

Хотя стек может храниться в списке или массиве, выполнение всех операций возможно с элементом в вершине стека! Иначе это НЕ СТЕК. Как же добраться до

второго элемента? Снять временно с его вершины первый, и сохранить его в дополнительном стеке. Таким образом все элементы по-очереди снимаются с вершины стека и перекладываются в дополнительный. Когда найдем нужный элемент, или просмотрим все элементы стека, надо возвратить все элементы так же по одному на место.

В дополнительном стеке элементы будут лежать в противоположном порядке.

Допустив, в стек положены элементы 23, 6, 8, 2. В вершине будет 2 (последний добавленный). Найдем сумму значений всех элементов стека.

Просмотр элементов стека

Цикл работы со стеком:
1) Разбор стека с переносом элементов в дополнительный стек.
2) Возвращение элементов в исходный стек из дополнительного.

```
//найти сумму элементов стека
function naiti( StackTop: PElem): real;
var
  Dop: PElem; // вершина дополнительного стека
  Sum: real; // сумма элементов стека
Begin
  Dop:=nil; Sum:=0;
  While StackTop<>nil do {1.}
  Begin
    Sum:=Sum+ StackTop^.info;
    TopToTop(StackTop, Dop);
  End;
  While Dop<>nil do TopToTop(Dop, StackTop);  {2.}
  Naiti:=Sum;
End;
```

Sum 0

2
8
6
23

Гречкина П.В. Стек

```
//найти сумму элементов стека
function naiti( StackTop: PElem): real;
var
  Dop: PElem; // вершина дополнительного стека
  Sum: real; // сумма элементов стека
Begin
  Dop:=nil; Sum:=0;
  While StackTop<>nil do {1.}
  Begin
    Sum:=Sum+ StackTop^.info;
    TopToTop(StackTop, Dop);
  End;
  While Dop<>nil do TopToTop(Dop, StackTop);  {2.}
  Naiti:=Sum;
End;
```

Sum 2

8
6
23

2

Гречкина П.В. Стек

Просмотр элементов стека

Цикл работы со стеком:
1) Разбор стека с переносом элементов в дополнительный стек.
2) Возвращение элементов в исходный стек из дополнительного.

```
//найти сумму элементов стека
function naiti( StackTop: PElem): real;
var
  Dop: PElem; // вершина дополнительного стека
  Sum: real; // сумма элементов стека
Begin
  Dop:=nil; Sum:=0;
  While StackTop<>nil do {1.}
  Begin
    Sum:=Sum+ StackTop^.info;
    TopToTop(StackTop, Dop);
  End;
  While Dop<>nil do TopToTop(Dop, StackTop);  {2.}
  Naiti:=Sum;
End;
```

Sum 39

23
6
8
2

Гречкина П.В. Стек

Сумма найдена. Возвращаем элементы в исходный стек.

Просмотр элементов стека

Цикл работы со стеком:
1) Разбор стека с переносом элементов в дополнительный стек.
2) Возвращение элементов в исходный стек из дополнительного.

```
//найти сумму элементов стека
function naiti( StackTop: PElem): real;
var
  Dop: PElem; // вершина дополнительного стека
  Sum: real; // сумма элементов стека
Begin
  Dop:=nil; Sum:=0;
  While StackTop<>nil do {1.}
  Begin
    Sum:=Sum+ StackTop^.info;
    TopToTop(StackTop, Dop);
  End;
  While Dop<>nil do TopToTop(Dop, StackTop);  {2.}
  Naiti:=Sum;
End;
```

Sum 39

2
8
6
23

Гречкина П.В. Стек

В рассмотренном коде использовалась дополнительная процедура TopToTop для перекладывания элемента из вершины одного стека в вершину другого.

```
// дополнительная процедура перекладывания элемента из
    одного стека в другой
procedure TopToTop( var StackTop, Dop: PElem);
var Elem: PElem;
begin
  Elem:=StackTop;
  StackTop:= StackTop^.next;
  Elem^.Next:= Dop;
  Dop:= Elem;
end;
```

На языке С код пишется аналогично. Сравним построчно на примере задачи:

Задание: смоделировать стек на основе собственного типа односвязного списка и решить на этой структуре задачу по своему варианту. Создать интерактивное консольное приложение для ОС *Windows* на языке *C/C++* с возможностью выбора команды: а) ввод данных в стек(и) из стандартного или нестандартного текстового файла(ов); б) вывод данных из всех стеков на экран (стандартный текстовый файл); в) решение задачи; г) освобождение стеков; д) выход.

Пример. Дан стек, хранящий информацию о прохождении теста в виде пар (ФамилияИО, балл). Баллы лежат в пределе от 0 до 100. Найти средний балл за тест. Если стек пуст, средний балл сделать равным -1. Результат вычисления вернуть как результат функции. (полностью расширенный Пример см. ниже)

<table>
<tr><td colspan="2">C/C++ (для выполнения этой лабораторной)</td><td>Delphi (на экзамене можно выбрать язык)</td></tr>
<tr><td>typedef struct {
  char FamIO[31];
  int Ball;
}TInfo;
// или без typedef →</td><td>struct TInfo{
  char FamIO[31];
  int Ball;
};</td><td>Type
  TInfo=record
  FamIO: String[30];
  Ball: integer;
  End;</td></tr>
<tr><td colspan="3">// моделирование стека на основе односвязного списка</td></tr>
<tr><td colspan="2">// достаточно одного рекурсивного типа
struct TElem {
  TInfo Info; // или struct TInfo Info; в C
  TElem *Next; // или struct TElem *Next;
}; // здесь и далее (зависит от версии Си)</td><td> PElem=^TElem; // нужны два типа
  TElem=record // взаимно рекурсивных
  Info: TInfo;
  Next: PElem;
  End;</td></tr>
<tr><td colspan="3">// дополнительная процедура перекладывания элемента из одного стека (StackTop) в другой(Dop)</td></tr>
<tr><td colspan="2">void TopToTop (TElem **PSt1, TElem **PSt2){
  TElem *Elem, *StTop=*PSt1, *Dop=*PSt2;
  Elem = StTop;
  StTop = StTop->Next; // или StTop=(*StTop).Next;
  Elem->Next = Dop;
  Dop = Elem;
  *PSt1 = StTop; *PSt2 = Dop;
  return;}</td><td>procedure TopToTop(var StackTop,Dop:PElem);
var Elem: PElem;
begin
  Elem:=StackTop;
  StackTop:= StackTop^.next;
  Elem^.Next:= Dop;
  Dop:=Elem;
end;</td></tr>
</table>

//найти средний балл	

double naiti(TElem *StackTop){
TElem *Dop = NULL;
int **Sum = 0;**
int **N = 0**;
 	
while (StackTop) { // 1. Разбор стека
Sum += StackTop->Info.Ball;
N++;
TopToTop(&StackTop, &Dop);
}
 	
while (Dop) { // 2. Возвращение элементов
TopToTop(&Dop, &StackTop);
}
 	
if (N>0) return Sum/(double)N;
else return -1;
}
	function naiti(StackTop: PElem):real;
	var
	Dop: PElem; // вершина дополнит. стека
	Sum: integer; // сумма баллов
	N: integer; // количество испытуемых
	Begin
	Dop:=nil; **Sum:=0; N:=0;**
	While StackTop<>nil do {1. Разбор стека }
	Begin
	Sum:=Sum+ StackTop^.Info.Ball; Inc(N);
	TopToTop(StackTop, Dop);
	End;
	
	While Dop<>nil do{2.Возвращение элементов
	}
	TopToTop(Dop, StackTop);
	
	If N>0 then Naiti:=Sum/N else Naiti:=-1;
	End;

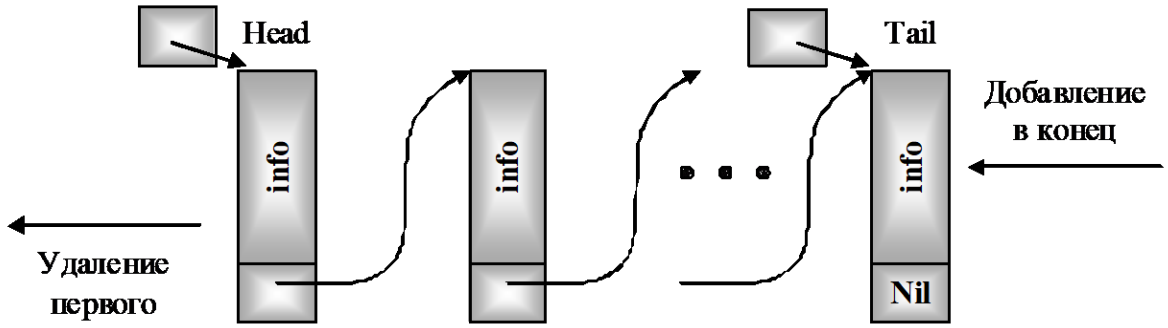
Абстрактные типы данных Очередь (FIFO) и Дэк (DEQ).

Очередь

(Информационно – логическая структура с порядком доступа FIFO)

(FIFO - First In First Out)

На основе односвязного линейного списка ее можно изобразить так:



Данные:

Адрес **головы** очереди,

Адрес **хвоста** очереди.

Допустимые операции:

Создать очередь из 1 элемента;

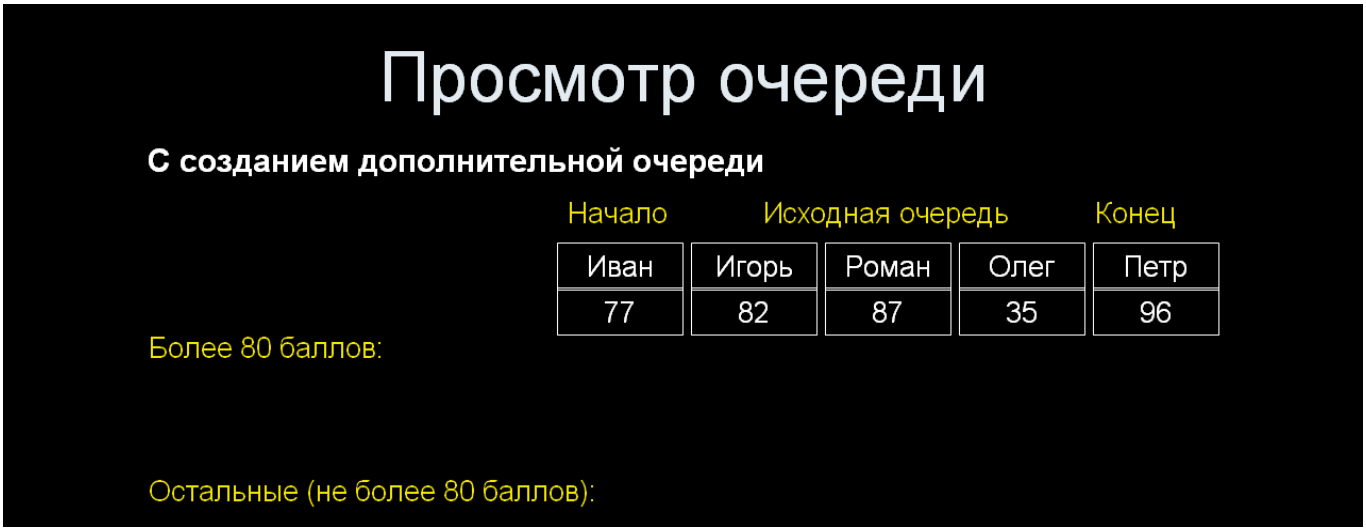
Добавить элемент в хвост очереди;

Удалить элемент из головы очереди;

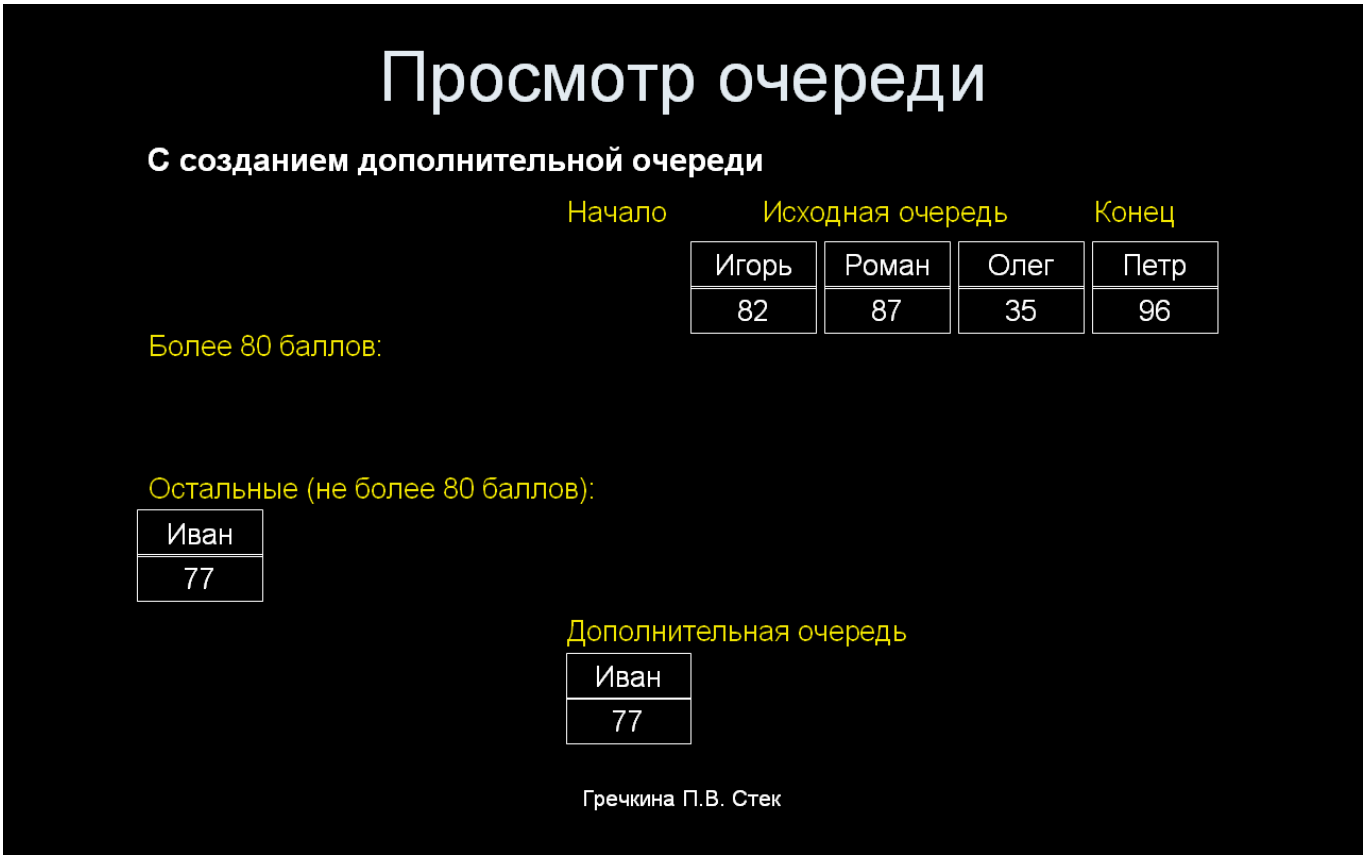
Проверка на отсутствие элементов (пустая очередь);

Узнать значение информационной части первого элемента.

Для работы с элементами очереди используется перемещение элементов из головы в хвост, с запоминанием адреса прежнего конца/начала очереди, чтобы не зациклиться, или использование дополнительной очереди.



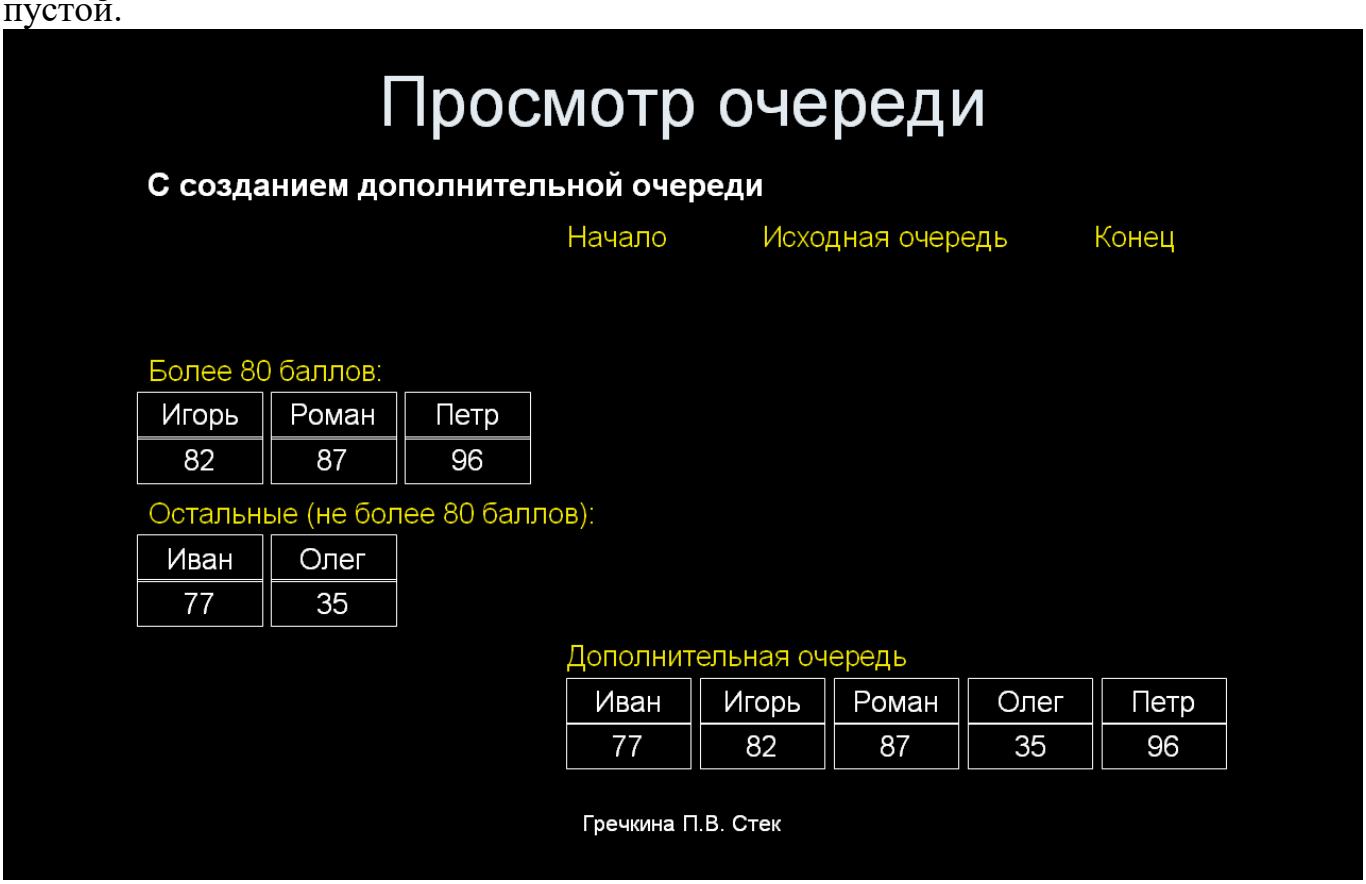
Рассмотрим пример разделения очереди на две: со студентами, получившими более 80 баллов, и со всеми остальными. Сначала пример с дополнительной очередью.



Начинаем всегда с первого элемента очереди .

У Ивана 77 баллов, значит, копируем информацию о нем в новый элемент и добавляем его в конец второй очереди и в дополнительную очередь, а сам элемент удаляем из исходной очереди, чтобы получить доступ к новому первому элементу очереди – Игорю. Игоря с 82 баллами копируем в первую очередь (для тех, кто набрал больше 80 баллов) и в дополнительную, удаляя из исходной и открывая доступ к Роману.

Таким образом, продолжаем обрабатывать исходную очередь, пока она не станет пустой.



Затем, по построению дополнительная очередь полностью совпадает с исходной. Поэтому можно ее считать исходной или перенести поэлементно данные обратно.

Просмотр очереди

С созданием дополнительной очереди

Более 80 баллов:

Игорь	Роман	Петр
82	87	96

Остальные (не более 80 баллов):

Иван	Олег
77	35

Начало Исходная очередь Конец

Иван	Игорь	Роман	Олег	Петр
77	82	87	35	96

Гречкина П.В. Стек

Цикл обработки очереди при этом выглядит так:

Просмотр очереди

С созданием дополнительной очереди

```
var
  ListDN, ListDK: PElem;    ball: byte;    chel: Tinfo;
begin
  ListDN:=nil; ListDK:=nil;
  FreeList( Res1Nachalo, Res1Konec);
  FreeList( Res2Nachalo, Res2Konec);

  while NachaloQueue<>nil do
  begin
    chel:=NachaloQueue^.info;



    DelFirst( NachaloQueue, KonecQueue);
    Dobavit( ListDN, ListDK, chel);
  end;
  NachaloQueue:=ListDN;    KonecQueue:=ListDK;
End;
```

Гречкина П.В. Стек

Остается только добавить строки, связанные с решением конкретной задачи разделения на две очереди по значению балла

Просмотр очереди

С созданием дополнительной очереди

```
var
  ListDN, ListDK: PElem;    ball: byte;    chel: Tinfo;
begin
  ListDN:=nil; ListDK:=nil;
  FreeList( Res1Nachalo, Res1Konec);
  FreeList( Res2Nachalo, Res2Konec);
  write('ball=?'); readln(ball);
  while NachaloQueue<>nil do
  begin
    chel:=NachaloQueue^.info;
    if chel.b > ball then
        Dobavit( Res1Nachalo, Res1Konec, chel)
    else       Dobavit( Res2Nachalo, Res2Konec, chel);
    DelFirst( NachaloQueue, KonecQueue);
    Dobavit( ListDN, ListDK, chel);
  end;
  NachaloQueue:=ListDN;    KonecQueue:=ListDK;
End;
```

Гречкина П.В. Стек

Можно просмотреть очередь и без создания дополнительной очереди, записывая данные в конец текущей очереди. Главное - не зациклиться при этом. Что возможно, особенно, если решение задачи связано с удалением элементов.

Рассмотрим этот способ на примере решения той же задачи. Начальное значение очереди то же:

Просмотр очереди

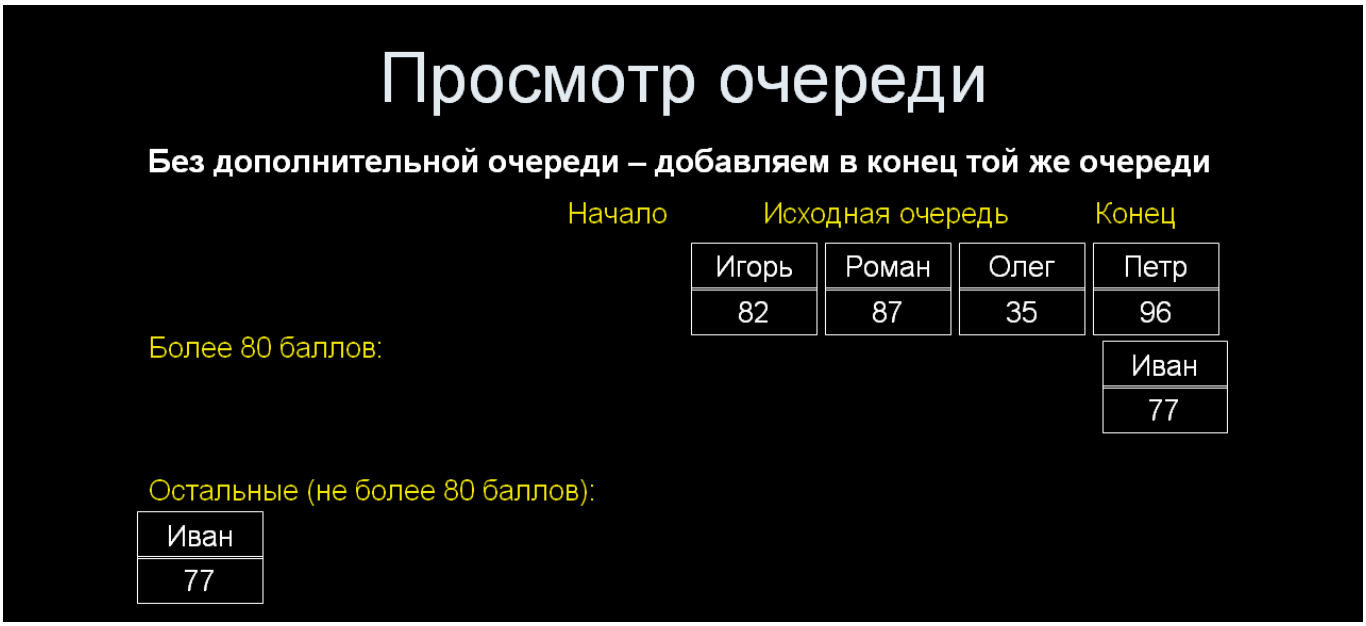
Без дополнительной очереди – добавляем в конец той же очереди

	Начало	Исходная очередь			Конец
	Иван	Игорь	Роман	Олег	Петр
	77	82	87	35	96

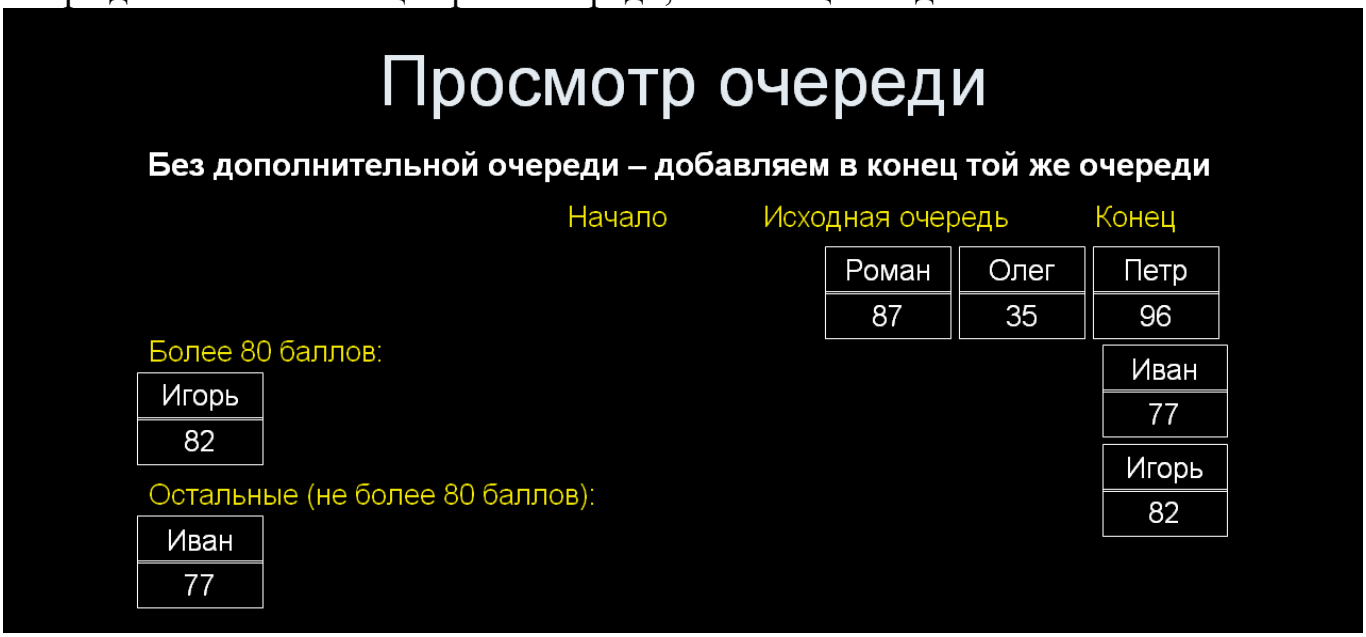
Более 80 баллов:

Остальные (не более 80 баллов):

Копию элемента Ивана добавляем во вторую очередь, а самого переставляем в конец исходной очереди:

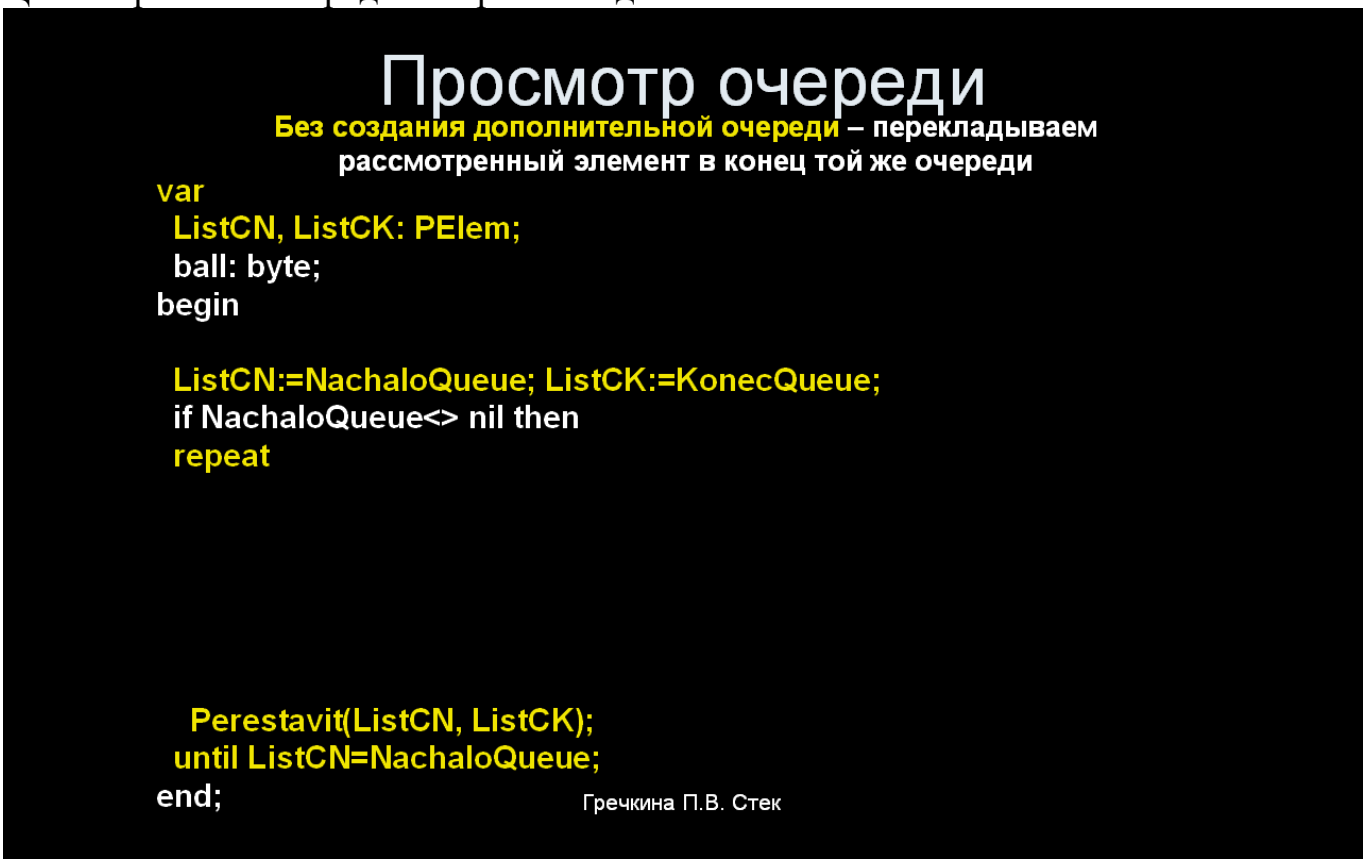


Игоря добавляем в конец первой очереди, и в конец исходной.



Рассматриваем все остальные элементы очереди, пока первым вновь не окажется Иван (Поскольку Иваны могут быть и другие, то определяем нужного по адресу элемента)

Цикл обработки очереди теперь выглядит так:



Добавляем в него код связанный с разделением очереди и получаем:

Просмотр очереди

Без создания дополнительной очереди – перекладываем рассмотренный элемент в конец той же очереди

```
var
  ListCN, ListCK: PElem;
  ball: byte;
begin
  ball:=UpDown2.Position;
  ListCN:=NachaloQueue; ListCK:=KonecQueue;
  if NachaloQueue<> nil then
  repeat

    if ListCN^.info.b > ball then
      Dobavit(Res1Nachalo, Res1Konec, ListCN^.info)
    else
      Dobavit(Res2Nachalo, Res2Konec, ListCN^.info);

    Perestavit(ListCN, ListCK);
  until ListCN=NachaloQueue;
end;
```

Гречкина П.В. Стек

Где процедура Perestavit переставляет первый элемент очереди в ее конец без изменения адреса элемента:

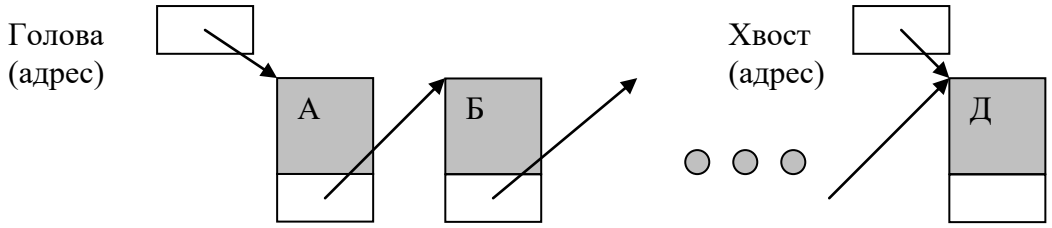
Просмотр очереди

Без создания дополнительной очереди – перекладываем рассмотренный элемент в конец той же очереди

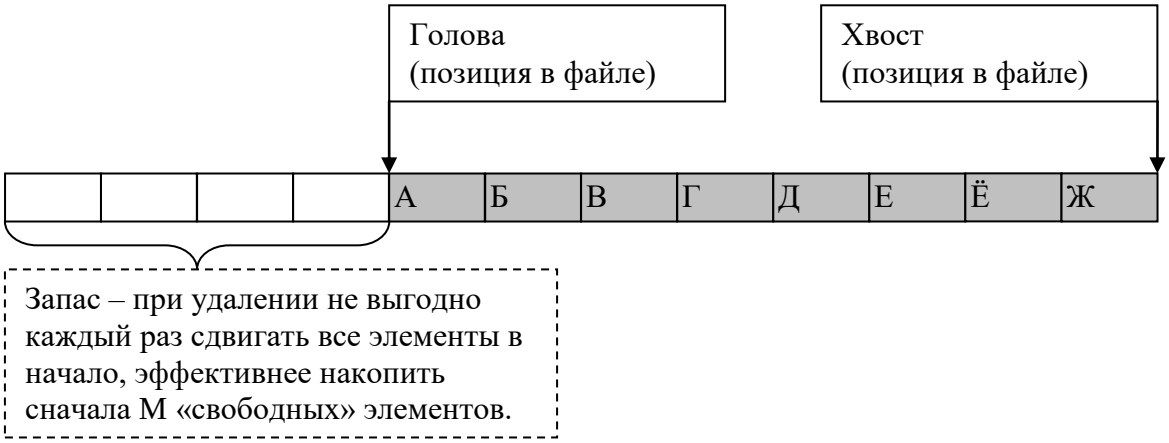
```
procedure Perestavit(var ListCN, ListCK: PElem);
begin
  if ListCN <> ListCK then
  begin
  ListCK^.next:=ListCN;
  ListCN:=ListCN^.next;
  ListCK:=ListCK^.next;
  ListCK^.next:=nil;
  end;
end;
```

В типовом расчете надо смоделировать очередь на основе двух разных структур данных:

1) на основе ссылочного типа «Линейный односвязный список»;



2) на основе типизированного/двоичного файла.

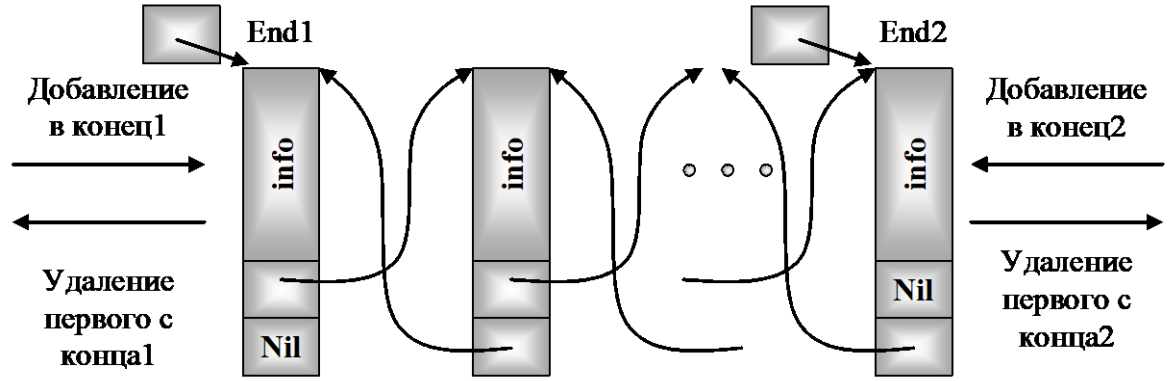


Дек (двухконечная очередь)

(Информационно – логическая структура с порядком доступа LIFO или FILO)

DEQ = Double Ended Queue

На основе двусвязного линейного списка дэк можно изобразить следующим образом



Данные:

Адрес головы (конца дека, играющего роль головы в данный момент),
Адрес хвоста (конца дека, играющего роль хвоста в данный момент).

Допустимые операции:

Создать дек из 1 элемента;
Поменять роль голова-хвост (поменять направление двухконечной очереди);
Добавить элемент в хвост дека;
Удалить элемент из головы дека;
Проверка на отсутствие элементов;
Очистить дек (удалить все элементы)
Узнать значение информационной головного элемента.

Для работы с элементами дека используется перемещение элементов из головы в хвост, с запоминанием прежнего конца дека, чтобы не зациклиться; а также можно использовать дополнительный дек.

Просмотр дека

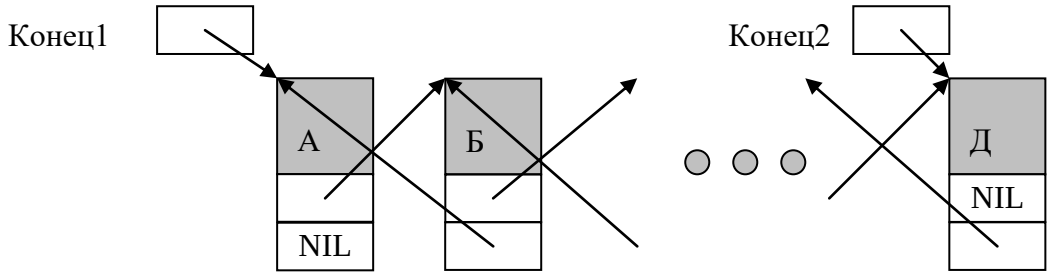
Просмотр выполняется аналогично просмотру очереди

А) с созданием дополнительного дека
Б) без него

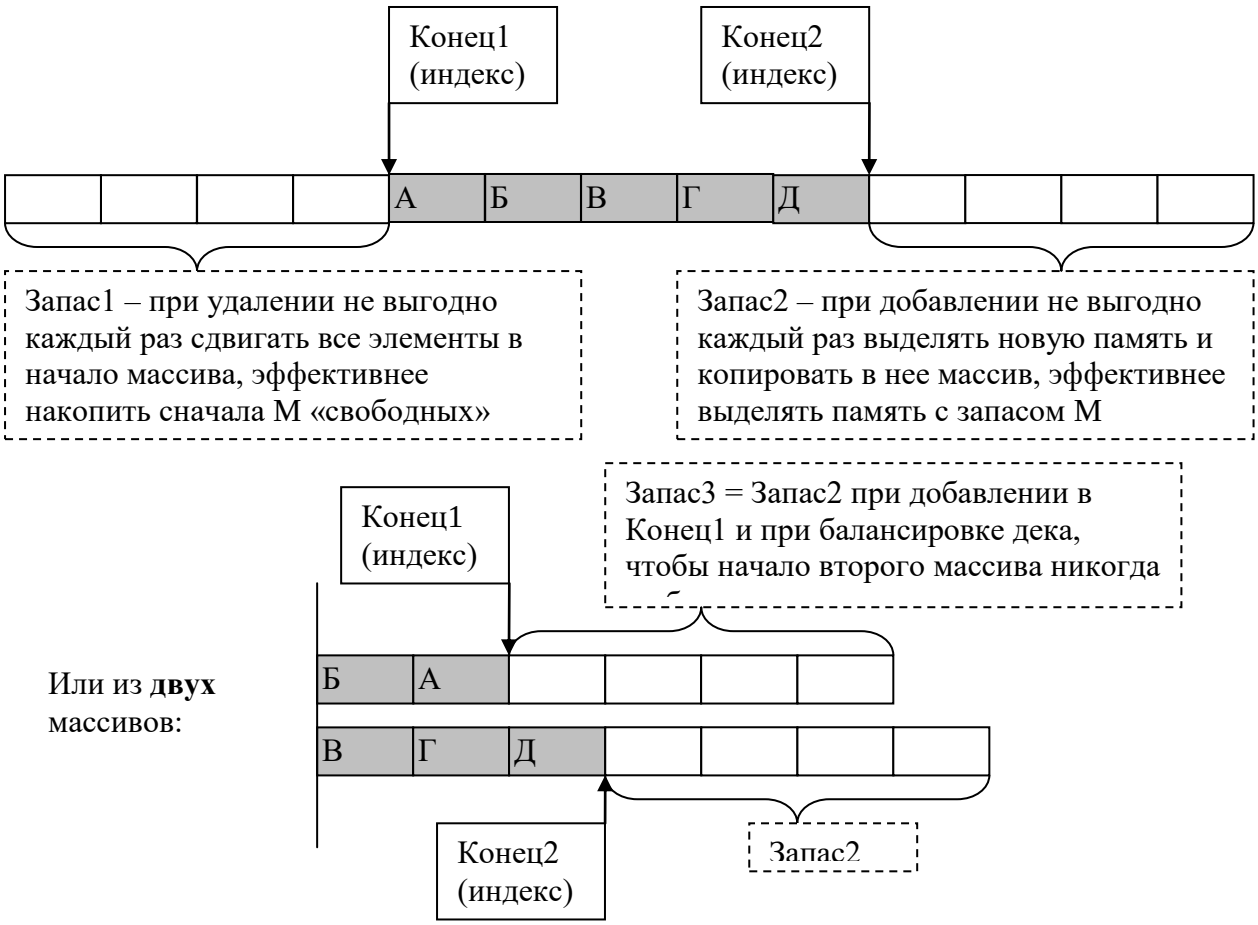
Отличие – можно просматривать очередь с ЛЮБОГО КОНЦА

Поэтому для удобства ДЕК моделируют на основе ДВУСВЯЗНОГО списка

В типовом расчете Дэк надо смоделировать на основе двух разных структур:
1) на основе ссылочного типа «Линейный двусвязный список»;



2) на основе динамического массива(ов).



И сделать вывод, на основе какой из структур это было делать удобней и эффективней.

В типовом расчете помимо абстрактных типов данных надо смоделировать такие типы данных как **Список и массив.**

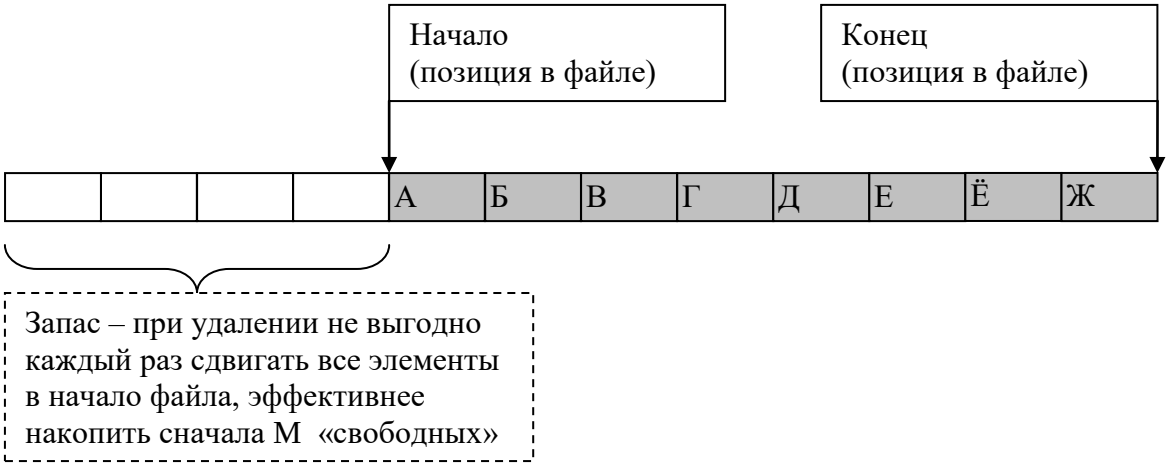
Список надо смоделировать без использования ссылочных типов списков (односвязного и двусвязного). Это позволит вам оценить неудобство отсутствия такого гибкого типа как ссылочный список, и Негибкость остальных типов.

Задание Б – «СПИСОК»

Часть Б1. Смоделировать двумя способами новый тип «Список»:

1) на основе одномерного динамического массива;

2) на основе типизированного/двоичного файла.

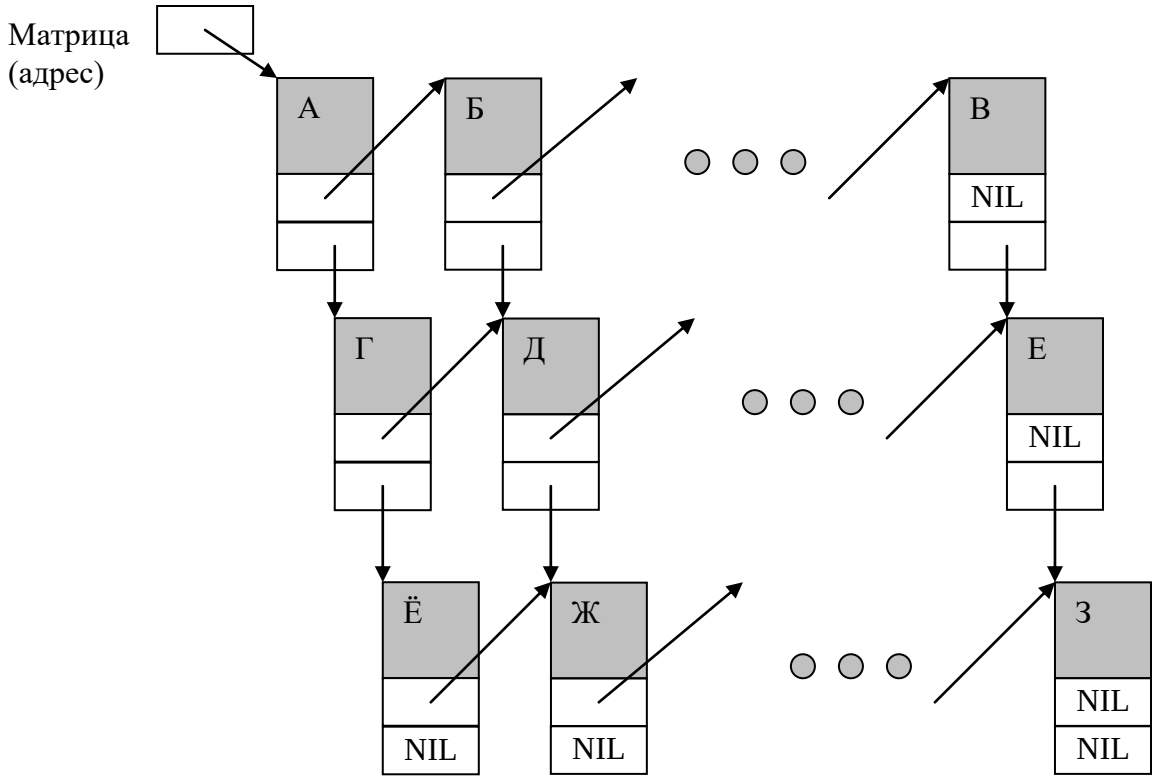


<u>Массив в свою очередь надо смоделировать без использования массивов!</u>

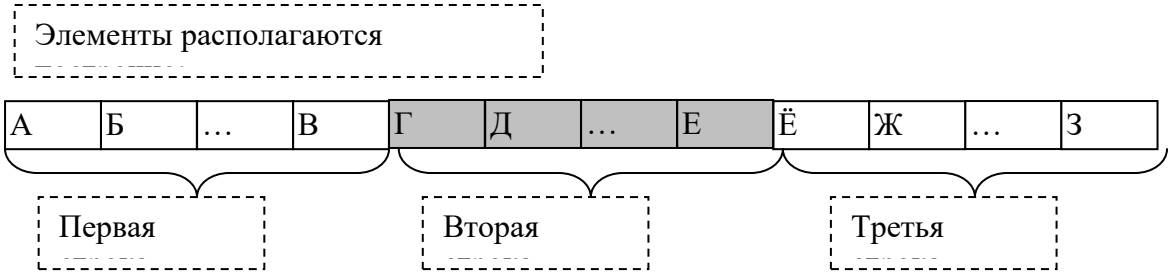
Задание Д – «МАТРИЦА»

Часть Д1. Смоделировать двумя способами новый тип «Матрица»:

1) на основе ссылочного типа «Нелинейный двусвязный список»;



2) на основе типизированного/двоичного файла.



Такое моделирование массива поможет лучше понять устройство массива, его недостатки и преимущества. Особенно второй способ, который соответствует размещению в памяти статических массивов.