Лекция 5.2. Динамические структуры. Стеки, очереди, деки

В данной лекции описаны различия между динамическими и статическими структурами. Описаны основные операции над стеком.

Статические структуры данных (СД), характеризуются:

- постоянством размера во время выполнения программы;
- память под них отводится во время компиляции для всех и она непрерывна;
- элементы структуры в этой области ОП смежны;
- отношения между элементами постоянны, достаточно просты и обычно хранятся в дескрипторах.

Полустатическая структура - это переменная текущая структура, которая имеет переменный размер, не превышающий заранее определенного верхнего предела.

Динамические структуры - это СД, для которых характерно:

- непостоянство и непредсказуемость размера (от 0 до верхнего предела памяти);
- отсутствие физической смежности элементов структуры в памяти;

Список

Динамические СД представляются списковыми структурами.

Список - это упорядоченная последовательность элементов данных E(1), E(2),..., E(n), где n > 0, причем, каждый из элементов характеризуется одним и тем же набором полей.

Если упорядоченная послед-ность линейна, то этой линейный список.

Упорядоченность можно задавать неявно (последнее расположение элементов списка, как в логической структуре, так и в памяти машины) - это **последоват. список** или последовательность.

Можно задать с помощью указателей (явно расположенных в эл-тах), дающих возможность для каждого эл-нта определить предшественника и последователя - это **динамически связный список**.

Т. о., список - это общее понятие, из кот. вытекают понятия остальных структур.

Так, при $n = const$ и соответствующих данных это будет вектор, массив, запись или таблица, а при $n \neq const$ и специфическом доступе - полустатические структуры:

стеки, очереди и деки.

Динамические массивы

1 - скорость доступа к данным в массиве;
2 - удобство программирования и лаконичность кода;
3 - ошибки при доступе к массивам.

	Память	Скорость	Удобство	Ошибки
Статические	ограничена	быстрее	проще	нет
Динамические	Не ограничена	медленнее	сложнее	есть

Избежание ошибок при работе с динамическими массивами

- При необходимости увеличить память, увеличивать ее вдвое или «большим куском» (10,20,100...)
- Не забывать освобождать память
- Следить за выходом индексов за границы выделенной памяти

Стек

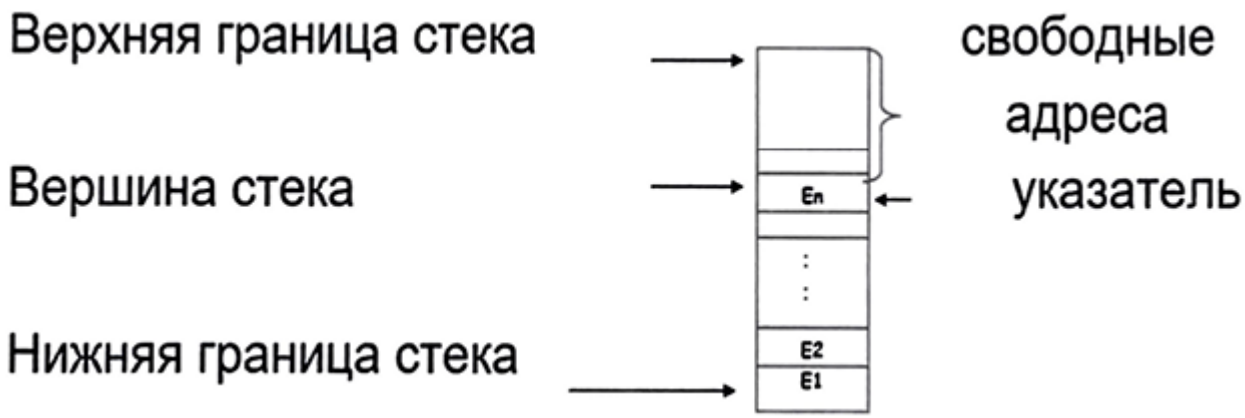
Стек - это последовательный односвязный список с переменной длиной, включение и исключение из которого происходят с одной стороны - его вершины.

Стек функционирует по принципу:
последним пришел - первым вышел, то есть, Last In - First Out (LIFO).

Примеры: Стакан, винтовочный патронный магазин, железнодорожный разъезд. Существует также стек вызовов и стек команд.

Логически стек представляется:

- Верхняя граница стека
- Вершина стека
- Нижняя граница стека



Вершина стека адресуется специальным указателем Point Stack (PS)

Основные операции:

- включение элемента в стек,
- исключение элемента из стека
- очистка стека.

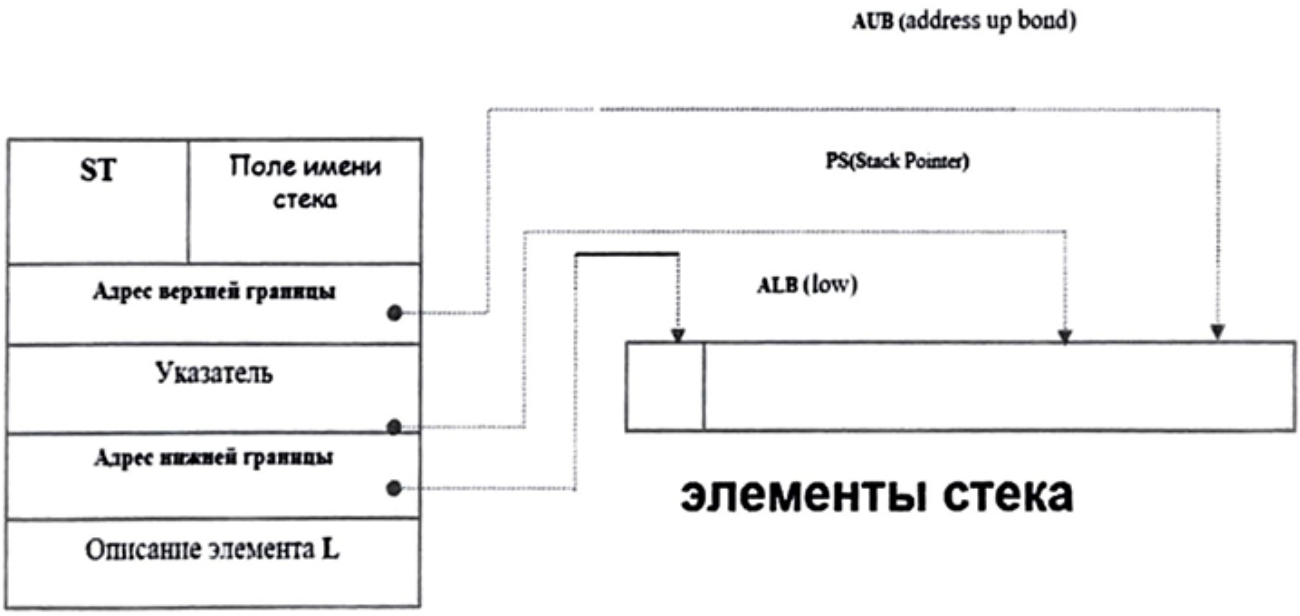
Реализация стека вектором: статическим или динамическим

При включении элемента (Push - заталкивать): сначала перемещается «вверх» указатель на длину типа данных, затем по значению указателя в стек помещается информация. Если указатель выходит за верхнюю границу стека, то стек переполняется.

При исключении элемента (Pop - выскакивать): по указателю стека прочитывается информация об исключаемом элементе, затем указатель смещается «вниз» на один элемент. Если указатель выходит за нижнюю границу стека, то стек пуст.

Для хранения стека отводится сплошная область памяти. Граничные адреса стека - это параметры физической структуры.

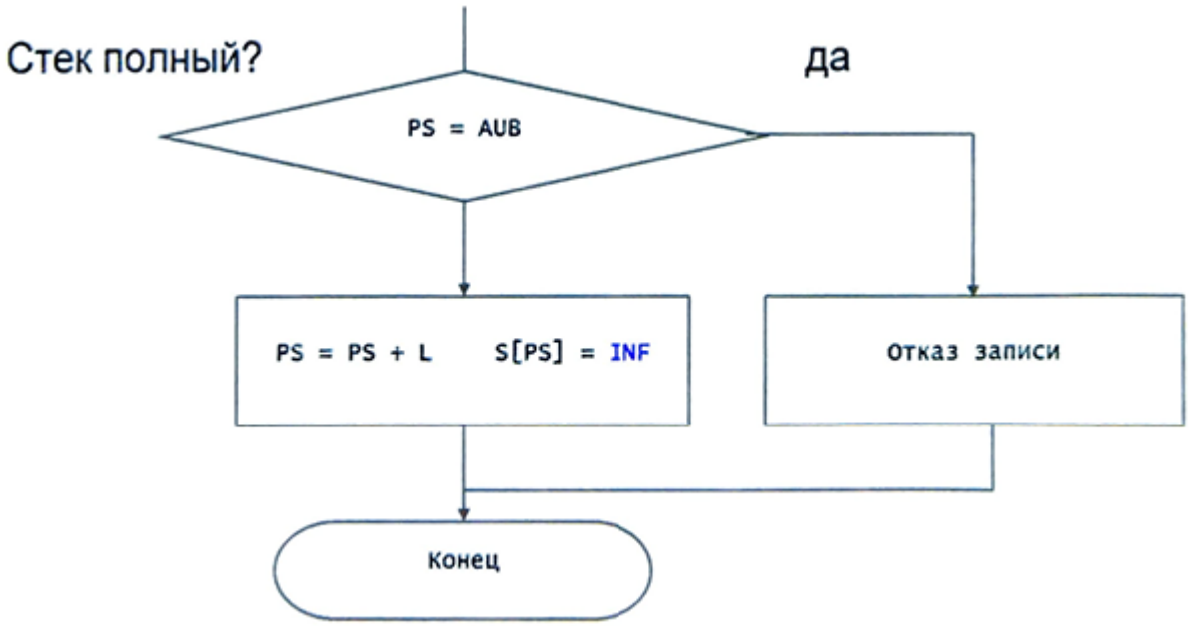
Физическая структура обычно дополняется дескриптором стека.



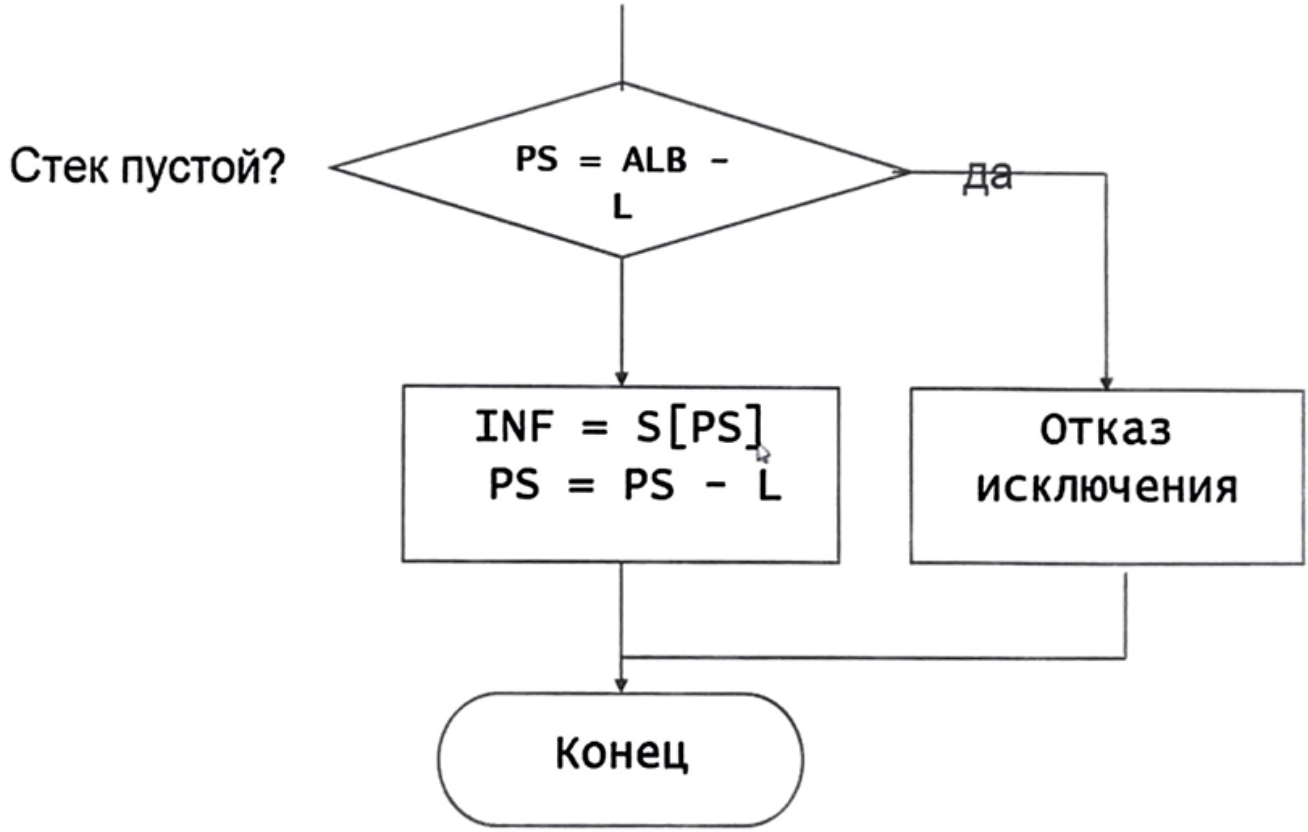
- `AUB` - адрес верхней границы (Address Up Bond)
- `PS` - (Pointer Stack) - указатель,
- `ALB` - адрес нижней границы (Low)
- `L` - элемент стека

Блок-схема алгоритма включения элемента в стек

Если стек реализован на основе вектора, то при
работе доступен только верхний элемент и блок-схемы алгоритмов операций будут такими:



Блок-схема алгоритма исключения элемента из стека

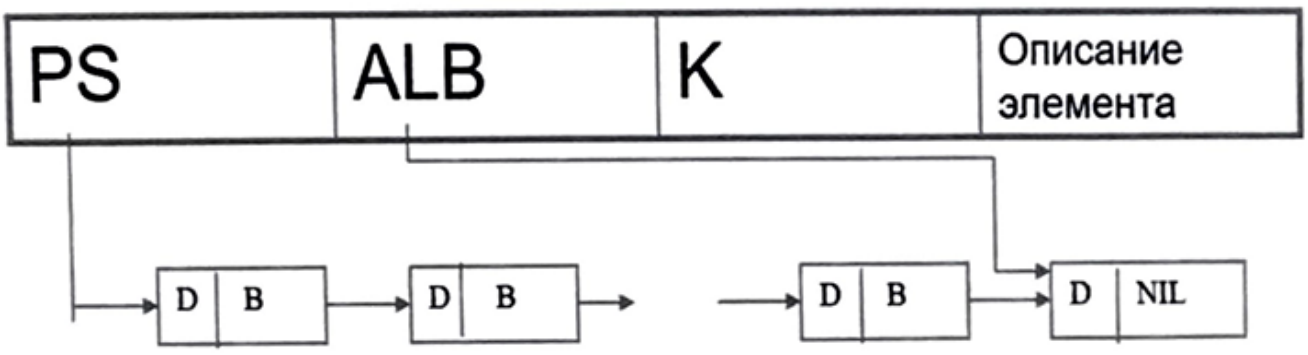


Операция очистки стека реализуется установкой вершины адреса вне стека, то есть, ниже нижней границы: `PS = ALB - L`

Операция определения количества элементов в стеке (проверка его объема): `n = (PS - ALB) / L + 1`

Стек реализован в виде односвязного линейного списка

вид дескриптора



Где

- `K` - текущее количество элементов,
- `D` - данные,
- `B` - адрес предыдущего элемента

При этой реализации теоретически объем стека ограничен только объемом доступной оперативной памяти

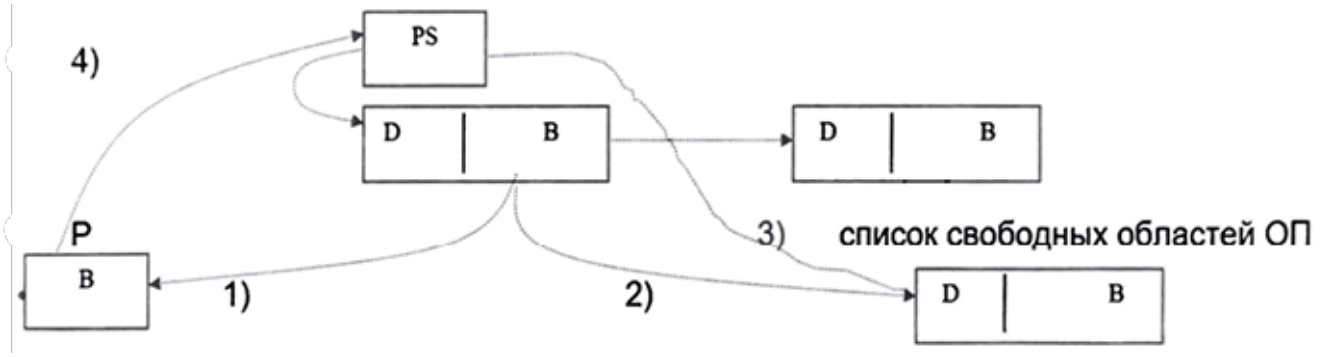
Операция включения элемента с адресом N в стек

- (если элемент создается, то происходит выделение памяти под элемент).
- Исходное состояние дескриптора стека:
- `PS <- Nil`; `ALB <- Nil`; `k <- 0`;
- `if k = 0 Then ALB <- N` {первый и он же последний элемент стека}
- `B(N) <- PS`; {в адрес «предыдущего» для нового элемента заносим адрес из `PS`}
- `PS N`; {в адрес указателя стека записываем адрес нового элемента}
- `k <- k + 1`; {увеличиваем количество элементов на единицу}

(обозначения в описании алгоритмов похожа на Паскаль)

Операция исключения элемента из стека

(с пополнением списка свободных областей памяти)




```
if к <> 0
    then
        печать D(Ps)
        1) P <- B (PS); {сохраняем адрес предыдущего элемента}
        2) B(PS) <- E;  {исключаемый элемент указывает на свободную область}
        3) E <- PS;      {адрес освободивш. элемента заносим в список
                          свободной области}
        4) PS <- P;     {изменим указатель стека на предыдущий}
        к <- к-1 else  {уменьшаем количество элементов на единицу}
else
    {стек пуст}
```

Очистить стек

- весь

```
B(PS) <- E;
E     <- PS;
PS    <- Nil;
ALB   <- Nil;
k     <- 0;
```

- поштучно (долго):

```
While B(PS)<> Nil
    Do B(PS) <- E
       E <- PS
PS  <- Nil
ALB <- Nil
K   <- 0
```

Операции со стеком

выполняются за время $O(1)$

- Основные операции:
  - Включение нового эл-та `Push(X, Stack)`;
  - Исключение эл-та `Pop(Stack)`;
- Вспомогат. операции:
  - Определение текущ. числа эл-тов стека;
  - Очистка стека;
  - Неразрушающее чтение эл-та из вершины стека, м.б. реализовано как комбинация основных операций: `X <- Pop(Stack); Push(X,Stack);`