Обратная польская запись (ОПЗ)

Сложные вычислительные задачи обычно требуют больших объемов вычислений, поэтому к разработчикам языков программирования предъявлялось требование: максимально приблизить форму записи математических выражений в коде программы к естественному языку математики.

Одну из первых областей системного программирования составили исследования способов трансляции математических выражений.

В результате наибольшее распространение получил метод трансляции с помощью обратной польской записи, которую предложил польский математик Я. Лукашевич.

ОПЗ представляет собой выражение, записанное в постфиксной форме, без скобок, по специальным правилам.

Пусть для операндов ***A*** и ***B*** выполняется операция сложения.

Привычная форма записи ***A+B*** называется ***инфиксной.***

Форма записи, в которой знак операции следует перед операндами ***+AB***, называется ***префиксной***.

Если же операция записывается после операндов ***AB+***, то это ***постфиксная*** форма.

Получение ОПЗ реализуется с использованием структур в виде стека и дерева.

Алгоритм, использующий стек

Получение ОПЗ с использованием стека может осуществляться весьма просто на основе алгоритма, предложенного Дейкстрой, который ввел понятие стекового приоритета операций, например:

Операция	**Приоритет**
(	**1**
+ –	**2**
*** /**	**3**

Суть алгоритма в следующем

Исходное выражение, записанное в виде строки символов ***S***, просматривается слева направо.

Операнды переписываются в выходную строку ***B***, операции обрабатываются с использованием стека, который первоначально пуст, на основе следующих правил.

1. Если в строке ***S*** встретился операнд, то его помещаем в строку ***B***.
2. Если в ***S*** встретилась ***открывающая скобка***, то ее помещаем в стек.

3. Если в ***S*** встретилась ***закрывающая скобка***, то извлекаем из стека и записываем в строку ***B*** все операции до "(", саму "(" скобку также извлекаем из стека; обе скобки игнорируются.

4. Если в ***S*** встретилась операция ***X***, то выталкиваем из стека все операции, приоритет которых не ниже ***X***, после чего саму операцию ***X*** записываем в стек.

5. При достижении конца строки ***S***, анализируем стек и, если он не пуст, извлекаем и переписываем его элементы в выходную строку ***B***.

Пример реализации

Исходное выражение задано в виде строки ***S***

"a + b*c + (d*e + f)*g"

Запишем это выражение в форме ОПЗ.

Ответом будет выражение (без скобок)

abc*+de*f+g*+

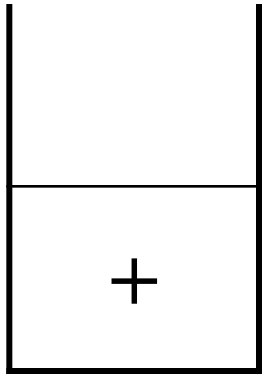
Результат будем получать в строке ***B***.

Начинаем последовательно просматривать символы исходной строки, причем ***B*** – пустая строка и стек пуст.

Всего в строке 15 символов (15 п.п.).

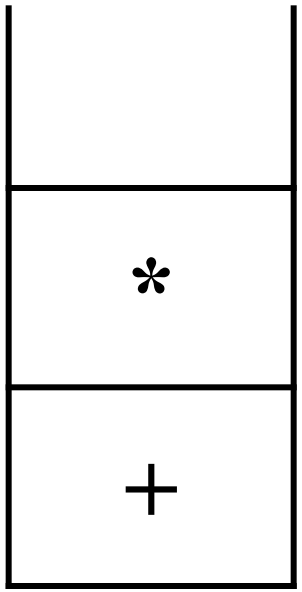
1. Букву «***a***» помещается в строку ***B***
2. Операцию «+» помещаем в стек.
3. Букву «***b***» помещаем в строку ***B***.

На этот момент стек и строка ***B*** выглядят следующим образом:



B = ” ***ab*** ”

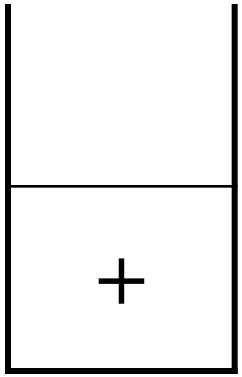
4. Операцию «*» помещаем в стек, т.к. элемент «+» в вершине стека имеет более низкий приоритет.
5. Букву «***c***» помещаем в строку ***B***, после чего имеем



B **= ” *abc* ”**

6. Следующая операция «+»: анализируем стек и видим, что в вершине стека «*» и следующая за ней «+» имеют приоритеты не ниже текущей. Следовательно, обе операции извлекаем из стека и помещаем в строку ***B***, а текущую операцию «+» помещаем в стек.

В итоге имеем

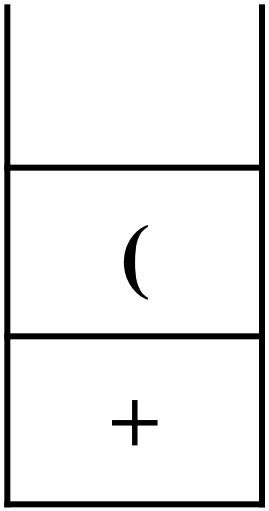


B = ” ***abc****+ ”

7. Далее следует символ «(», его помещаем в стек.

8. Букву «***d***» помещаем в строку ***B***.

В результате получается

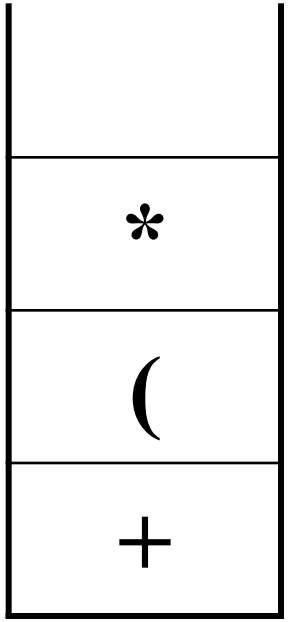


B = ” ***abc*+d*** ”

9. Операцию «*» помещаем в стек, т.к. приоритет у скобки самый низкий.

10. Букву «***e***» помещаем в строку ***B***.

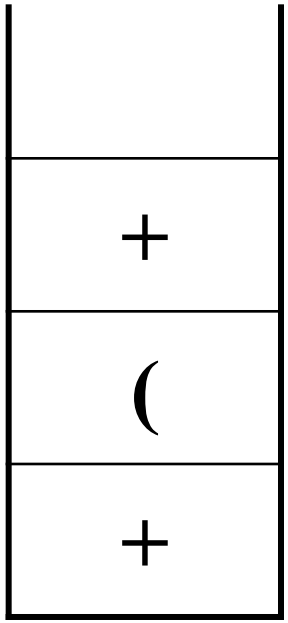
Получили



$$B = \text{”}\ abc*+de\ \text{”}$$

11. Следующая операция «+»: приоритет операции «*» в вершине стека выше, поэтому извлекаем из стека «*» и помещаем в строку ***B***. Текущий символ «+» помещаем в стек.

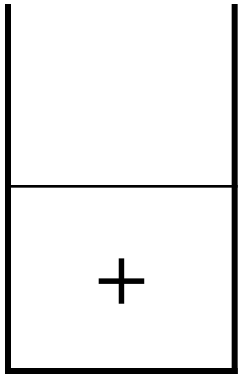
12. Букву «***f***» помещаем в строку ***B***. Получаем



B = ” abc*+de*f ”

13. Далее идет закрывающая скобка, все элементы до символа «(» извлекаем из стека и помещаем в строку ***B*** (это элемент «+»), сам символ «(» тоже извлекаем из стека.

Обе скобки игнорируются:

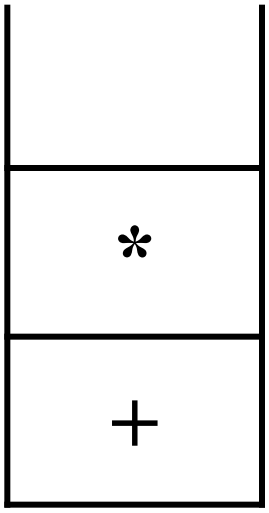


B = ” ***abc*+de*f+*** ”

14. Операцию «*» помещаем в стек, т.к. ее приоритет выше операции «+» в вершине стека.

15. Букву «***g***» записываем в строку ***B***.

Получаем



B = ” ***abc*+*de*f*+*g*** ”

Все символы строки ***S*** просмотрены, следовательно, анализируем состояние стека, если он не пуст, то переписываем все его элементы в строку ***B***, т.е. операции «+» и «*» последовательно извлекаем из стека в строку:

B = ”abc*+de*f+g*+”

Просмотрев исходную информацию только один раз, мы решили поставленную задачу.

Вычисление выражения, записанного в ОПЗ, может проводиться путем однократного просмотра, что является весьма удобным при генерации объектного кода программ.

Рассмотрим простой пример.

Выражение $(A + B) * (C + D) - E$ в виде ОПЗ:

$$AB+CD+*E-$$

Его вычисление проводится следующим образом (R_1 и R_2 – вспомогательные переменные):

Шаг	Анализируемая строка	Действие
1	$A\ B + C\ D + *\ E -$	$R_1 = A + B$
2	$R_1\ C\ D + *\ E -$	$R_2 = C + D$
3	$R_1\ R_2\ *\ E -$	$R_1 = R_1 * R_2$
4	$R_1\ E -$	$R_1 = R_1 - E$
5	R_1	

Текст программы, реализующий рассмотренный алгоритм, может иметь следующий вид:

```
    . . .
struct Stack {
 char c;            // Символ операции
 Stack *next;
} ;
int  Prior (char);
Stack*  InS ( Stack*, char);
Stack*  OutS ( Stack*, char*);
```

```
void main ()
{
   Stack *t,
   *Op = NULL;        // Стек операций Op – пуст
   char a, In [81], Out [81];
// In – входная (S), Out – выходная (B) строки
   int   k = 0, l = 0;
// Текущие индексы для строк
   cout << " Input formula : ";
   cin >> In;
```

```
// Анализируем символы строки In
while ( In[k] != '\0') {
//  1. Если символ – буква, заносим ее в Out
    if ( In[k] >= 'a' && In[k] <= 'z' )
        Out[l++] = In[k];
//  2. Если «(», записываем ее в стек
    if ( In[k] == '(' )
        Op = InS ( Op, In[k] );
```

/* **3**. Если «)», извлекаем из стека в строку ***Out*** все операции до открывающей скобки */

```
 if ( In[k] == ')' ) {
     while ( (Op -> c) != '(' ) {
// Считываем элемент a из стека
         Op = OutS ( Op, &a );                    if ( !Op ) a = '\0';
// и записываем его в строку Out.
         Out[l++] = a;                    }
```

```
// Удаляем из стека открывающую скобку
        t = Op;
        Op = Op -> next;
        delete t;
    }
```

```
/* 4. Если операция, извлекаем из стека в Out опе-
   рации с большим или равным приоритетом */
   if (In[k]=='+' || In[k]=='–' || In[k]=='*' || In[k]=='/') {
    while ( Op != NULL  &&
        Prior (Op -> c) >= Prior (In[k])) {
            Op = OutS (Op, &a);
            Out[l++] = a;
        }
// Текущий символ операции – в стек
    Op = InS (Op, In[k]);
   }
    k++;
}   // Конец цикла анализа входной строки
```

```
/* 5. Если стек не пуст, извлекаем и записываем
   операции в выходную строку  */
   while ( Op != NULL) {
    Op = OutS (Op, &a);
    Out[l++] = a;
   }
   Out[l] = '\0';        //  Окончание строки
   cout << "\n Polish = " << Out << endl;
   getch();              // system("pause");
}
```

```
//------ Реализация приоритета операций ------

int  Prior ( char a )  {
 switch ( a ) {
      case '*': case '/':  return 3;
      case '–': case '+':      return 2;
      case '(':   return 1;
 }
 return 0;
}
```

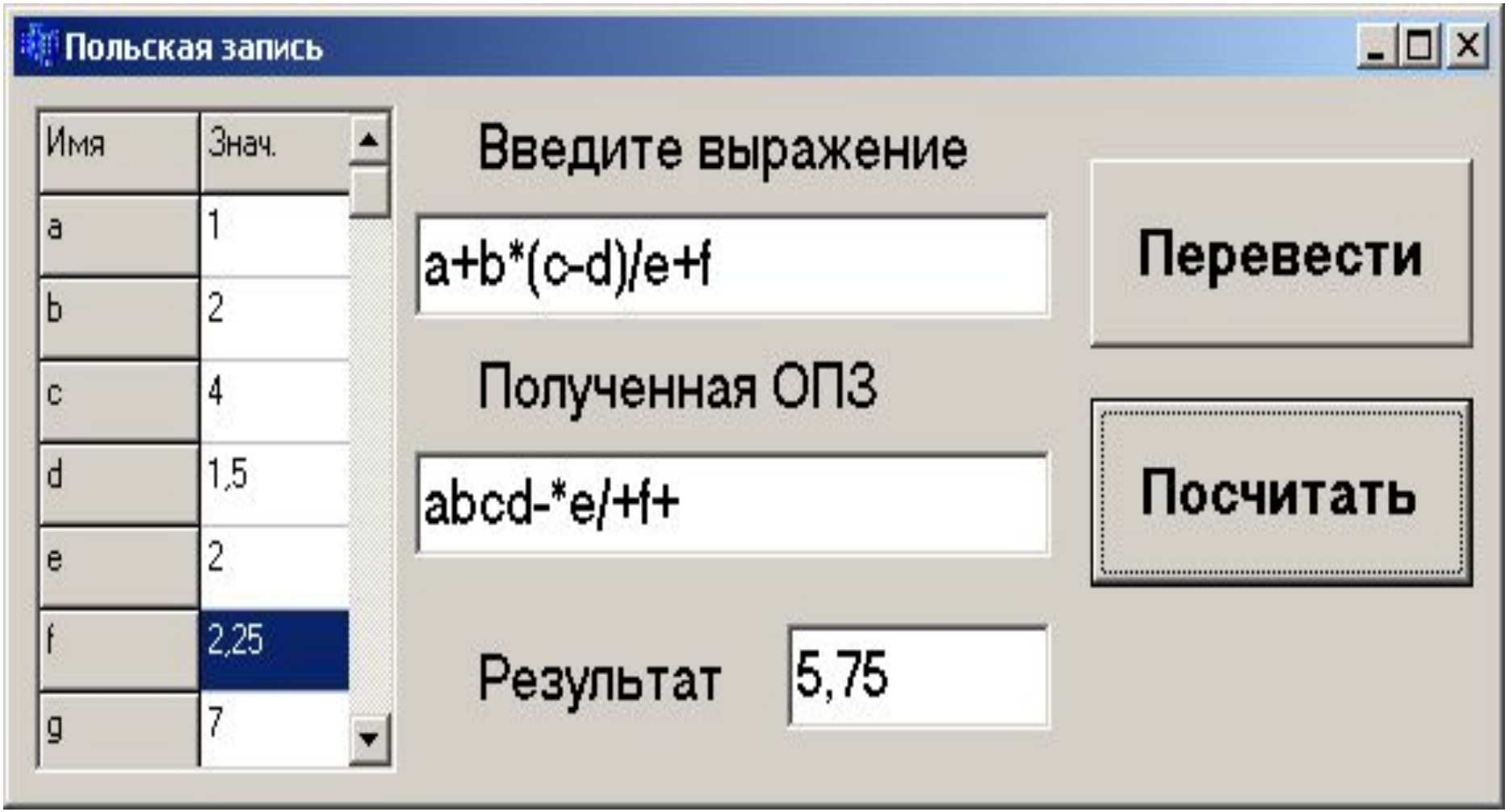
```
// -------- Добавление элемента в стек --------

Stack*  InS ( Stack *p, char s)
{
    Stack *t =  new Stack;
    t -> c  = s;
    t-> next = p;
    return t;
}
```

```
// ------- Извлечение элемента из стека -------


   Stack*  OutS ( Stack *p,  char *s )
   {
        Stack *t = p;
        *s = p -> c;
        p = p -> next;
        delete t;
        return p;
   }
```

Пример, реализованный в методичке для оконного приложения:



```
    struct Stack {
     char info;
     Stack *next;
    } *begin;
int  Prior (char);
Stack*  InStack ( Stack*, char);
Stack*  OutStack ( Stack*, char*);
double  Rezult (String);
double mas[100];    // Массив для вычисления
Set  < char, 0, 255 > znak;
// Множество символов-знаков
    int Kol = 10;
```

//---- Текст функции-обработчика ***FormCreate*** ----

```
Edit1->Text = "a+b*(c-d)/e";
Edit2->Text = "";
char a = 'a';
StringGrid1->Cells[0][0] ="Имя";
StringGrid1->Cells[1][0] ="Знач.";
for (int i = 1; i <= Kol; i++) {
        StringGrid1->Cells[0][i] = a++;
    StringGrid1->Cells[1][i] = i;
}
```

// ---- Текст обработчика кнопки ***Перевести*** ----

```
Stack *t;
begin = NULL;
char ss, a;
String InStr, OutStr;
OutStr = "";
Edit2->Text = "";
InStr = Edit1->Text;
znak << '*' << '/' << '+' << '-' << '^';
int len = InStr.Length(), k;
```

```
    for (k = 1; k <= len; k++) {
     ss = InStr[k];
// ---------- Пункт 1 алгоритма ----------
     if (ss >=  'a' && ss <= 'z' )
          OutStr += ss;
// ---------- Пункт 2 алгоритма ----------
     if ( ss == '(' )
          begin = InStack ( begin, ss);
```

```
// ---------- Пункт 3 алгоритма ----------
    if ( ss == ')' ) {
            while ( (begin -> info) != '(' ) {
                begin = OutStack ( begin, &a );
            OutStr += a;
        }
            begin = OutStack ( begin, &a );
    }
```

```
// ---------- Пункт 4 алгоритма ----------
      if ( znak.Contains ( ss ) ) {
       while ( begin != NULL &&
         Prior ( begin->info ) >= Prior ( ss ) ) {
                    begin = OutStack ( begin, &a );
                    OutStr += a;
       }
          begin = InStack ( begin, ss );
      }     // Конец оператора  if
  }     // Конец оператора  for
```

```
// ---------- Пункт 5 алгоритма ----------
   while ( begin != NULL) {
            begin = OutStack ( begin, &a );
            OutStr += a;
   }
   Edit2 -> Text = OutStr;
// Выводим полученную строку
}
```

```
//---- Текст обработчика кнопки Посчитать ----
   char ch;
   String OutStr = Edit2 -> Text;
   for ( int i = 1; i <= Kol; i++) {
       ch = StringGrid1 -> Cells[0][i][1];
       mas[int(ch)] = StrToFloat(SG1->Cells[1][i]);
   }
// SG это StringGrid
   Edit3->Text=FloatToStr( Rezult ( OutStr ) );
```

//-- Функция реализации приоритета операций --

```
int Prior ( char a ){
 switch ( a ) {
        case '^':          return 4;       // !!!
        case '*':   case '/':  return 3;
        case '-':   case '+':     return 2;
        case '(':          return 1;
 }
 return 0;
}
```

```
//------ Расчет арифметического выражения ------

   double Rezult(String Str)
   {
    char ch, ch1, ch2, chr;
    double op1, op2, rez;
    int len = Str.Length();
    znak << '*' << '/' << '+' << '-' << '^';
    chr = 'z' + 1;
    for (int i=1; i <= len;  i++)  {
            ch = Str[i];
```

```
if (! znak.Contains (ch) )
        begin = InStack ( begin, ch );
    else {
        begin = OutStack ( begin, &ch1 );
        begin = OutStack ( begin, &ch2 );
        op1 = mas[int (ch1) - 97];   // код ‘a’ - 97
        op2 = mas[int (ch2) - 97];
```

```
    switch (ch){
            case '+' : rez = op2 + op1;         break;
            case '-' :  rez = op2 - op1;        break;
            case '*' :  rez = op2 * op1;        break;
            case '/' :   rez = op2 / op1;       break;
            case '^' :  rez = pow(op2,op1);  break;
        }
        mas[int (chr) - 97] = rez;
        begin = InStack ( begin, chr);
        chr++;
    }       // Конец  else
}           // Конец оператора  for
  return rez;
}
```