0.1 Рекурсивный разбор выражений

В этом разделе мы научимся считать значение выражения, заданного строкой. Например, мы хотим научиться считать "2+2*2".

0.1.1 Форма Бэкуса-Наура

Форма Бэкуса-Наура (БНФ) — набор определений, разъясняющих структуру рассматриваемого текста (например, программ на языке C++). Правила БНФ похожи на присваивания, так как состоят из трех частей: **определяемой лексемы** (той лексемы, смысл которой мы хотим разъяснить), **знака "::="**, который читается как "есть по определению" и, собственно, **определения**. При записи формы Бэкуса-Наура лексемы принято заключать в треугольные скобки, чтобы не путать их с символами. Например,

<цифра> ::= 0|1|2|...|9

Эта строчка читается так: "цифра есть по определению символ '0', либо ("либо" обозначается вертикальной палочкой) символ '1', либо символ '2', либо ..., либо символ '9'". Теперь, когда мы знаем, что такое <цифра>, мы можем определить понятие числа.

<число> ::= <цифра>|<цифра><число>

Как вы видите, форма Бэкуса-Наура допускает "рекурсивные определения". В БНФ можно записывать и более сложные правила. Таким образом можно записать синтаксис любого языка программирования! Например, if в C++ записывается так:

<if> ::= if (<выражение>) <операторы> | if (<выражение>) <операторы> else <операторы>

Еще один пример — определение правильной скобочной последовательности. Как мы помним, правильная скобочная последовательность (ПСП) определяется так:

- Пустая строка (ε) является ПСП
- Две записанные подряд ПСП образуют ПСП
- ПСП, заключенная в скобки, тоже является ПСП

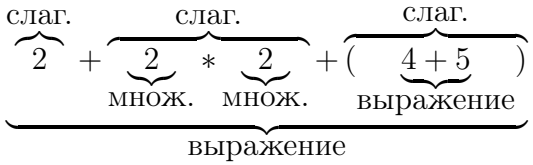
То же самое записывается в одну строчку:
<ПСП> ::= ε | <ПСП><ПСП> | (<ПСП>)

0.1.2 Разбор выражения

Перед тем, как говорить о разборе арифметического выражения, надо дать определение арифметического выражения. Сделаем это в уже известной нам форме Бэкуса-Наура.

1. <**выражение**> **::=** <**слагаемое**> | <**слагаемое**> + <**выражение**> | <**слагаемое**> − <**выражение**>
2. <**слагаемое**> **::=** <**множитель**> | <**множитель**> ∗ <**слагаемое**> | <**множитель**> / <**слагаемое**>
3. <**множитель**> **::=** <**число**> | (<**выражение**>)

Это означает следующее: выражение состоит из слагаемых, слагаемые — из множителей, а множители — это либо числа, либо выражение в скобках.



Мы будем разбирать выражение слева направо, поэтому заведем глобальную переменную curPos — номер текущего (**еще не рассмотренного** символа). Итак, мы хотим написать 4 функции: getExpression, getSum, getFactor, getNumber. Все эти функции разбирают глобальную переменную expr, начиная с позиции curPos. Сначала напишем заголовки функции, так как функции могут циклически вызывать друг друга.

```
1  double getExpression();
2  double getSum();
3  double getFactor();
4  double getNumber();
```

Теперь начнем писать разбор по БНФ: посмотрим на первую формулу: <**выражение**> **::=** <**слагаемое**> | <**слагаемое**> + <**выражение**> | <**слагаемое**> − <**выражение**>
Или, что тоже самое, <**выражение**> **::=** <**слагаемое**> ± <**слагаемое**> ± **...** ± <**слагаемое**>

```
1  double getExpression() {
2      double res = getSum();
3      while (expr[curPos] == '+' || expr[curPos] == '-')
4          if (expr[curPos] == '+') {
5              curPos++;
6              res += getSum();
7          } else {
8              curPos++;
9              res -= getSum();
10     }
11     return res;
12 }
```

Вторая строчка кода получает первое слагаемое. Затем в цикле while находятся оставшиеся слагаемые и добавляются к результату с соответствующим знаком. Обратите внимание на 5-ю и 8-ю строчки. Они говорят, что символ '+' или '-' соответственно мы уже обработали, и что разбор надо начинать со следующего символа, так как '+' и '-' не входят в слагаемые.

Функция getSum() описывается похоже.

```
1  double getSum() {
2      double res = getFactor();
3      while (expr[curPos] == '*' || expr[curPos] == '/')
4          if (expr[curPos] == '*') {
5              curPos++;
6              res *= getFactor();
7          } else {
8              curPos++;
9              res /= getFactor();
10     }
11     return res;
12 }
```

При необходимости можно убедиться в отсутствии деления на ноль. Для этого надо добавить проверку на то, что значение, возвращенное функцией getFactor() в 9-й строчке, ненулевое.

Теперь научимся разбирать множители.

```
1  double getFactor() {
2      if (expr[curPos] == '(') {
3          curPos++;
4          double res = getExpression();
5          curPos++;
6          return res;
7      } else
8          return getNumber();
9  }
```

Во второй строчке мы узнаем, в каком из случаев мы находимся: либо (<выражение>), тогда текущий обрабатываемый символ — '(', либо <число>. 3-я строчка пропускает '(', а 5-й соответствующую ей ')'.

Осталось лишь научиться считывать число.

```
1  double getNumber() {
2      double res = 0;
3      while (curPos < strLenght   &&
               expr[curPos] >= '0'  &&
               expr[curPos] <= '9') {
4          res = res * 10 + (expr[curPos] - '0');
5          curPos++;
6      }
7      return res;
8  }
```

Здесь strLength — длина строки curPos. В функции main() разбор будем запускать так:

```
...
1  curPos = 0; //начнем обрабатывать сначала
2  double result = getExpression(); //разберем
3  cout << result;
...
```

Заметьте, что такой разбор верно учтет приоритеты операций. Это обеспечивается тем, что выражение разбивается сначала по операциям '+' и '−' (1 уровень), а только потом по '*' и '/' (2 уровень).

Мы написали функции разбора по готовой форме Бэкуса-Наура. По БНФ можно писать и более сложные разборы, в том числе и разбор числа в общем виде (со знаком и без знака, с и без дробной части, с и без экспоненциальной части...), или, например, разбор логического выражения. Только при их разборе надо быть аккуратным, так как в языке C++ действуют законы сокращенной логики. Если мы разбираем логическое ИЛИ, а значение текущего значения выражения result уже истинно, то строчка result = result || getAND() не запустит функцию getAND(). Это произойдет, из-за того, что компилятор посчитает этот вызов лишним (не только для компилятора очивидно, что выражение истинно). Из-за этого curPos сместится на неверную позицию.

В целом, задачи на разбор выражений решаются в два этапа. Сначала надо посидеть немножко с бумажкой — написать БНФ, а затем взять и написать программу, сверяясь с БНФ.

0.1.3 Проверка арифметического выражения на корректность

Иногда пользователь может ошибаться и вводить некорректные выражения. Сразу предположим, что в корректном арифметическом выражении нет пробелов. Ошибки могут быть следующими:

- Была открыта скобка, но соответствующий закрывающий скобки нет.
  - Проверяется просто: когда мы пропускаем ожидаемую закрывающую скобку, надо проверить, что пропускаемый символ действительно ')' и, в противном случае, выдать сообщение об ошибке.
- В разборе множетеля мы встретили не число и не скобку.
  - Добавить проверку на корректность рассматриваемого символа в функции getFactor: если не цифра и не скобка, то ошибка.
- Было произведено деление на ноль.
  - При разборе множителей перед применением деления сначала проверить, что значение знаменателя ненулевое. В противном случае, выдать сообщение об ошибке.

- Остальные ошибки приводят к тому, что разбор выражения останавливается не в конце разбираемой строки.
  - После завершения разбора проверить, что текущий рассматриваемый символ идет за последним значащим символом (curPos == strLength). В противном случае, выдать сообщение об ошибке.