

Лабораторная работа №2

Цель

Целью данной лабораторной работы является применение на практике алгоритма рекурсивного спуска на основе библиотеки обработки входной строки.

Общие сведения

Инструменты

Для выполнения данной лабораторной работы Вам потребуется любой компилятор языка C, способный генерировать 32 разрядные программы для той ОС, с которой Вы работаете. Также Вам потребуется программа, для сборки проектов, такая, как программа make.

Основная Ваша работа будет заключаться в формировании программы перевода выражения в постфиксную запись, которая была разобрана на лекции. Для первичной обработки строк предлагается использовать модуль match.

Процедура рекурсивного спуска

```
prim()
{
    int s;
    if (match("(")) {
        if (expr()==0) return 0;
        if (!match(")")) { cerror(E_RIGHT_BRACKET); return 0;
    }

    else if (a=numb(&s)) {
        printf("%d ", s);
    } else { cerror(E_EXPRESS_SINTAX); return 0; }
    return 1;
}

mult()
{
    if (prim()==0) return 0;
    while (1) {
        if (match("*")) {
            if (prim()==0) return 0;
            printf("* ");
        } else if (match("/")) {
            if (prim()==0) return 0;
            printf("/ ");
        } else break;
    }
    return 1;
}

add()
{
    if (mult()==0) return 0;
    while (1) {
        if (match("+")) {
            if (mult()==0) return 0;
            printf("+ ");
        }
    }
}
```

```

        } else if (match("-")) {
            if (mult()==0) return 0;
            printf("- ");
        } else break;
    }
    return 1;
}

```

Модуль match

Данный модуль предназначен для первичной обработки строк и позволяет выделять из строки лексемы. Здесь описаны интерфейсные функции модуля match.

void match_init(char* line);

Процедура инициализации. Разбор строки всегда начинается с вызова этой функции. В качестве параметра передается входная строка.

void match_done(void);

Процедура деинициализации. После разбора строки необходимо вызвать данную функцию.

int match(char *lit);

Поиск указанной подстроки в начале входной строки. Если есть, то найденная подстрока удаляется и функция возвращает 1.

int amatch(char *lit, int len);

Аналогично предыдущему, но дополнительно указывается длина подстроки. Это используется для того, чтобы различать выражения типа 'for' и 'formula', где первое, это ключевое слова, а второе – идентификатор.

int symname(char *sname);

Ищет идентификатор (метку) во входной строке. Метка начинается с подчеркивания или буквы, далее следуют буквы, подчеркивание или цифры.

int number(int val[]);

Ищет целое знаковое число во входной строке

void skipblanks(void);

Пропускает последовательность пробельных символов (пробел, табуляция) во входной строке.

skipchars(void);

Пропускает лексемы и следующие пустые символы. Используется для обработки ошибок.

Задание

1. Реализовать программу рекурсивного спуска выражения и перевода в постфиксный вид.
2. Реализовать главную программу, считывающую строки из входного потока, и выдающие в конце сообщения "ОК", в случае успешного разбора или "FAIL", в случае не успешного разбора.
3. Добавить в процедуру разбора задание по номеру варианта.

4. (дополнительное). Изучить разработку make файлов и разработать make файл для сборки проекта из двух модулей.
5. (дополнительное). Функция разбора вещественного числа в инженерной форме.

Список вариантов для гр. 58х-1

1. Поддержка бинарной операции \wedge – возведение в степень.
2. Поддержка унарной операции $++$ – постинкремент.
3. Поддержку функций \sin , \cos , tg , asin , acos , atan . В формате \sin ($\langle \text{выражение} \rangle$)
4. Поддержку произвольного набора функций в формате: $\langle \text{имя} \rangle (\langle \text{выражение} \rangle)$. После разбора необходимо выдать список функций, используемых в выражении.
5. Поддержка переменных в виде $\langle \text{имя} \rangle$. После разбора необходимо выдать список переменных, используемых в выражении.
6. Поддержка унарных операций $+$ и $-$.
7. Поддержка ввода знаковых целых чисел языка Си. Учесть возможность указания числа в 16-ричной, 8-ричной и десятичной системах счисления.

Список вариантов для гр. 58х-2

1. Комплексные числа

Обеспечить поддержку вещественных и комплексных чисел. Числа вводить в нижеследующем формате

```
<вещ. число> := <число>
<вещ. число> := <число> . <число>
<комплексн. число> := i
<комплексн. число> := i + <вещ. число>
<комплексн. число> := <вещ. число> i
<комплексн. число> := <вещ. число> i + <вещ. число>
```

На выходе комплексные числа представлять в следующем виде:

```
complex (<комплексн. часть>, <реальн. часть>)
```

где **комплексн. часть** и **реальн. часть** - вещественные числа. Вещественные числа выводить без изменений.

Например

5.6i+4.9 на выходе `complex(5.6,4.9)`

0.6i на выходе `complex(0.6,0)`

i+7 на выходе `complex(1,7)`

2. Матрицы

Обеспечить поддержку матриц, состоящих из целых чисел. Матрицы вводить в нижеследующем формате:

```
<матрица> := [<строки>]
<строки> := <строка> | <строка> ; <строки>
<строка> := <число> | <число> , <строка>
```

Например,

Матрица $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ будет записана в виде `[1,2,3;4,5,6]`

вектор-столбец (матрица) $\begin{pmatrix} 1 \\ 4 \\ 5 \end{pmatrix}$ будет записана в виде `[1;4;5]`

запись вида `[1,2,3;5,6]` будет считаться ошибочной.

на выходе матрицы представлять в виде:

matrix (<строка>, <столбцов>, <значения>)

где **строка** и **столбцов** это целые числа - количество строк и столбцов соответственно, а значения, это перечисленные через запятую слева направо и сверху вниз, значения элементов матрицы - целые числа.

3. Строки

Обеспечить поддержку строки. Строки записываются в кавычках. Дополнительно необходимо обеспечить ввод кавычки в виде последовательности из двух кавычек. На выходе строки выдавать в следующем формате:

string ("<строка>")

где **строка** есть последовательность символов. Кавычка, если она была в исходной строке должна быть заменена последовательностью символов `\`.

Например

"Hello World!" на выходе **string** ("Hello World!")

"Hello "World"!" на выходе **string** ("Hello \"World!\")

"\"" на выходе **string** ("\\")

4. Массивы

Обеспечить поддержку массивов чисел. Массивы вводить в нижеследующем формате:

<массив> := (<числа>)

<числа> := <число> , <числа> | <число>

На выходе массивы записывать в следующем виде:

array (<размер>, <числа>)

Где, **размер** – это целое число – размер массива, **числа** – это элементы массива, перечисленные через запятую.

Например

(1, 2, 3, 4, 5) на выходе **array** (5,1,2,3,4,5)

5. Диапазоны

Обеспечить поддержку диапазонов (аналогично MathCAD). Диапазоны записывать в следующем формате:

<диапазон> := <старт> ... <финиш>

<диапазон> := <старт>, <приращение> ... <финиш>

где **старт** – целое число – начало диапазона, **финиш** – целое число – конец диапазона, **приращение** – целое число – приращение (по умолчанию 1 или -1, в зависимости от того, что больше **старт** или **финиш**). Приращение может быть отрицательным. Обеспечить контроль существования диапазона.

На выходе диапазоны записывать в следующем виде:

range (<старт>, <приращение>, <финиш>)

Например

5...10 на выходе **range** (5,1,10)

5...0 на выходе **range** (5,-1,0)

5,2...10 на выходе **range** (5,2,10)

5,-2...10 на дает ошибку «Неправильно задан диапазон»

6. Функции многих переменных

Обеспечить поддержку функций с любым количеством параметров (от 0).

<функция> ::= <имя> (<параметры>)

<параметры> ::= <выражение> | <выражение> < параметры > | е

На выходе функцию необходимо записывать в следующем виде:

Перечислить ее параметры слева направо (выражения должны вычисляться), а затем имя функции.

Например

row(3,5) на выходе **3 5 row**

row(4+1,2) на выходе **4 1 + 2 row**

Дополнительно после обработки всех строки вывести список всех функций, использованных в строках.