

ОПНЧИК

Фоменко М. Ю.

Информатика

зачем мы это вообще делаем

В лабе 5 у некоторых студентов появился вопрос: "а мы это все зачем вообще делаем? Есть же готовый, он работает и тд".

"бесконечных" списков теперь не бывает

зачем нам этот "польский калькулятор«

В этой лабе мы рассмотрим алгоритм обработки

КАК СЧИТАЕТ КОМП

$2+2*2$ не подходит для вычислений

Иф – элс лапша ммм

Будем записывать выражение другим образом

Обратная Польская Нотация

- Обратная польская запись, постфиксная нотация — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.

$3 + 4 \rightarrow 3\ 4\ +$

$(3 + 4) * 5 \rightarrow 3\ 4\ +\ 5\ *$

Инфиксная запись

Инфиксная запись (инфиксная нотация) — форма записи математических и логических формул, в которой операторы (сложение, вычитание, умножение и деление) записываются между operandами (числами, переменными, функциями и другими математическими объектами).

$$2+2*2$$

Префиксная запись

Префиксная запись (польская нотация) — форма записи логических, арифметических и алгебраических выражений, в которой оператор располагается слева от operandов. В отличие от инфиксной нотации, где операторы находятся между operandами.

Было: $2+2*2$

Стало: $+ 2 * 2 2$

Постфиксная запись

Постфиксная (обратная польская) запись — форма записи математических и логических выражений, в которой знак операции записывается не между operandами, а после operandов. Также называется обратной бесскобочной записью, постфиксной нотацией.

Было: $2+2*2$

Стало: $2\ 2\ *\ 2\ +$

А мы что используем?

- Префикс: + 2 * 2 2
- Постфикс: 2 2 * 2 +

На первый взгляд может показаться, что разница тут не сильно почувствуется, просто зеркально? НЕТ.

Считаем префиксную

- Выражение $+ 2 * 2 2$

Читаем $+$. Это команда. Но где данные? Их нет. Они будут "где-то там" в будущем.

Нам нужно запомнить этот плюс (положить в стек операторов или уйти в рекурсию) и ждать.

Читаем 2. Это первое число.

Читаем $*$. Это команда. Но где данные? Их нет. Они будут "где-то там" в будущем.

Читаем 2. Это первое число.

Читаем 2. Это второе число.

А теперь надо "вспомнить", что там было умножение.

А потом еще и для плюса...

Считаем префиксную

Чтобы посчитать Префиксную запись простым алгоритмом, нам нужно читать строку справа налево (с конца).

А чтобы читать с конца, нам нужно сначала загрузить всю строку в память.

Считаем постфиксную

- 2 2 * 2 +

Читаем 2. Это данные? Да. В стек.

Читаем 2. Это данные? Да. В стек.

Читаем *. Это команда? Да. Данные для неё УЖЕ лежат в стеке (мы их только что прочли).

Посчитали, результат (4) кладем обратно в строку (получится строка 4 2 +)

Читаем 4 в стек.

Читаем 2 в стек.

Читаем +. Считаем результат, кладем в строку.

Это конец, в этой же строке будет ответ.

СПИСОК В СТЕК

В прошлой лабе вы, мученики, писали свой `List<T>`. Зачем?
Чтобы понять, что под капотом.

В данной работе нам понадобится новая структура данных -
стек. Стек работает по принципу LIFO.

Принцип LIFO (`Last In, First Out`).

Основные методы

- `Push` (положить В КОНЕЦ),
- `Pop` (взять С КОНЦА и удалить),
- `Peek` (подсмотреть КОНЕЦ).

Генерики (<T>)

- Почему Stack<T>, а не StackInt?

Потому что в одной части лабы нам нужен Stack<double> (для чисел), а в другой — Stack<char> (для плюсиков и минусиков).

Вычислитель записи ОПН

Идем по строке слева направо. Для вычисления нам нужен 1 стек для чисел.

- Видим число -> Push в стек.
- Видим знак -> Делаем Pop два раза (достаем два последних числа), считаем, результат -> Push обратно.
- В конце в стеке останется одно число. Это ответ.

вычислитель записи ОПН

Решаем:

3 4 + 2 *

и

2 3 4 * +

Это $2+3^4$

Алгоритм сортировочной станции

Алгоритм предложен Эдсгером Дейкстрой в 1961 году.

Как нам перевести строку $3 + 4 * 2$.

Входные данные

input: Стока (или массив строк), содержащая инфиксное выражение. Токены уже разделены.

Пример: `["3", "+", "4", "*", "2"]`

output: Список (или строка), куда мы записываем результат. В начале пуст.

stack: Стек символов (или строк) для хранения операторов. В начале пуст.

Алгоритм сортировочной станции

Таблица приоритетов (Веса операторов)

Оператор	Приоритет	Описание
(0	Нижайший (барьер)
)	-	Не имеет приоритета, это команда на исполнение
+,-	1	Низкий
*, /	2	Средний

Алгоритм сортировочной станции

Мы читаем input слева направо, токен за токеном.

Ситуация 1: Токен — Число

Действие: Сразу добавляем в output.

Стек: Не трогаем.

Логика: Операнды в ОПН сохраняют свой относительный порядок. Если в исходнике 3 шла перед 4, то и в ОПН 3 будет перед 4.

Алгоритм сортировочной станции

Ситуация 2: Токен — Открывающая скобка (

Действие: stack.Push('(')

Логика: Это "начало изоляции". Всё, что произойдет дальше, не должно взаимодействовать с тем, что было до скобки, пока мы её не закроем.

Алгоритм сортировочной станции

Ситуация 3: Токен — Закрывающая скобка)

Действие:

Запускаем цикл: `while (stack.Peek() != '(')`

Внутри цикла: `output.Add(stack.Pop())`. (Перекладываем операторы из стека в выходную строку).

ВАЖНО: Если стек опустел, а (так и не нашли — Ошибка:
Пропущенная скобка.

После цикла: `stack.Pop()`. Мы удаляем открывающую скобку из стека, но НЕ пишем её в `output`. Скобки в ОПН не нужны, они своё отработали.

Алгоритм сортировочной станции

Ситуация 4: Токен — Оператор (op1)

Действие:

Смотрим на верхушку стека (op2 = stack.Peek()).

- Проверяем условие цикла WHILE:
- Стек не пуст.
- На вершине стека НЕ открывающая скобка (.
- Приоритет(op2) >= Приоритет(op1).

Внутри цикла: output.Add(stack.Pop()). Выкидываем "сильного" оператора из стека в результат.

Повторяем проверку (п. 2) для нового оператора на вершине.

После цикла: stack.Push(op1). Текущий оператор занял своё место.

Алгоритм сортировочной станции

Ситуация 5: Входная строка закончилась

Действие: while (stack.Count > 0) ->
output.Add(stack.Pop()).

Логика: Всё, что залежалось в стеке, вываливаем в
конец строки.

ВАЖНО: Если в этот момент встретили (— Ошибка:
Непарная скобка.

Алгоритм сортировочной станции

Решаем

$3+4*2$ (ожидаем $3\ 4\ 2\ *\ +$)

И

$3*4+2$ (ожидаем $3\ 4\ *\ 2\ +$)

Спасибо за внимание