Проектирование компиляторов. Лабораторная работа №1. Создание лексического анализатора.

МАИ, 01.03.2023

1 Введение

Создание лексического анализатора с нуля является сложным и трудоёмким процессом, для его упрощения применяются специальные программы, называемые «генераторами лексических анализаторов». Классическим примером генератора лексических анализаторов является программа `flex`. На вход ей подаётся файл с правилами распознавания лексем, описанными специальным форматом. На выходе `flex` даёт исходный код на языке C, позволяющий распознавать описанные лексемы из входного потока.

2 Задание

На языке C11 реализовать синтаксический анализатор для калькулятора с функцией запоминания значений. Калькулятор предполагает ввод различных вычислительных выражений, вызов функций и запоминания значений в переменных.

2.1 Описание входного языка

Входной язык будет различаться в зависимости от варианта задания, однако некоторые лексемы будут одинаковы для всех вариантов. Каждая лексема имеет свой класс, который понадобится для печати результата работы лексического анализатора.

- **Идентификатор**: название функций и переменных, состоящее из маленьких и заглавных букв латинского алфавита, цифр и подчёркиваний. Идентификатор не может начинаться с цифры. Класс токена идентификатора называется `IDENTIFIER`.
- **Скобки**: `(`, `)`. Классы токенов скобок называются `LPARENT` и `RPARENT` соответственно.

- **Запятая**: `,`. Класс токена запятой называется `COMMA`. Запятая используется для разделения аргументов при вызове функции.
- **Присваивание**: `=`. Класс токена присваивания называется `ASSIGN`. Присваивание служит чтобы задать переменной какое-либо значение.
- **Ошибка**: Класс токена ошибки называется `ERROR`. Лексема, содержащая ошибку появляется если входные символы не являются символами рассматриваемого языка.

Входные данные и арифметические действия будут разные для разных вариантов.

Вариант 1

Калькулятор вычисляет логические выражения. Символьными литералами в данном случае являются символы `T` (класс токена `TRUE`) и `F` (класс токена `FALSE`). Возможные действия над аргументами: конъюнкция `&&` (класс токена `LAND`), дизъюнкция `||` (класс токена `LOR`), строгая дизъюнкция `^` (класс токена `LXOR`), отрицание `!` (класс токена `LNOT`).

Вариант 2

Калькулятор вычисляет арифметические выражения выражения. Символьными литералами в данном случае являются целые числа в диапазоне от -2^{31} до $2^{31}-1$ (класс токена `ILIT`). Возможные действия над аргументами: сложение `+` (класс токена `PLUS`), вычитание `-` (класс токена `MINUS`), умножение `*` (класс токена `MUL`), деление `/` (класс токена `DIV`).

Вариант 3

Калькулятор вычисляет битовые арифметические выражения выражения. Символьными литералими в данном случае являются целые числа в диапазоне от 0 до 2^{32} в шестнадцатиричной записи, перед которыми идёт префикс `0x` (класс токена `ILIT`). Возможные действия над аргументами: побитовая конъюнкция `&` (класс токена `BAND`), побитовая дизъюнкция `|` (класс токена `BOR`), побитовая строгая дизъюнкция `^` (класс токена `BXOR`), побитовое отрицание (класс токена `BNOT`).

2.2 Описание входных данных

На вход программе подаётся название файла с распознаваемым текстом. Если файл не подан, программа считывает данные со стандартного ввода. Подача нескольких файлов в качестве аргумента вызывает завершние программы с ошибкой некорректного использования.

Входной файл содержит произвольные данные. Примеры входных данных:

Вариант 1. Пример 1.

```
(T && T) || !F
f = F
T || f
is_true(f)
```

Вариант 1. Пример 2.

```
123
```

Вариант 2. Пример 1.

```
(3 + 4) / 7
i = 1
pow(i, 3)
```

Вариант 2. Пример 2.

```
0xF
```

Вариант 3. Пример 1.

```
(0x1 | 0x2) & 0xF
b = 0x7
bswap(b)
```

Вариант 3. Пример 2.

```
0xtrue
```

2.3 Описание выходных данных

Для каждого выделенного токена программа должна выводить информацию о нём в формате: `Класс, Лексема`.

Вариант 1. Пример 1.

```
LPARENT, "("
TRUE, "T"
LAND, "&&"
TRUE, "T"
RPARENT, ")"
LOR, "||"
LNOT, "!"
FALSE, "F"
IDENTIFIER, "f"
ASSIGN, "="
FALSE, "F"
TRUE, "T"
LOR, "||"
IDENTIFIER, "f"
IDENTIFIER, "is_true"
LPARENT, "("
IDENTIFIER, "f"
RPARENT, ")"
```

Вариант 1. Пример 2.

```
ERROR, "123"
```

Вариант 2. Пример 1.

```
LPARENT, "("
ILIT, "3"
PLUS, "+"
ILIT, "4"
RPARENT, ")"
DIV, "/"
ILIT, "7"
IDENTIFIER, "i"
ASSIGN, "="
ILIT, "1"
IDENTIFIER, "pow"
LPARENT, "("
IDENTIFIER, "i"
COMMA, ","
ILIT, "3"
RPARENT, ")"
```

Вариант 2. Пример 2.

```
ERROR, "0xF"
```

Вариант 3. Пример 1.

```
LPARENT, "("
ILIT, "0x1"
BOR, "|"
ILIT, "0x2"
RPARENT, ")"
BAND, "&"
ILIT, "0xF"
(0x1 | 0x2) & 0xF
IDENTIFIER, "b"
ASSIGN, "="
ILIT, "0x7"
IDENTIFIER, "bswap"
LPARENT, "("
IDENTIFIER, "b"
RPARENT, ")"
```

Вариант 3. Пример 2.

```
ERROR, "0xtrue"
```

3 Критерии выполнения

Когда автор посчитает что его работа готова, он отправляет преподавателю архив, содержащий исходный код программы, сценарий сборки и набор тестов, на котором лексический анализатор отлаживался. Письмо должно содержать фамилию и имя автора, тема должна содержать слова «МАИ, Лабораторная работа 1». Архив упаковывается одним из следующих форматов: `.bz2`, `.tgz`, `.xz`. Письма, не выполняющие указанные требования, не рассматриваются.

4 Сроки

При сдаче работы начиная с 17.03.2022 максимально возможная оценка за лабораторную работу снижается на один балл в неделю.

5 Литература

- *А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман* Компиляторы: принципы, технологии и инструменты.
- CS 143 — Compilers, Stanford University

6 Автор

Задание предоставлено ассистентом кафедры вычислительной математики и информатики факультета прикладной математики и физики Московского авиационного института Маркиным Алексеем Леонидовичем в 2023 году.

7 Справочное приложение

Полную справку по работе генератора лексических анализаторов `flex` можно получить введя команду `info flex` в своей операционной системе. Ниже представлена краткая справка по работе с `flex`, содержащая материалы, которые могут понадобиться для выполнения лабораторной работы.

7.1 Структура программы на flex

Программа на flex состоит из 3 основных секций:

```
Определения
%%
Правила
%%
Пользовательские процедуры
```

В секции определений задаются синонимы для регулярных выражений формата:

```
Синоним Шаблон
```

Например для того чтобы задать определение числа можно использовать следующую конструкцию:

```
DIGIT [0-9]
```

Кроме того, в секции определений возможно задавать функции на языке C и подключать заголовочные файлы. Весь код на C должен быть в пределах конструкции:

```
%{
#include <stdio.h>

void bar(void) {printf("world\n");}
void foo(void) {printf("Hello\n");}
%}
```

В секции правил в соответствие регулярному выражению ставится набор действий, который выполняется при его распознавании в формате:

```
Шаблон Действия
```

Действием является код на языке C, например:

```
[ \t]   {bar();}
{DIGIT} {foo();}
```

В данном примере первый шаблон задан в виде регулярного выражения, распознающего пробел или символ табуляции. Второй шаблон подставит регулярное выражение, записанное в синоним DIGIT, определённое в секции определений.

Секция пользовательских процедур содержит различные процедуры, которые должны идти вместе с модулем лексического анализатора. Например можно определить функцию main:

```
int main(void)
{
    yylex();
    return 0;
}
```

которая непосредственно запустит лексический анализатор. Функция yylex будет сгенерирована flex и начнёт считывать входной поток до тех пор пока не достигнет конца потока или команды выхода из функции.

7.2 Входной поток

По умолчанию программа, сгенерированная flex, принимает на вход данные стандартного ввода. Указатель на файл содержится в переменной yyin. Её можно модифицировать напрямую или через вызов процедуры yyrestart(in) где in — указатель на входной поток.

7.3 Сборка программы на flex

Для получения исполняемого файла необходимо сначала транслировать файл из формата flex в файл на языке C:

```
$ flex --header-file=lexer.h -o lexer.c lexer.l
```

В данной команде в качестве входного файла подаётся файл lexer.l. Опция -o задаёт выходной файл. Опцией --header-file= задаётся название генерируемого заголовочного файла. Он нужен только в случае если функции лексического анализатора необходимо вызывать из других модулей программы.

После трансляции лексического анализатора в файл на языке C, можно его скомпилировать и запускать на исполнение:

```
$ gcc lexer.c -o lexer
```

В некоторых случаях может возникать ошибка компоновки, сообщающая об отсутствии функции yywrap. Для устранения этой ошибки нужно в секции определений записать специальное указание:

```
%option noyywrap
```