Практическая работа № 17, ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКОГО ВЫРАЖЕНИЯ

Постановка задачи

Создать программу, которая по введенной строке, состоящей из цифр (от 0 до 9), знаков арифметических операций «+», «–», «*», «/» и круглых скобок, преобразовывает исходное выражение в строку, представленную в постфиксной форме, а затем вычисляет данное выражение.

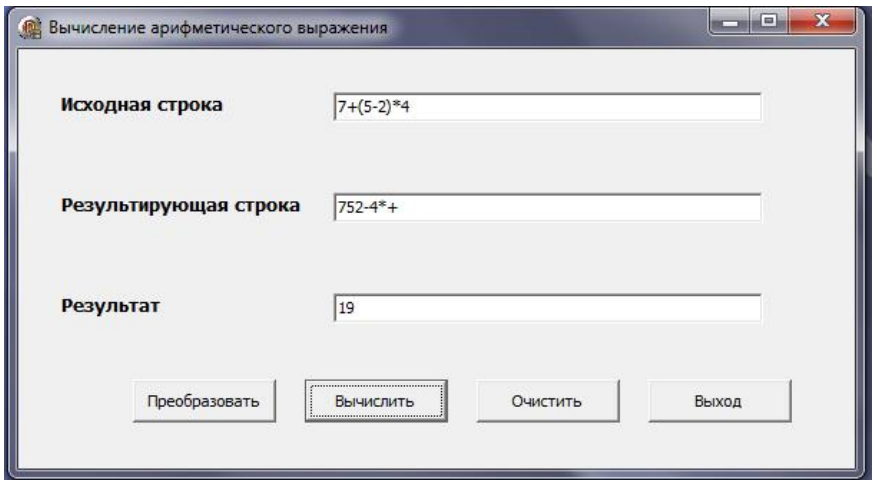


Рис.1

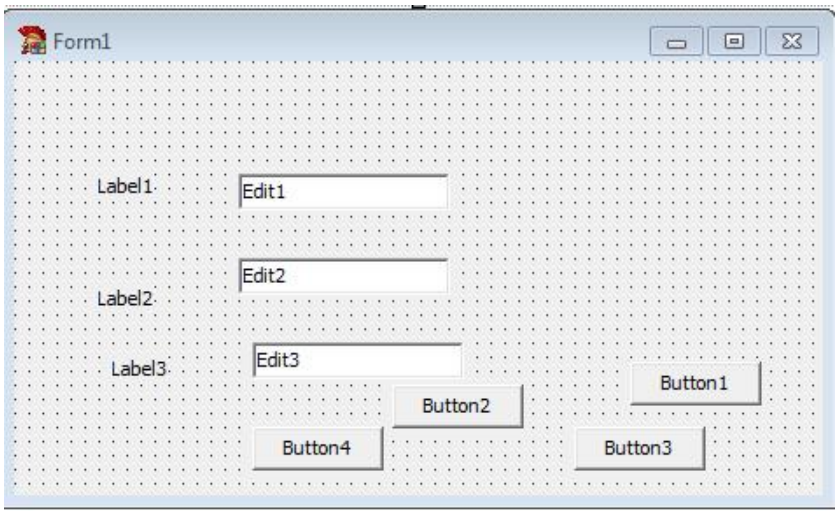


Рис.2

Новым в этой работе является:

- представление арифметических выражений в постфиксной форме (обратная польская запись) и вычисление арифметического выражения.

План разработки программы

1. Откройте новый проект.

2. Сохраните код программы и проект под именами, например, **Unit_18.pas** и **Pr_18.dpr**. В последующем сохраняйте проект и проверяйте его работоспособность после каждого изменения.

3. Разместите в форме объекты в соответствии с рис.2.

Не стремитесь располагать компоненты на свои места. Это мы сделаем в программе при создании формы. Для этого выделите объект **Form1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **OnCreat,** справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
// Описание формы
Form1.Width:=600;
Form1.Height:=330;
Form1.Caption:='Вычисление арифметического выражения';

...
```

Подобным образом необходимо описать все расположенные компоненты на форме. Для упрощения записи представим все свойства компонентов в виде трех таблиц.

Таблица 1

Компонента	**Свойства**				
	Caption	**Left**	**Top**	**Font.Style**	**Font.Size**
Label1	Исходная строка	30	30	fsBold	10
Label2	Результирующая строка	30	100	fsBold	10
Label3	Результат	30	170	fsBold	10

Таблица 2

Компонента	**Свойства**					
	Text	**Left**	**Top**	**Width**	**Height**	**Font.Size**
Edit1	"	220	30	300	20	11
Edit2	"	220	100	300	20	11
Edit3	"	220	170	300	20	11

Таблица 3

Компонента	**Свойства**				
	Caption	**Left**	**Top**	**Width**	**Height**
Button1	Вычислить	80	230	100	30
Button2	Очистить	200	230	100	30
Button3	Выход	320	230	100	30
Button4	Преобразовать	440	230	100	30

Немного теории

Представление арифметических выражений

Инфиксная форма	Префиксная форма	Постфиксная форма
A+B	+AB	AB+
A+B–C	–+ABC	AB+C–
(A+B)×(C–D)	×+AB–CD	AB+CD–×
A×B×C–D+E/F/(G+H)	+–×AB×CD/EF+GH	AB×C×D–EF/GH+/+
A–B/(C×D×E)	–A/B×C×DE	ABCD×E×/–

Примечание

1. Префиксная форма не является зеркальным отображением постфиксной.
2. При вычислении арифметических выражений в компьютере это выражение представляется в постфиксной форме.

1-ая часть программы (преобразование)

Исходная строка **A+B*C+(D*E+F)*G**

Постфиксная форма **ABC*+DE*F+G*+**

Алгоритм решения задачи с использованием стека

Анализируем символы исходной строки.

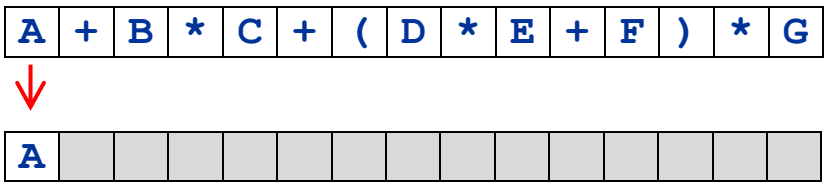
1. Если встретилось **число (или переменная)**, то помещаем его в результирующую строку.
2. Если символ – **знак операции** («+», «–», «*», «/»), то проверяем приоритет данной операции, а затем проверяем стек.

Приоритет операции

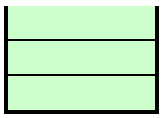
(	0
+, –	1
*, /	2

   1) если стек пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек;
   2) если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в результирующую строку до тех пор, пока выполняется это условие; затем переходим к пункту а).
3. Если текущий символ – **открывающая скобка**, то помещаем ее в стек.
4. Если текущий символ – **закрывающая скобка**, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку, которую следует просто уничтожить. Закрывающая скобка также уничтожается.
5. Если вся входная строка разобрана, а в стеке еще остаются знаки операций, переписываем все элементы стека в результирующую строку.

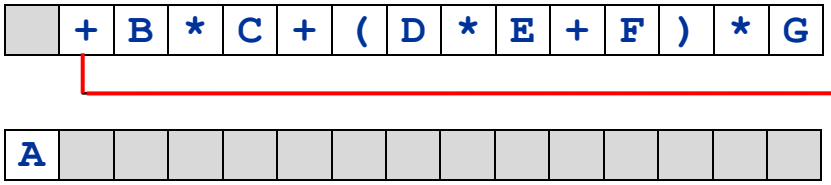
1 шаг



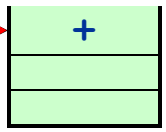
Стек



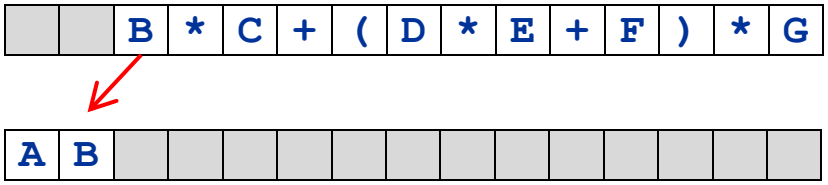
2 шаг



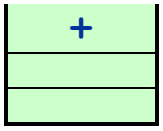
Стек



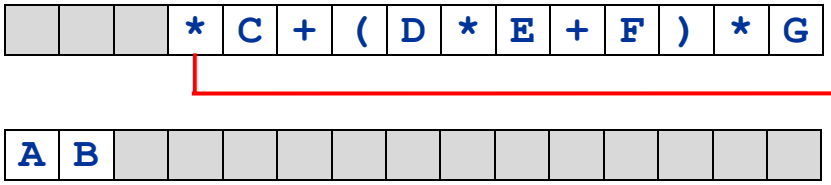
3 шаг



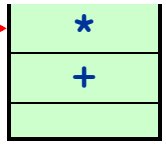
Стек



4 шаг

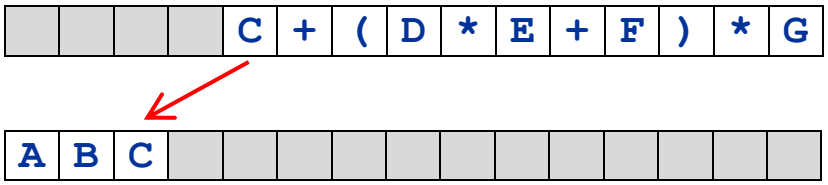


Стек

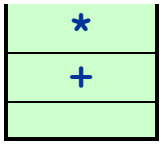


Элемент в вершине стека «+» имеет более низкий приоритет, чем «*», поэтому помещаем в стек.

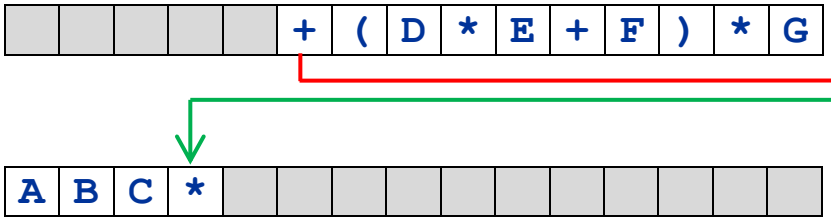
5 шаг



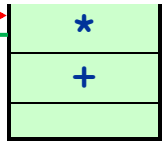
Стек



6-а шаг

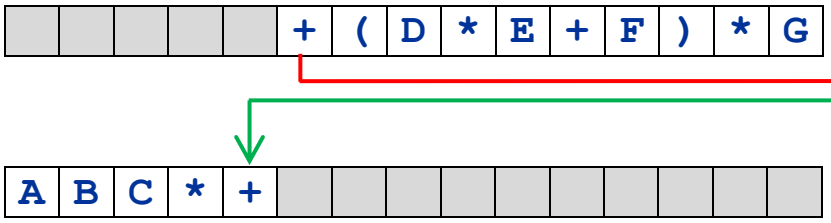


Стек

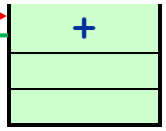


В вершине стека символ «*» имеет более высокий приоритет, чем символ в исходной строке «+». Поэтому извлекаем «*» из стека и помещаем в результирующую строку.

6-б шаг

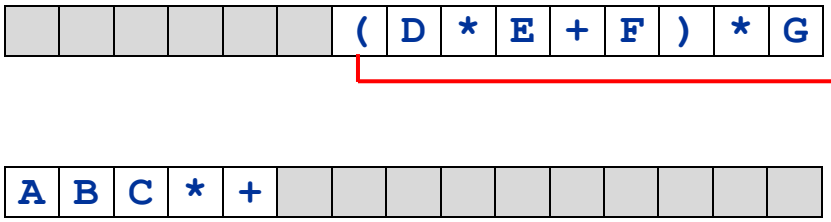


Стек

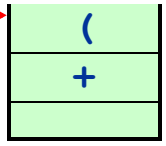


Следующий элемент стека «+» имеет тот же приоритет, что и текущий символ исходной строки. Следовательно, извлекаем из стека «+» и помещаем его в результирующую строку, а прочитанный символ «+» из исходной строки помещаем в стек.

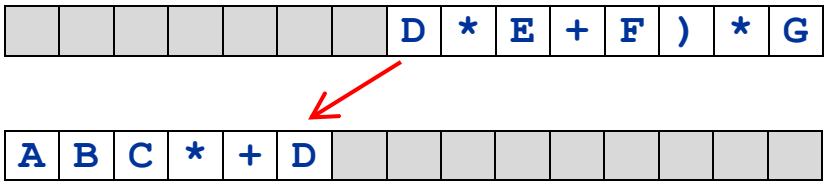
7 шаг



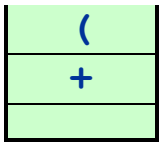
Стек



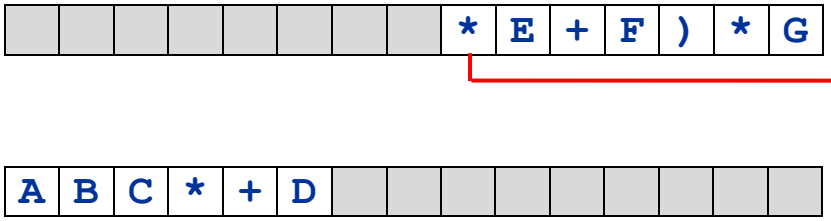
8 шаг



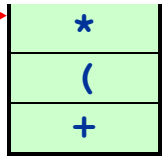
Стек



9 шаг

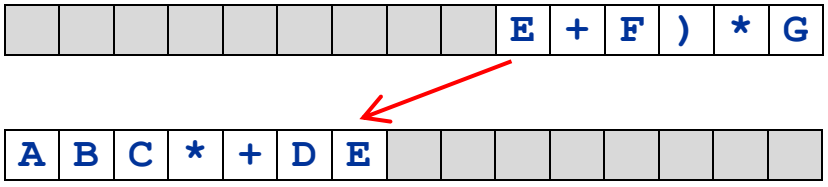


Стек

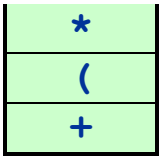


«(» нельзя извлечь из стека до тех пор, пока не встретилась «)». Символ «*» помещаем в стек.

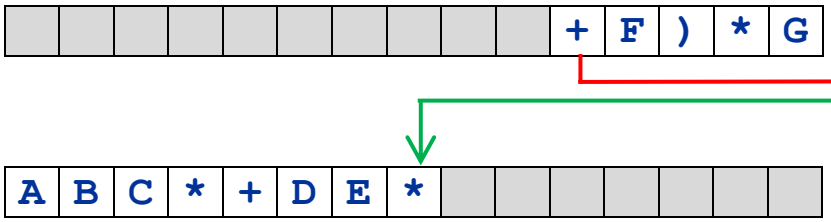
10 шаг



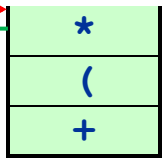
Стек



11-а шаг

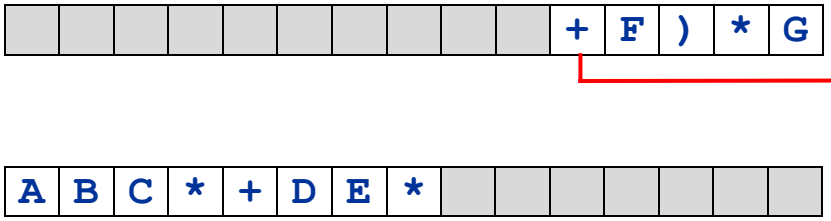


Стек

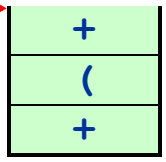


Символ стека «*» имеет более высокий приоритет, чем текущий символ исходной строки «+». Следовательно, извлекаем из стека «*» и помещаем его в результирующую строку.

11-б шаг

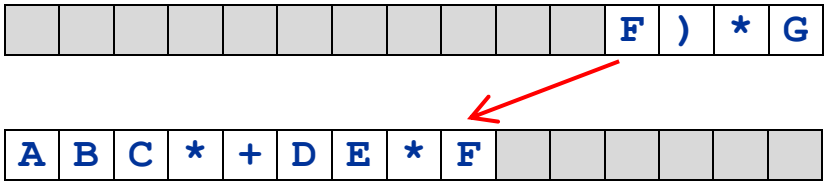


Стек

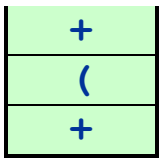


Прочитанный символ «+» из исходной строки помещаем в стек.

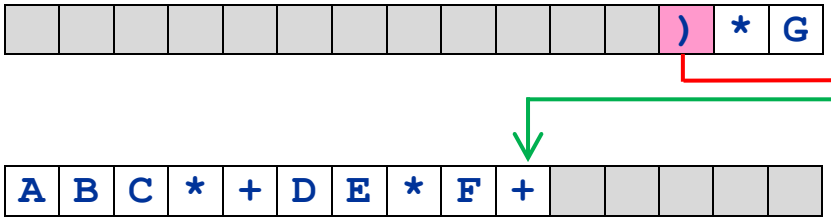
12 шаг



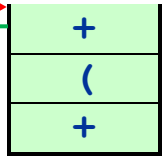
Стек



13 шаг

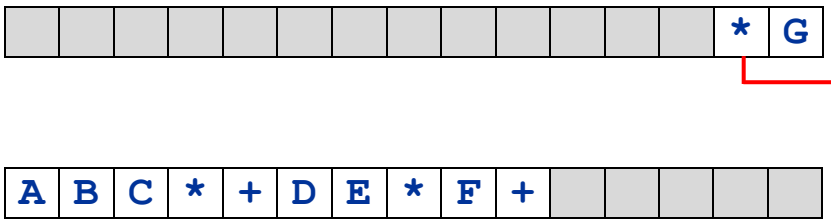


Стек

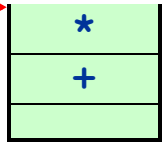


Все элементы стека до символа «(» помещаем в результирующую строку. «(» извлекаем из стека, а «)» в результирующую строку.

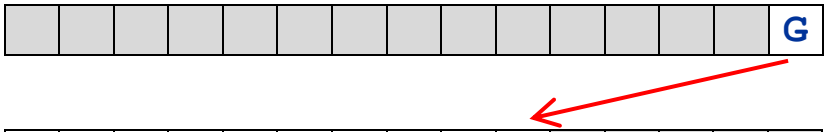
14 шаг



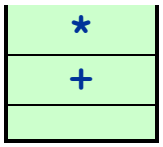
Стек



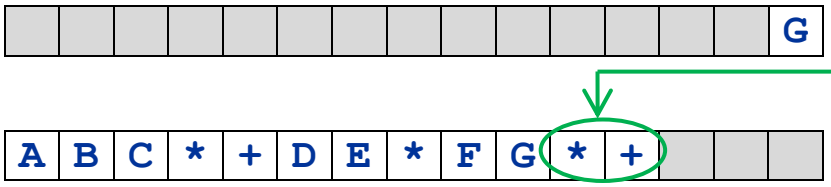
15 шаг



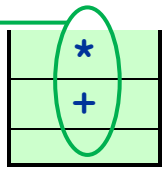
Стек



16 шаг



Стек



При решении задачи будем использовать динамическую память – стек.

Для работы со стеком будем использовать следующие процедуры и функцию.

1. Разместите в блоке реализации после слова **implementation** описание типов и переменных:

```
Type pt_1=^el_1;
     el_1=Record //описание стека для первой части программы
           data:Char;
           next:pt_1;
          End;
     pt_2=^el_2;
     el_2=Record //описание стека для второй части программы
           data:Real;
           next:pt_2;
          End;
Var  s:String;
     f:Boolean;
     rez_1:String;
     rez_2:Real;
```

2. Запись символа в стек

```
Procedure WriteStack_1(Var u:pt_1; dig:Char);
//запись в стек
//u – указатель на начало стека
//dig – значение символа, которое записывается в стек

Var x: pt_1;
 Begin
  New(x);
  x^.data:=dig; x^.next:=u;
  u:=x;
 End;
```

3. Извлечение элемента из стека

В результате выполнения процедуры параметру dig присваивается значение первого элемента стека и изменяется значение указателя на начало списка.

```
Procedure ReadStack_1(Var u:pt_1; Var dig:Char);
//извлечение из стека
//u – указатель на начало стека
//dig – значение символа, которое извлекается из стека

   Var  x: pt_1;
    Begin
     dig:=u^.Data;
     x:=u;
     u:=u^.Next; Dispose(x);
    End;
```

4. Определение приоритета операции

Результатом выполнения функции является значение приоритета операции. Эту функцию будем использовать при записи операции в стек.

```
Function Priority(c: Char): Byte;
Var R:Byte;
Begin
  R := 0;
  Case c of
  '+', '-': R:= 1;
  '*', '/': R:= 2;
    end;
    Priority:=R;
   End;
```

5. Определение есть ли записи в стеке

Результатом выполнения функции является значение True, если стек не пуст и False, если стек пуст. Эту функцию будем использовать перед процедурой ReadStack.

```
Function Free_1(u:pt_1):Boolean;
//u – указатель на начало стека
 Begin
  If u=Nil Then Free_1:=False Else Free_1:= True;
    End;
```

6. Преобразование исходного арифметического выражения в строку, представленную в постфиксной форме

```
Procedure Convert_expression (a:String;Var z:String);
  Var head:pt_1;
      i:Integer;
      w:Char;
  Begin
   head:=Nil;z:='';
   i:=1;
   While i<=Length(a) Do
Begin
Case a[i] of
'0'..'9': z:=z+a[i];
'(': WriteStack_1(head,a[i]);
')': Begin     //Считываем из стека до символа «(»
      ReadStack_1(head,w);
      While w<>'(' Do
      Begin z:=z+w; ReadStack_1(head,w); End;
     End;
'+','-','*','/':
begin
 If not Free_1(head)  Then WriteStack_1(head,a[i])
      else
     begin
       w:=head^.data;
       While Free_1(head)and(Priority(head^.data)>= Priority(a[i]))
do
 Begin  ReadStack_1(head,w);z:=z+w;  End;
        WriteStack_1(head,a[i]) ;
```

```
    end;
end;
     End;
    Inc(i);
   End;
   //Дополняем строку символами операций, запомненных в стеке
  While Free_1(head) Do Begin  ReadStack_1(head,w);  z:=z+w; End;
     End;
```

Для выполнения самого расчета у нас предназначена кнопка **Button4** (**Преобразование**), поэтому оформим процедуру обработки события **OnClick**.

```
procedure TForm1.Button4Click(Sender: TObject);
begin
s:=  Edit1.Text;
If s<>'' Then
     begin Convert_expression(s,rez_1); Edit2.Text:=rez_1 end
      Else Edit2.Text:='Пустая строка';
end;
```

Протестируйте полученную программу (преобразование в постфиксную форму арифметического выражения) на контрольном примере

Исходная строка **5+4*3+(4*2+1)*6**

Результирующая строка **543*+42*1+6*+**

2-ая часть программы (вычисление)

Задано выражение, записанное в постфиксной форме. Вычислить его значение.

Постфиксная форма `6523+8*+3+*`

Исходная строка `6*(5+(2+3)*8+3)=288`

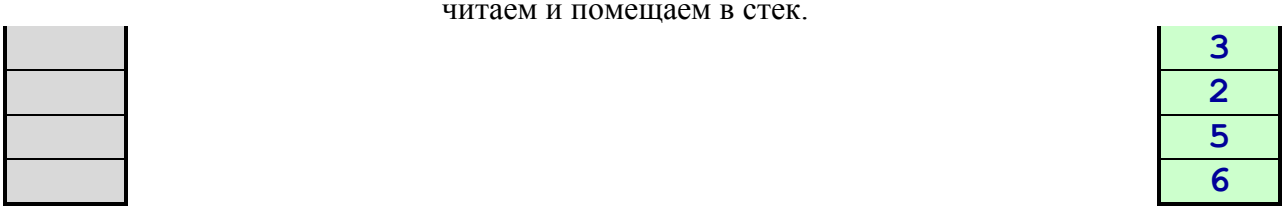
Алгоритм решения задачи с использованием стека

В постфиксной записи каждая операция относится к двум предшествующим операндам. Один из этих операндов может быть результатом операции, выполненной ранее.

Будем записывать в стек каждый встретившийся операнд. Если встречается операция, то ее операндами будут два верхних элемента стека. Извлекаем эти элементы, выполняем над ними текущую операцию, а результат помещаем в стек.

1 шаг

`6523+8*+3+*` Первые четыре символа выражения – операнды, читаем и помещаем в стек.



2 шаг

`+8*+3+*` Символ «+».

Извлекаем из стека соответствующие операнды («3» и «2»), выполняем действие 3 + 2 = 5 и результат помещаем в стек.

3 шаг

`8*+3+*` В стек помещаем «8».

4 шаг

+3+

Символ «*».

Извлекаем из стека соответствующие операнды («8» и «5»), выполняем действие 8 × 5 = 40 и результат помещаем в стек.

8
5
5
6

40
5
6

5 шаг

+3+*

Символ «+».

Извлекаем из стека соответствующие операнды («40» и «5»), выполняем действие 40 + 5 = 45 и результат помещаем в стек.

40
5
6

45
6

6 шаг

3+*

В стек помещаем «3».

45
6

3
45
6

7 шаг

+*

Символ «+».

Извлекаем из стека соответствующие операнды («3» и «45»), выполняем действие 3 + 45 = 48 и результат помещаем в стек.

3
45
6

48
6

8 шаг

*

Символ «*».

Извлекаем из стека соответствующие операнды («48» и «6»), выполняем действие 48 × 6 = 288 и результат помещаем в стек.

48
6

288

Для работы со стеком будем использовать почти такие же процедуры, что и для первой части, но описание стека будет несколько иное:

Описание стека для преобразования	Описание стека для вычисления

`Type pt_1=^el_1;`
`el_1=Record`
`data:Char;`
`next:pt_1;`
`End;`
	`Type pt_2=^el_2;`
	`el_2=Record`
	`data:Real;`
	`next:pt_2;`
	`End;`

1. Запись символа в стек

```
Procedure WriteStack_2(Var u:pt_2; dig:Real);
//запись в стек
//u – указатель на начало стека
//dig – значение символа, которое записывается в стек

Var x: pt_2;
 Begin
  New(x);
  x^.data:=dig; x^.next:=u;
  u:=x;
 End;
```

3. Извлечение элемента из стека

В результате выполнения процедуры параметру dig присваивается значение первого элемента стека и изменяется значение указателя на начало списка.

```
Procedure ReadStack_2(Var u:pt_2; Var dig:Real);
//извлечение из стека
//u – указатель на начало стека
//dig – значение символа, которое извлекается из стека

   Var  x: pt_2;
    Begin
     dig:=u^.Data;
     x:=u;
     u:=u^.Next; Dispose(x);
    End;
```

4. Определение есть ли записи в стеке

Результатом выполнения функции является значение True, если стек не пуст и False, если стек пуст. Эту функцию будем использовать перед процедурой ReadStack.

```
Function Free_2(u:pt_2):Boolean;
//u – указатель на начало стека
 Begin
  If u=Nil Then Free_2:=False Else Free_2:= True;
    End;
```

5. Преобразование исходного арифметического выражения в строку, представленную в постфиксной форме

```
Procedure Convert_calculate(a:String;Var pp:Boolean;Var z:Real);
  Var head: pt_2;
```

```
      i,k: Integer;
      r,w: Real;
  Begin
   head:=Nil; pp:=True;
   i:=1;
   While (i<=Length(a)) And pp Do
   Begin
    If a[i]<>' ' Then Begin{*Пропускаем пробелы.*}
    If Not(a[i] In ['+','-','*','/'])
      Then Begin
       Val(a[i],r,k);
       If k=0 Then WriteStack_2(head,r)
              Else pp:=False;
      End
      Else Begin
        If Free_2(head) Then ReadStack_2(head,r) Else pp:=False;
        If Free_2(head) Then ReadStack_2(head,w) Else pp:=False;
        If pp Then Begin
             Operation(a[i],r,w,r);
             WriteStack_2(head,r);
                   End;
          End;
     End;
    Inc(i);
   End;
   If Free_2(head) Then ReadStack_2(head,z) Else pp:=False;
    End;
```

Для выполнения самого расчета у нас предназначена кнопка **Button1** (**Dsxbckbnm**), поэтому оформим процедуру обработки события **OnClick**.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
s:=  Edit2.Text;
If s<>'' Then Begin
    Convert_calculate(s,f,rez_2);
    If f Then Edit3.Text:=Floattostr(rez_2)
         Else  Edit3.Text:='Выражение не может быть вычислено'
    End
   Else Edit3.Text:='Пустая строка';

end;
```

Протестируйте полученную программу (преобразование в постфиксную форму арифметического выражения) на контрольном примере

Исходная строка	**5+4*3+(4*2+1)*6**
Результирующая строка	**543*+42*1+6*+**
Результат	**71**

Задание для самостоятельного выполнения

1. Вставьте обработку нажатия кнопки **Button3** (**Выход**).

2. Вставьте обработку нажатия кнопки **Button2** (**Очистить**), которая очищает все компоненты **Edit1**-**Edit3**.

3. В программе не предусмотрена обработка ошибок в исходной строке. Устраните этот недостаток.

4. Измените программу так, чтобы она могла обрабатывать в качестве операндов не только цифры, но и числа (целые, вещественные).