Лабораторная работа № 9

Работа со стеками и очередями

Теоретическая часть

Стек (Stack) - линейный список, в котором все включения и исключения (и обычно всякий доступ) производятся с одного конца списка. Элемент, вставленный в стек, делает недоступными все элементы, вставленные до него. Удаление элемента делает доступным элемент, вставленный предпоследним. Единственно доступным элементом в стеке является тот, который вставлен последним. Процесс помещения объектов в стек называется проталкиванием (pushing), а процесс извлечения верхнего элемента из стека называется выталкиванием (popping). Учитывая характер дисциплины обслуживания списка, стек называют списком типа LIFO ("last-in-first-out" - "последним-пришел первым-вышел"). Графически стеки чаще всего изображаются как вертикальные объекты: элементы располагаются снизу вверх, так что наверху оказывается элемент, вставленный последним.

Для стека определены следующие основные функции:

- *Push*(*S, x*); проталкивает элемент *x* в стек *S*.
- *Pop*(*S*); выталкивает элемент, находящийся на вершине стека *S*, и возвращает на него ссылку. Если стек пустой, то функция *Pop* возвращает **nil**.
- *Top*(*S*) (или *Peek*(*S*)); возвращает ссылку на элемент, находящийся на вершине стека *S*, не удаляя его из стека. Если стек пустой, то функция *Top* возвращает **nil**.
- *Clear*(*S*); удаляет все элементы из стека *S*.
- *Count*(*S*); возвращает число элементов, содержащихся в стеке *S*.
- *IsEmpty*(*S*); тестирует стек *S* на наличие в нем элементов и возвращает *True*, если стек пуст, и *False* в противном случае.

Очередь (Queue) - линейный список, в котором все включения производятся с одного конца, а все исключения (и обычно всякий доступ) производятся с другого конца списка. Идея очереди состоит в том, что ее первый элемент обрабатывается первым. Учитывая характер дисциплины обслуживания списка, очередь называют списком типа FIFO ("first-in-first-out" - "первым-пришел первым-вышел"). Графически очереди чаще всего изображаются как горизонтальные объекты: элементы располагаются слева направо, так что слева оказывается элемент, вставленный первым, а справа - элемент, вставленный последним. Используются термины начало и конец очереди, обозначающие левую и правую сторону очереди соответственно. Также используются термины голова (head) и хвост (tail) очереди. Когда элемент ставится в очередь, он занимает место в ее хвосте. Из очереди всегда выводится элемент, который находится в ее головной части.

Для очереди определены следующие основные функции:

- *Enqueue(Q, x*); добавляет элемент *x* в конец (правую сторону) очереди *Q*.
- *Dequeue*(*Q*); удаляет элемент из очереди *Q* и возвращает на него ссылку. Если очередь пустая, то функция *Dequeue* возвращает **nil**.
- *Peek*(*Q*); возвращает элемент, находящийся в начале очереди *Q*, но не удаляет его. Если очередь пустая, то функция *Peek* возвращает **nil**.
- *Clear*(*Q*); удаляет все элементы из очереди *Q*.
- *Count*(*Q*); возвращает число элементов, содержащихся в очереди *Q*.
- *IsEmpty*(*Q*); тестирует очередь *Q* на наличие в ней элементов и возвращает *True*, если очередь пуста, и *False* в противном случае.

Стеки широко применяются в программировании, например, для отслеживания точек возврата из подпрограмм. Языки программирования высокого уровня используют стек вызовов для передачи параметров при вызове процедур. Стеки используются при решении многих интересных с алгоритмической точки зрения задач, например, такой как: «Расставить 8 ферзей на шахматной доске, чтобы ни один из них не угрожал другому». Наиболее часто стеки используются в теории синтаксического анализа, компиляции и перевода.

Частным, но интересным случаем использования стеков является трансляция инфиксных арифметических выражений в постфиксные выражения и их интерпретация.

Обычный метод записи арифметических выражений, в которых знак бинарной операции записывается между операндами, известен под названием *инфиксной записи*. Однако существуют другие способы описания того, как нужно комбинировать арифметические величины. Одним из таких способов является так называемая *постфиксная польская запись*, разработанная польским математиком Я. Лукасевичем. В постфиксной польской записи знак операции следует сразу за ее операндами. Каждому выражению в инфиксной записи соответствует выражение в постфиксной польской записи.

Примеры.

Инфиксная запись	Постфиксная польская запись
A*B	AB*
A*B+C	AB*C+
(A+B)*C	AB+C*
A+B*(C+D)*(E+F)	ABCD+*EF+*+

Постфиксная польская запись часто используется интерпретаторами в качестве промежуточного языка. Интерпретатор читает постфиксное выражение слева направо и вычисляет его значение с помощью *стека* при сумматоре.

Алгоритм интерпретации постфиксной польской записи.

Вход. Постфиксное выражение *w* с правым концевым маркером $.
Выход. Значение выражения на дне стека.
Метод. Будем считать, что $ - маркер дна стека и стек пуст. Пусть *a* текущий входной символ.

```
while a <> $ do
begin
//читаем очередной символ;
        if  a – операнд then
                перенести его со входа в стек
        else if a – операция then
                применить ее к верхним элементам стека
                // число участвующих в операции операндов зависит от ее арности
end;
// В стеке находится одно единственное значение, а именно значение выражения
// (синтаксическая корректность постфиксной польской записи устанавливается  транслятором).
// Конец алгоритма
```

Для трансляции инфиксных арифметических выражения в постфиксные предложен ряд методов, самым известным из которых является метод Дейкстры.

Алгоритм Дейкстры.

Рассматривается грамматика инфиксных арифметических выражений в форме Бэкуса-Наура:

```
<выражение> ->
        <выражение> + <терм> |
        <выражение> – <терм> |
        <терм>
<терм> ->
        <терм> * <фактор> |
        <терм> / <фактор> |
<фактор>
        <фактор> ->
```

<фактор> ^ <первичное> |
<первичное>
<первичное> ->
(<выражение>) |
переменная

Каждой операции ставится в соответствие некоторый приоритет, а именно:

+, -	2
*, /	3
^	4

(Эти приоритеты отражают старшинство операций умножения и деления по сравнению со сложением и вычитанием и старшинство возведения в степень по сравнению со всеми остальными операциями.) Алгоритм преобразования основан на использовании *стека* и *очереди* и состоит в следующем:

Алгоритм Дейкстры.

Вход. Инфиксное выражение *w* с правым концевым маркером $, приоритеты и ассоциативность всех операций.
Выход. Если *w* корректное выражение, то постфиксная польская запись, в противном случае сообщение об ошибке.
Метод. Будем считать, что $ - маркер дна стека и стек пуст. Будем считать, что выходная лента организована в виде очереди и очередь пуста. Пусть *a* текущий входной символ, а *b* верхний символ стека.

```
while a <> $ do
begin
//читаем очередной символ;
    if a – атом (константа или идентификатор) then
        записать a на выходную ленту
    else if a - идентификатор функции then
        перенести его со входа в стек
    else if a = "(" then
        перенести его со входа в стек
    else if a = ")" then
    begin
        while b <> "(" do
            вытолкнуть b из стека и записать на выходную ленту;
        if b = $ then
            error()
    end
    else if a - оператор, например, OP then
    begin
        if OP левоассоциативный then
            while приоритет OP ≤ приоритета b do
                вытолкнуть b из стека и записать на выходную ленту;
        else if OP правоассоциативный then
            while приоритет OP < приоритета b do
                вытолкнуть b из стека и записать на выходную ленту;
        перенести оператор OP со входа в стек;
    end;
end;
вытолкнуть все символы из стека в выходную строку;
// В стеке должны были остаться только символы операторов; если это не так, значит в выражении не
// согласованы скобки.
// Конец алгоритма
```

Пример. Для входной строки A+B*C+(D+E)*F алгоритм вычисляет такую последовательность конфигураций:

Входная строка	Выходная лента	Стек (вершина стека справа)
A+B*C+(D+E)*F$		$
+B*C+(D+E)*F$	A	$
B*C+(D+E)*F$	A	$+
*C+(D+E)*F$	AB	$+
C+(D+E)*F$	AB	$+*
+(D+E)*F$	ABC	$+*
(D+E)*F$	ABC*+	$+
D+E)*F$	ABC*+	$+(
+E)*F$	ABC*+D	$+(
E)*F$	ABC*+D	$+(+
)*F$	ABC*+DE	$+(+
*F$	ABC*+DE+	$+
F$	ABC*+DE+	$+*
$	ABC*+DE+F	$+*
$	ABC*+DE+F*+	$

Задание

Написать программу, которая

1. Читает инфиксное арифметическое выражение и преобразовывает его в постфиксное выражение.
2. Читает значения величин переменных, входящих в состав выражения.
3. Вычисляет значение выражения и результат выводит на печать.

Дополнительные указания

1. Переменные обозначать строчными и заглавными буквами английского алфавита (т.е. всего 52 переменных).
2. Аддитивные и мультипликативные операции – левоассоциативные.
3. Префиксные унарные операции – правоассоциативные.
4. Расширить грамматику, включив в нее
   (a) одноместные (унарные) операции, например, префиксный минус,
   (b) вызовы стандартных функций, например, sin(x),
   (c) числовые константы.

Варианты

1. Операция возведения в степень – левоассоциативная. Приоритет операции возведения в степень выше приоритета префиксных унарных операций.
2. Операция возведения в степень – левоассоциативная. Приоритет операции возведения в степень ниже приоритета префиксных унарных операций.
3. Операция возведения в степень – правоассоциативная. Приоритет операции возведения в степень выше приоритета префиксных унарных операций.
4. Операция возведения в степень – правоассоциативная. Приоритет операции возведения в степень ниже приоритета префиксных унарных операций.