yowaic

A weak implement of a subset of C programming language

使用方法

- 环境

  64位Linux

  ```
  Linux version 4.14.48-1-MANJARO (builduser@development) (gcc version 8.1.0 (GCC)) #1 SMP PREEMPT Tue Jun 5 18:53:59 UTC 2018
  ```

- 编译

  ```
  [λ yowaic]$ make yowaic
  ```

- 以前缀表达式表示语法树

  ```
  [λ yowaic]$ ./yowaic -a
  ```

  输入一个函数，以 **CTRL+D**结束

  ```
  int f(){return 0;}
  ```

  返回

  ```
  (int)f(){(return 0);}
  ```

  初始化变量

  ```
  [λ yowaic]$ ./yowaic -a
  int f() {
  int x = 10;
  x = x + 2;
  return x;
  }
  ```

  输出

  ```
  (int)f(){(decl int x 10);(= x (+ x 2));(return x);}
  ```

- 生成汇编

```
[λ yowaic]$ ./yowaic
int f() {
int x = 10;
x = x + 2;
return x;
}
```

输出

```
    .data
    .text
    .global f
f:
    pushq %rbp
    movq %rsp, %rbp
    subq $8, %rsp
    movl $10, %eax
    movl %eax, -8(%rbp)
    movl -8(%rbp), %eax
    push %rax
    movl $2, %eax
    popq %rcx
    add %rcx, %rax
    movl %eax, -8(%rbp)
    movl -8(%rbp), %eax
    leave
    ret
leave
    ret
```

- 生成汇编输出到文件，使用gcc汇编和链接

```
[λ Example]$ ls
a.out  fibonacci.c  foo.s
[λ Example]$ ../yowaic < fibonacci.c > foo.s
[λ Example]$ gcc -o a.out foo.s
[λ Example]$ ./a.out
0
1
1
2
3
5
8
13
21
34
55
89
144
*b = 30
```

```
t h i s   i s   a   s i n g l e   s t r i n g
t i   s a s n l   t i g ◆ ◆ ◆
```

- 测试

```
make test
```

- 清理二进制文件

```
make clean
```

文件说明

- 文件统计

```
[λ yowaic]$ wc -l *
  467 generator.c  # 将语法树转换为 x86-64 汇编
   14 generator.h
  674 LICENSE
   42 Makefile
  808 parser.c     # 将 token 流分析为语法树
  140 parser.h
  173 README.md    # 本说明
  131 scanner.l    # lex词法分析说明文件
  152 test.sh      # 测试文件
  237 token.c      # 将字符流分析为token流
   32 token.h
   47 unit_test.c  # 测试本程序中使用的数据结构
  117 util.c       # 本程序中通用的数据结构
  134 util.h
   29 yowaic.c     # 入口文件
 3210 total
```

词法分析

- 使用 flex 进行此法分析
- 文件为 scanner.l token.h token.c
- nextToken(读取 token
- peekToken(可预读一个 token
- ungetToken(可以将非期望的 token 退回
- 依据 C99 标准识别 C 语言注释，所有关键字，存放类型
- 标识符，存放在字符数组
- 十六进制，八进制，十进制数字，使用 scanf 读取，存放在 int 变量中
- 字符常量及转意字符，存放在 char 变量
- 浮点数，存放在 double 变量
- 字符串常量，存放在字符数组
- 运算符，返回类型；单个字符运算符以 ascii 表示类型
- ' ', '\t', '\v', '\n', '\f', 作为空白字符忽略
- <<EOF>>识别文件末尾

语法分析

parser.h parser.c

- 顶层语法结构为函数定义

变量类型有

- char，int以及 char 和 int 的 数组和指针

函数最多可以有6个参数，内部可以有变量声明和语句

变量声明必须初始化，如

```
int x = 2;
char str[] = "this is a string";
```

语句包括

空语句

```
;
```

复合语句

```
{}
```

if 语句

```
if (1) {
    printf("hey");
}
```

for 语句

```
for (int x = 0; x < 10; x = x + 1) {
    printf("%d", x);
}
```

return 语句

```
int foo() {
    return 1;
}
```

表达式语句

```
1 + 2 * 3 - 4;
```

生成代码

generator.h generator.c

根据语法树，输出汇编程序，如

```
[λ yowaic]$ ./yowaic
int f() {return 0;}
    .data
    .text
    .global f
f:
    pushq %rbp
    movq %rsp, %rbp
    movl $0, %eax
    leave
    ret
leave
    ret
```