

Линейные списки

Линейные списки

Линейным списком называется упорядоченная структура, каждый элемент которой связан с соседним элементом. Наибольшее распространение получили два вида линейных списков: стеки и очереди.

Стек это список типа LIFO (последним пришел – первым вышел). Стек имеет одну точку доступа, которая называется вершиной.

Аналогом стека является стопка книг, в которой дополнение и изъятие книг происходит сверху.

Другим примером может служить обойма с патронами (магазин), в которой зарядка и подача для стрельбы выполняется с одного конца. Именно этим примером объясняется бывшее в употреблении русскоязычное название стека “магазин”.

В программировании через стеки передаются параметры при обращении к процедурам. Если имеется цепочка вызова функций, то локальные переменные могут сохраняться в стеке, который расширяется при загрузке функции и сокращается при возврате из нее.

Иногда стеки реализуются аппаратным образом. В Ассемблере имеется регистр стека и соответствующие команды для работы с ним.

Стеки: представление в ОП

Стеки могут представляться как в статической, так и динамической памяти. В статическом представлении стек задается одномерным массивом, величина которого определяется с запасом. Пусть он описан в виде `Var Stack[1..N] of T`, где `T` – тип элементов стека. Вершина стека задается индексом массива `Top`.

Дополнение в стек (`push`) производится командами

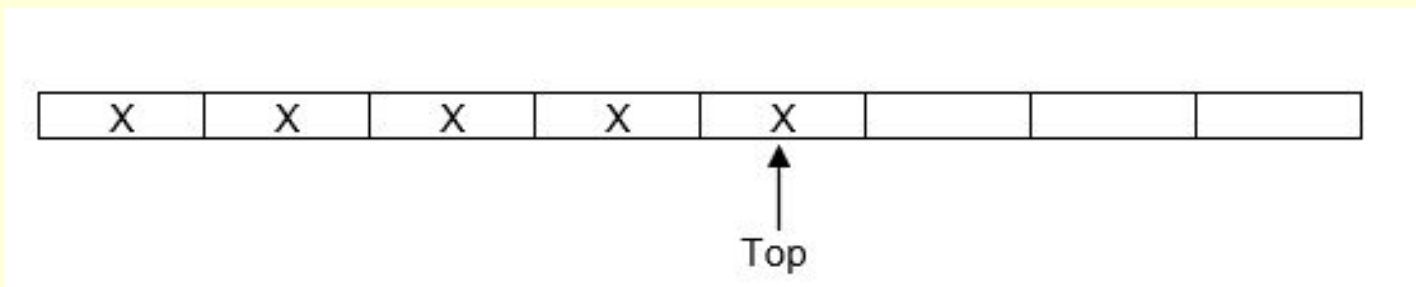
```
Top := Top+1;
```

```
Stack[Top] := NewElement;
```

Удаление из стека (`pop`) выполняется командой `Top:= Top-1`.

Для обработки возможных ошибок при дополнении необходимо проверять выход за границы массива, а при удалении проверять непустоту стека.

Признаком пустого стека при индексации с 1 является условие `Top = 0`.



Стеки: представление в ОП

В динамической памяти стек представляется в виде

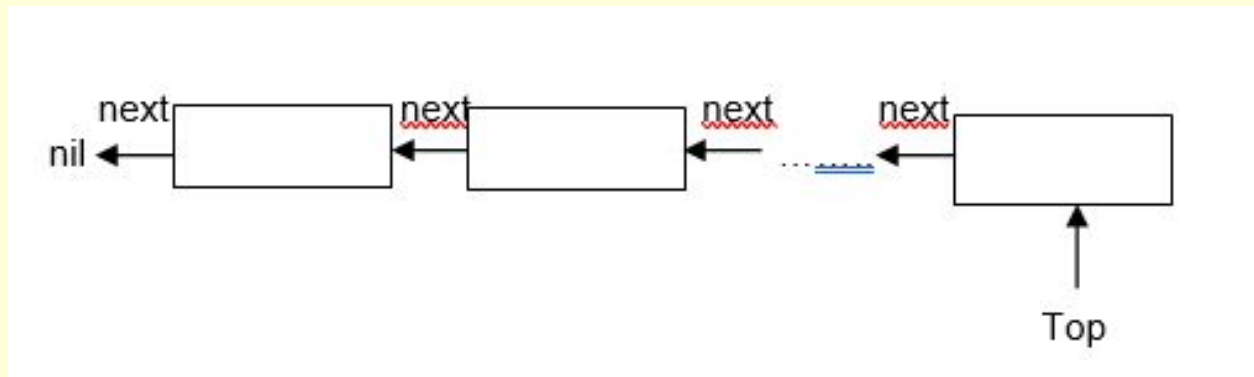
Type

```
Ukaz = ^Stek;  
Stek = record  
    name: string;  
    next: ukaz;  
end;
```

Var

```
Top, Kon: ukaz;
```

Здесь в качестве примера информационная часть элементов описана переменной name, а указатель next обеспечивает связь с предыдущим элементом.



Операции со стеками

Включение в стек (push) реализуется командами

```
New(Kon);      { создание элемента, на который указывает Kon }
```

```
Kon^.next:=Top;
```

```
Kon^.name:=NewName;
```

```
Top:=Kon;
```

Удаление из стека (pop):

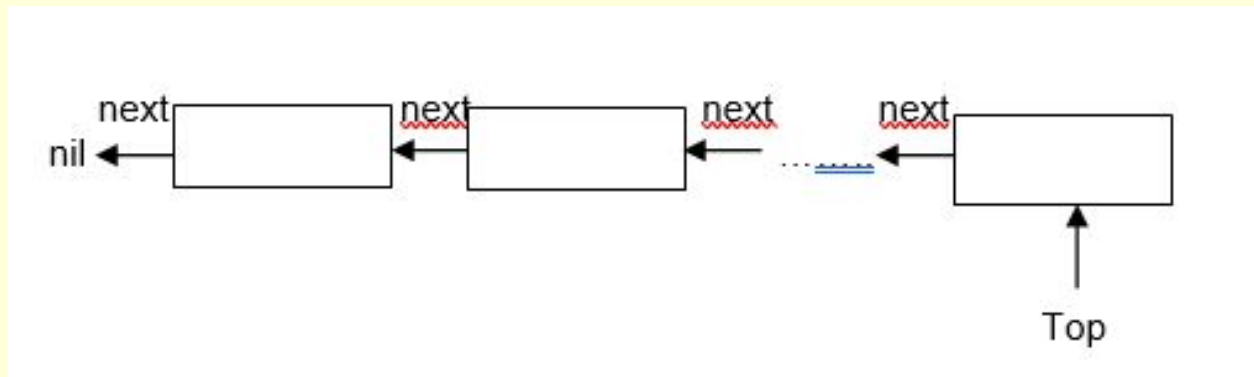
```
Kon:=Top;
```

```
Top:=Top^.next;
```

```
Dispose(Kon);  { удаление бывшей вершины стека }
```

Операции push и pop обычно оформляют в виде функций или процедур.

Признаком пустого стека является условие $Top = nil$.



Операции со стеками

Обычно изменение стека идет через его вершину. Для демонстрации гибкости динамических структур рассмотрим другую задачу. Имеется стек, описанный ранее. Требуется вставить элемент с именем `NewName` после элемента `KeyName`.

Пусть переменные `P` и `Q` имеют тип `Ukaz`, а `B` имеет тип `boolean`. Необходимая корректировка данных показана на рисунке и выполняется так:

```
P:=Top; B:=true;
```

```
While (P<>nil) and B do
```

```
  if P^.Name = KeyName then
```

```
    Begin
```

```
      B:=false; {для выхода из цикла}
```

```
      New(Q);
```

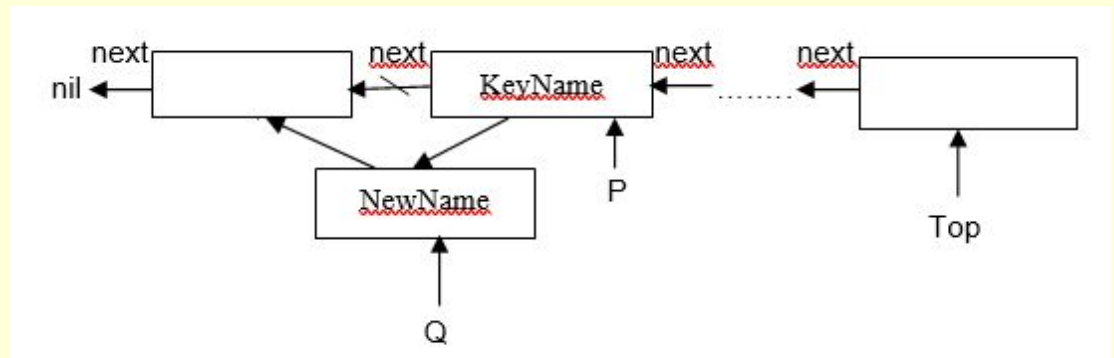
```
      Q^.Name:= NewName;
```

```
      Q^.next:=P^.next;
```

```
      P^.next:=Q;
```

```
    End
```

```
  else P:= P^.next;
```



Операции со стеками

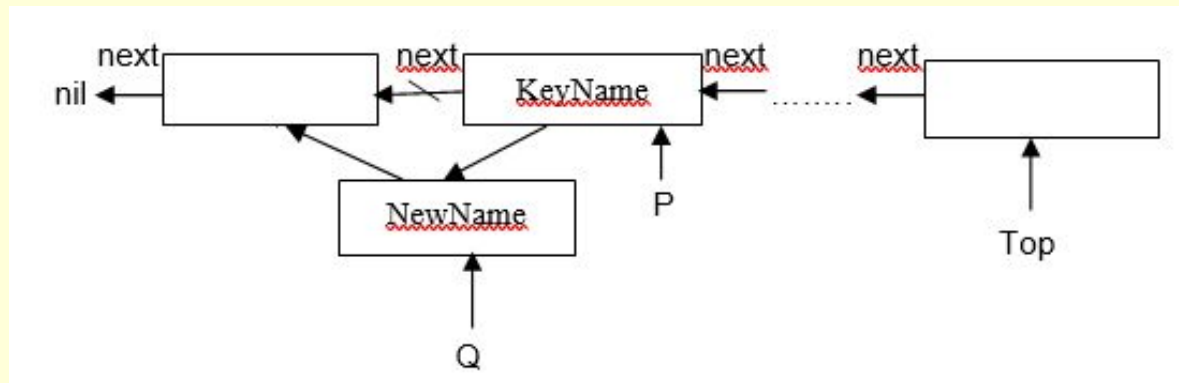
А теперь видоизменим задачу. Пусть вставка требуется перед элементом `KeyName`. На первый взгляд кажется, что сейчас придется сохранять указатель на предыдущий элемент либо анализировать вместе с текущим и следующий элемент, но указатели позволяют выбрать более простое и красивое решение. Изменения касаются последних трех операторов, следующих после `New(Q)`.

`Q^ := P^;`

`P^.Name := NewName;`

`P^.next := Q;`

Первый оператор обеспечивает копирование элемента `KeyName` на место вновь организуемого элемента. При этом автоматически обеспечивается связь с элементом, стоявшим ранее после `KeyName`, так как поле указателя `next` также копируется. Остается откорректировать старый элемент `KeyName`.



Формы представления алгебраических выражений

Постфиксная запись представляет собой такую запись алгебраического выражения, в которой сначала записываются операнды, а затем – знак операции.

Например, для выражения $a + b * c$ постфиксная запись будет $a b c * +$. Здесь операндами операции $*$ будут b и c (два ближайших операнда), а операндами операции $+$ будут a и составной операнд $b c *$.

Эта запись удобна тем, что она не требует скобок. Например, для выражения $(a + b) * c$ постфиксная запись будет $a b + c *$. В этой записи не требуется ставить скобки для того, чтобы изменить порядок вычисления, зависящий от приоритета операций, как в исходном выражении. Поэтому постфиксная запись удобна для вычисления алгебраических выражений и широко применяется на практике.

В **префиксной записи** сначала наоборот записывается знак операции, а затем операнды. Например, для выражения $a + b * c$ префиксная запись будет $+ a * b c$.

Префиксная запись также не требует расстановки скобок. Префиксную форму называют еще **польской записью**, а постфиксную – **обратной польской записью**.

Формы представления алгебраических выражений

Привычная форма записи со скобками называется **инфиксной**.

Выражение может включать как двуместные операции, так и одноместные. Примерами одноместных операций могут быть функции.

Вычислить значение выражения, записанного в постфиксной записи, очень просто. Требуется единственный последовательный просмотр символов (лексем) выражения.

Просматриваем постфиксную запись. Значения переменных и констант кладутся в стек. Когда встречается операция, из стека берутся два верхних (последних) значения, вычисляется результат применения операции к этим значениям, и результат помещается в стек. Если встречается функция, то берётся одно значение из стека, а результат помещается в стек на его место.

Например, выражение в постфиксной форме **a b c + * d - sin** соответствует инфиксной форме **sin (a * (b + c) - d)** и вычисляется в порядке, задаваемом скобками.

Алгоритм Дейкстры преобразования выражения из инфиксной формы в постфиксную

Алгоритм Дейкстры перевода в постфиксную запись обрабатывает исходный массив лексем и строит новый массив из тех же лексем, расположенных в другом порядке. Кроме того, необходим еще стек – аналогичный массив, используемый для временного хранения операций.

Операции имеют разные приоритеты. Наименьший приоритет у операций '+' и '-'. Более высокий приоритет имеют операции '*' и '/'. Еще более высокий приоритет у операции возведения в степень '^'. Самый высокий приоритет имеют операции, задаваемые функциями, такими как 'sin', 'cos', 'exp'. Заметим, что для этих операций требуется единственный операнд.

Если операнд представляет собой выражение с другими знаками операций, то он должен заключаться в скобки.

Алгоритм Дейкстры преобразования выражения из инфиксной формы в постфиксную

1. Константы и переменные \rightarrow запись.
2. Операция:
 - 1) стек пустой или вершина '(' , операция \rightarrow стек;
 - 2) приоритет операции больше, чем в вершине - операция \rightarrow стек;
 - 3) левоассоциативная ('+', '-', '*', '/') - операции из стека \geq приоритета до '(' или до конца стека \rightarrow запись, новая операция \rightarrow стек;
 - 4) правоассоциативная ('^', 'sin', 'cos', 'exp') - операции из стека $>$ приоритета до '(' или до конца стека \rightarrow запись, новая операция \rightarrow стек;
3. '(' - операция \rightarrow стек.
4. ')' - стека \rightarrow запись все операции до ближайшей '(' , '(' удаляется из стека.
5. Конец выражения - стека \rightarrow запись.

Алгоритм Дейкстры: пример 1

$a - b + c$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
a	a		1
-	a	-	2-1)
b	a b	-	1
+	a b -	+	2-3)
c	a b - c	+	1
Конец	a b - c +		5

Алгоритм Дейкстры: пример 2

$a + b - c * d$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
a	a		1
+	a	+	2-1)
b	a b	+	1
-	a b +	-	2-3)
c	a b + c	-	1
*	a b + c	- *	2-2)
d	a b + c d	- *	1
Конец	a b + c d * -		5

Алгоритм Дейкстры: пример 3

$a + b * c - d$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
a	a		1
+	a	+	2-1)
b	a b	+	1
*	a b	+ *	2-2)
c	a b c	+ *	1
-	a b c * +	-	2-3)
d	a b c * + d	-	1
Конец	a b c * + d -		5

Алгоритм Дейкстры: пример 4

(a + b * c) / 2

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
((3
a	a	(1
+	a	(+	2-1)
b	a b	(+	1
*	a b	(+ *	2-2)
c	a b c	(+ *	1
)	a b c * +		4
/	a b c * +	/	2-1)
2	a b c * + 2	/	1
Конец	a b c * + 2 /		5

Алгоритм Дейкстры: пример 5

(a * (b + c) + d) / 2

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
((3
a	a	(1
*	a	(*	2-1)
(a	(* (3
b	a b	(* (1
+	a b	(* (+	2-1)
c	a b c	(* (+	1
)	a b c +	(*	4
+	a b c + *	(+	2-3)
d	a b c + * d	(+	1
)	a b c + * d +		4
/	a b c + * d +	/	2-1)
2	a b c + * d + 2	/	1
Конец	a b c + * d + 2 /		5

Алгоритм Дейкстры: пример 6

$a \wedge b \wedge c$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
a	a		1
^	a	^	2-1)
b	a b	^	1
^	a b	^ ^	2-4)
c	a b c	^ ^	1
Конец	a b c ^ ^		5

Алгоритм Дейкстры: пример 7

$\sin \cos a$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
\sin		\sin	1
\cos		$\sin \cos$	3-4)
a	a	$\sin \cos$	1
Конец	$a \cos \sin$		5

Алгоритм Дейкстры: пример 8

exp(a * ln b)

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
exp		exp	1
(exp (3
a	a	exp (1
*	a	exp (*	2-2)
ln	a	exp (* ln	2-2)
b	a b	exp (* ln	1
)	a b ln *	exp	4
Конец	a b ln * exp		5

Алгоритм Дейкстры: пример 9

$\sin a \wedge b$

Обрабатываемая лексема	Результат	Стек	Пункт алгоритма
\sin		\sin	2-1)
a	a	\sin	1
\wedge	$a \sin$	\wedge	2-4)
b	$a \sin b$	\wedge	1
Конец	$a \sin b \wedge$		5

Вычисление выражения $a b c + * d + 2 /$
 $a = 1, b = 2, c = 3, d = 4$

Обрабатываемая лексема	Стек
a	1
b	1 2
c	1 2 3
+	1 5
*	5
d	5 4
+	9
2	9 2
/	4.5

Вычисление выражения $x \times \sin^2 x + x = 3.14$

Обрабатываемая лексема	Стек
x	3.14
x	3.14 3.14
sin	3.14 0
2	3.14 0 2
*	3.14 0
+	3.14

Операции со стеком на C++

```
#include <iostream>
using namespace std;

struct St
{
    int key;
    St *next;
};

void push(St *&p, int elem); // включение в стек (p меняется)
void pop(St *&p);           // удаление из стека (p меняется)
void vivod(St *p);         // вывод содержимого стека на экран (p не меняется)
void clear(St *p);         // очистка стека (p не меняется)

int main()
{
    setlocale(LC_ALL, "rus");
    system("cls");
    St *top=0; // признак пустого стека
    int answer = 1;
```

Операции со стеком на C++

```
while (answer != 5)
{
    printf("\n1 Включение в стек");
    printf("\n2 Удаление из стека");
    printf("\n3 Выдача стека");
    printf("\n4 Удаление всего стека");
    printf("\n5 Конец");
    printf("\nВаш выбор? ");
    cin >> answer;
    switch (answer)
    {
        case 1: // Включение в стек
            int k;
            printf("Введите целое число ");
            cin >> k;
            push(top, k);
            break;
        case 2: // Удаление из стека
            if (top)
            {
                pop(top);
            }
            else printf("Стек пуст\n");
            break;
```


Операции со стеком на C++

```
case 3: // Вывод на экран
```

```
if (top)
```

```
{
```

```
    vivod(top);
```

```
}
```

```
else printf("Стек пуст\n");
```

```
break;
```

```
case 4: // Очистка стека
```

```
if (top)
```

```
{
```

```
    clear(top);
```

```
}
```

```
else printf("Стек пуст\n");
```

```
top = 0; // функция clear не возвращает top!
```

```
break;
```

```
case 5:
```

```
clear(top); // Сначала очистка стека
```

```
top = 0; // функция clear не возвращает top!
```

```
break;
```

```
}
```

```
}
```

```
return 0;
```

```
}
```

Благодарю за внимание!

