Kaleidoscope-compiler

A compiler for the Kaleidoscope language. Written in C++ with lex/yacc and generates LLVM IR.

Features

- Identifier: [A-Za-z_][0-9A-Za-z_]+
- Literal
  - 64bit integer
  - Double precision floating point number
  - List
- Operator:
  - Arithmatic operators: +, -, *, /, %
  - Logical operators: <, <=, >, >=, ==, !=,
  - Relational operators: &&, ||, !
  - Assignment operator: =
- Conditional expression
- Loop expression
- Function
- Global/local variables
- External declaration

메뉴얼

산술 연산자

```
1. + .2;        # 덧셈
3.14 - 0;       # 뺄셈
3.14 * 10;      # 곱셈
-1 / 10;        # 나눗셈
7 % 5.2;        # 나머지. 실수 연산에서도 적용된다.
```

비교 연산자

```
x < y;
x <= y;
x > y;
x >= y;
x == y;
x != y;
```

논리 연산자

Kaleidoscope는 논리곱(`&&`), 논리합(`||`), 그리고 부정(`!`) 기호를 제공한다.
논리곱과 논리합 연산은 short-circuit에 의거하여 평가된다.

타입

확장된 Kaleidoscope는 배정밀도 부동소숫점, 64비트 정수, 그리고 배열 타입을 지원한다.
타입 선언은 `<type> <identifier>` 으로 표현된다.
다중 타입 지원에 따라 Kaleidoscope는 정적 타이핑 언어가 되었다.
단, 기존과의 하위 호환성을 위하여 타입 지정을 하지 아니할 경우, 실수형으로 간주한다.
실수형은 `double` 로 표기하고 정수형은 `int` 로 표기하며 배열 타입은 `<type>[<length>]` 으로 표기한다.

변수

확장된 Kaleidoscope는 명시적 변수 선언을 지원하며, global 영역과 local 영역 모두에 선언 가능하다.
global 영역에 선언된 변수는 함수 내부 및 외부에서 접근이 가능하며 local variable로 가릴 수 없다 (Variable shadowing).
Local 변수는 함수의 파라미터로 선언되었거나, 함수 내부에서 명시적으로 변수를 선언한 경우이며 함수의 종료 이후에는 해당 변수로 접근이 불가한다.

배열

Kaleidoscope에서는 배열 타입을 새로 지원하며, row-major과 0-indexed라는 특징을 가진다.
배열 타입은 Java와 같은 `<type>[<length>]` 표기법을 따른다.
예를 들어 (3 X 4) 형태의 double 배열을 선언하기 위해서는 `double[3][4]` 와 같이 타입을 지정하여야 한다.
이를 이용하여 배열 타입의 변수를 대입하는 것은 아래의 예를 참조한다.

```
double[2][3] matrix = [[1,2], [3, 4], [5, 6]];
```

배열간의 연산은 지원되지 않으며, 배열내 원소 참조는 `get_element(list, index)` 를 사용한다.

대입문

Vanilla kaleidoscope가 함수 parameter만을 이용하여 변수를 선언할 수 있었던 것과 달리 본 컴파일러는 함수 body 내부에서의 변수 선언을 허용한다.
지원하는 타입의 수가 늘어남에 따라 정적 타이핑 언어로 언어의 특성이 변경되었다.
타입 정보를 명시하기 위하여 `<type> <identifier> = <value>` 의 형식을 따르며, 함수형 언어처럼 한 번만 대입이 가능하다.
대입문 또한 expression의 일부로 취급되어 대입문의 평가 결과는 대입된 값이다. 따라서 `x = double y = 3.14` 와 같은 다중 대입 또한 가능하다.

함수

함수는 다음과 같은 형식을 지닌다.

```
def function_identifier(<type> arg1, <type> arg2) -> <return type>
        body;
```

`def` 키워드를 사용하여 함수의 시작을 알리고, identifier와 argument list가 그 뒤를 잇는다.
Argument list를 구성하는 각 argument는 `<type> <identifier>` 의 형식을 갖추어야 한다.
변수의 타입 지정과 마찬가지고 type을 명시하지 않은 경우에는 double 타입으로 간주한다.
Argument list 다음에는 `->` 와 반환 타입이 명시된다.
마지막으로 body에는 expression들이 나열될 수 있다.
각각의 expression은 `,` 로 구분되며 가장 마지막 expression 뒤에는 `;` 이 위치하여 함수의 끝을 알린다.
함수의 반환값으로는 마지막 expression의 값이 이용된다.

예를 들어 세 정수를 받아 int[3]을 반환하는 함수는 다음과 같이 작성 가능하다.

```
def func(int a, int b, int c) -> int[3]
        [a, b, c];
```

Expression

Expression은 수, 변수, 이항 연산, 함수 호출, 조건문, 대입문 및 반복문을 의미한다.
expression은 `,` 을 통하여 연속하여 작성할 수 있으며 이를 이용하며 일반적인 다른 언어의 statements를 모방할 수 있다.

```
int x = 3;
int y = 4;
x + y;
```

위와 같은 C언어로 된 여러 줄의 statement와 expression 조합이 있을 경우 Kaleidoscope로는 다음과 같이 표현할 수 있다.

```
int x = 3,
int y = 4,
x + y;
```

조건문

```
def conditional_example(double x, double y)
        if x < y then
                x
        else if x == y then
                0
        else
                y;
```

조건문은 expression으로 표현되기 때문에 삼항연산자의 역할을 수행할 수 있다.

```
def min_plus_three(lhs, rhs)
        min_val = if lhs < rhs then lhs else rhs,
        min_val + 3;
```

반복문

'for <초기식>, <반복 조건>, 변화값 in' 의 문법으로 구성되었다.

```
extern print(double i);
def loop_example(n)
        # initiazation, loop_condition, step
        for _i = 0, i < n, 1 in
                print(_i);
```

반복문 또한 expression으로 간주되며, 그 값은 항상 0.0이다.