15. АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Оглавление

§15.1 Преобразование выражения в обратной польской записи с использованием стека

Как правило арифметические выражения удобно преобразовывать в обратную польскую запись (ОПЗ), чтобы избавиться от скобок, содержащихся в выражении. Выражения, преобразованные в ОПЗ, можно вычислять последовательно, слева направо.

Существует два наиболее известных способа преобразования в ОПЗ. Рассмотрим коротко каждый из них:

Нам понадобится стек для переменных типа char, т.к. исходное выражение мы получаем в виде строки.

Рассматриваем поочередно каждый символ:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.

2. Если символ - знак операции (+, -, *, /), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.

Получив один из этих символов, мы должны проверить стек:

а) Если стек все еще 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.

б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную 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 (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения: Дано выражение:

a + (b - c) * d

Рассмотрим поочередно все символы:

Таблица 15.1 – Последовательное преобразование выражений

Символ	**Действие**	**Состояние выходной строки после совершенного действия**	**Состояние стека после совершенного действия**
a	'a' - переменная. Помещаем ее в выходную строку	a	пуст
+	'+' - знак операции. Помещаем его в стек (поскольку стек пуст, приоритеты можно не проверять)	a	+
(	'(' - открывающая скобка. Помещаем в стек.	a	+ (
b	'b' - переменная. Помещаем ее в выходную строку	a b	+ (

-	'-' - знак операции, который имеет приоритет 2. Проверяем стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в стек.	a b	+ (-
c	'c' - переменная. Помещаем ее в выходную строку	a b c	+ (-
)	')' - закрывающая скобка. Извлекаем из стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки.	a b c -	+
*	'*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '*'. Следовательно мы должны просто поместить текущий символ '*' в стек.	a b c -	+ *
d	'd' - переменная. Помещаем ее в выходную строку	a b c - d	+ *

Теперь вся входная строка разобрана, но в стеке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку. Поскольку стек - это структура, организованная по принципу LIFO, сначала извлекается символ '*', затем символ '+'.

Итак, мы получили конечный результат:

a b c - d * +

§15.2 Алгоритм вычисления выражения, записанного в обратной польской записи

Для реализации этого алгоритма используется стек для чисел (или для переменных, если они встречаются в исходном выражении). Алгоритм очень прост. В качестве входной строки мы теперь рассматриваем выражение, записанное в ОПЗ:

1. Если очередной символ входной строки - число, то кладем его в стек.

2. Если очередной символ - знак операции, то извлекаем из стека два верхних числа, используем их в качестве операндов для этой операции, затем кладем результат обратно в стек.

Когда вся входная строка будет разобрана в стеке должно остаться одно число, которое и будет результатом данного выражения.

Рассмотрим этот алгоритм на примере выражения:

7 5 2 - 4 * +

Рассмотрим поочередно все символы:

Таблица 15.2 – Последовательное вычисление выражения в ОПЗ

Символ	**Действие**	**Состояние стека после совершенного действия**
7	'7' - число. Помещаем его в стек.	7
5	'5' - число. Помещаем его в стек.	7 5
2	'2' - число. Помещаем его в стек.	7 5 2
-	'-' - знак операции. Извлекаем из стека 2 верхних числа (5 и 2) и совершаем операцию 5 - 2 = 3, результат которой помещаем в стек	7 3
4	'4' - число. Помещаем его в стек.	7 3 4
*	'*' - знак операции. Извлекаем из стека 2 верхних числа (3 и 4) и совершаем операцию 3 * 4 = 12, результат которой помещаем в стек	7 12
+	'+' - знак операции. Извлекаем из стека 2 верхних числа (7 и 12) и совершаем операцию 7 + 12 = 19, результат которой помещаем в стек	19

Теперь строка разобрана и в стеке находится одно число 19, которое является результатом исходного выражения.

§15.3 Программная реализация алгоритма вычисления выражения, записанного в обратной польской записи

Для реализации алгоритма преобразования и вычисления выражения в обратной польской записи требуется определить класс «Оператор»:

Листинг 15.1

```
1  public class Operator
2  {
3  public int priority;
4  public string symbol;
5  public Operator(string symbol, int priority)
6  {
7  this.priority = priority;
8  this.symbol = symbol;
9  }
10 }
```

Свойство `priority` будет отвечать за хранение приоритета операции, свойство `symbol` – знака операции.

Контейнерный класс для хранения операторов, используемых в арифметических выражениях:

Листинг 15.2 – Контейнерный класс для хранения операторов

```
1  public class ContainerOperator
2  {
3  public static List<Operator> operators = new List<Operator>();
4  public static void Add_Operators(Operator op)
5  {
6  operators.Add(op);
7  }
8  public static Operator Find_Operator(string s)
9  {
10 foreach (var item in operators)
11 {
12 if(item.symbol == s)
13 {
14 return item;
15 }
16 }
```

```
17  return null;
18  }
19  }
```

Программная реализация алгоритма преобразования выражения в обратной польской записи с использованием стека представлена в листинге 15.3. Здесь `operators_stack` – это стек, хранящий объекты класса `Operator`, а `input_string` – выходная строка (переменная строкового типа).

Листинг 15.3

```
1   foreach (char c in textBox1.Text)
2   {
3   if (Char.IsDigit(c))
4   {
5   this.input_string += c;
6   continue;
7   }
8   if (c == '+' || c == '-' || c == '*' || c == '/')
9   {
10  if (this.operators_stack.Count == 0)
11  {
12  this.operators_stack.Push(Convert.ToString(c));
13  continue;
14  }
15  else if (this.operators_stack.Count != 0)
16  {
17  if (ContainerOperator.Find_Operator
    (this.operators_stack.Peek()).priority <
    ContainerOperator.Find_Operator
    (Convert.ToString(c)).priority)
18  {
19  this.operators_stack.Push(Convert.ToString(c));
20  continue;
21  }
22  }
23  if (this.operators_stack.Count != 0)
24  {
25  if (ContainerOperator.Find_Operator
    (this.operators_stack.Peek()).priority <
    ContainerOperator.Find_Operator
    (Convert.ToString(c)).priority)
26  {
27  continue;
28  }
```

```
29  try
30  {
31  while (ContainerOperator.Find_Operator
    (this.operators_stack.Peek()).priority >=
    ContainerOperator.Find_Operator
    (Convert.ToString(c)).priority)
32  {
33  this.input_string += this.operators_stack.Pop();
34  }
35  }
36  catch
37  {
38  }
39  if (this.operators_stack.Count == 0)
40  {
41  this.operators_stack.Push(Convert.ToString(c));
42  continue;
43  }
44  else if (this.operators_stack.Count != 0)
45  {
46  if (ContainerOperator.Find_Operator
    (this.operators_stack.Peek()).priority <
    ContainerOperator.Find_Operator
    (Convert.ToString(c)).priority)
47  {
48  this.operators_stack.Push(Convert.ToString(c));
49  continue;
50  }
51  }
52  }
53  }
54  if (c == '(')
55  {
56  this.operators_stack.Push(Convert.ToString(c));
57  continue;
58  }
59  if (c == ')')
60  {
61  while (this.operators_stack.Peek() != "(")
62  {
63  this.input_string += this.operators_stack.Pop();
64  }
65  this.operators_stack.Pop();
66  continue;
```

```
67  }
68  }
69  while (this.operators_stack.Count != 0)
70  {
71  input_string += this.operators_stack.Pop();
72  }
```

Программная реализация алгоритма вычисления выражения, записанного в обратной польской записи:

Листинг 15.4

```
1   foreach (char c in this.input_string)
2   {
3   if (Char.IsDigit(c))
4   {
5   this.operators_stack.Push(Convert.ToString(c));
6   continue;
7   }
8   else if (!Char.IsDigit(c))
9   {
10  if (c == '+')
11  {
12  temp = (Convert.ToString(Convert.ToInt32
    (Convert.ToString(this.operators_stack.Pop())) +
    Convert.ToInt32(Convert.ToString
    (this.operators_stack.Pop()))));
13  }
14  else if (c == '-')
15  {
16  this.ro = Convert.ToInt32(Convert.ToString
    (this.operators_stack.Pop()));
17  this.lo = Convert.ToInt32(Convert.ToString
    (this.operators_stack.Pop()));
18  temp = (Convert.ToString(this.lo - this.ro));
19  }
20  else if (c == '*')
21  {
22  temp = (Convert.ToString(Convert.ToInt32
    (Convert.ToString(this.operators_stack.Pop())) *
    Convert.ToInt32(Convert.ToString
    (this.operators_stack.Pop()))));
23  }
24  else if (c == '/')
25  {
26  this.ro = Convert.ToInt32(Convert.ToString
```

```
        (this.operators_stack.Pop()));
27      this.lo = Convert.ToInt32(Convert.ToString
        (this.operators_stack.Pop()));
28      temp = (Convert.ToString(this.lo / this.ro));
29      }
30      this.operators_stack.Push(temp);
31      }
32      }
33      label_Result.Text                                    =
        Convert.ToString(this.operators_stack.Pop());
```

Так как операция вычитания и деления не обладают свойством коммутативности, при их обработке используется две целочисленные переменные – `lo` и `ro`. Эти переменные представляют собой левый и правый операнд данных арифметических операций.