Lab. 1 初代编译器实验说明和要求

一、初代编译器功能描述

初代编译器将 C 语言顺序语句序列翻译为等价的汇编程序，所输出的汇编程序符合 x86 或 MIPS 汇编语言格式要求，能够被后续的汇编器翻译为可执行程序运行。如果目标平台为 x86，则生成的可执行程序能够在目标机上执行和验证结果；如果目标平台为 MIPS，则生成的汇编程序可以在 MIPS 模拟器中运行和验证结果。

二、初代编译器文法要求

初代编译器能够处理的文法如下所示：

关键字：int, return

标识符：单个英文字母，如 a、b 等

常量：十进制整型，如 1、223、10 等

操作符：=、+、-、*、/

分隔符： ；

语句：表达式语句、赋值语句

三、初代编译器测试样例

测试样例不用考虑优先级，即表达式中算术操作符优先级相同，且无括号。

所有输入测试样例文件中单词之间均由空格或者回车分隔，但输入文件中可能存在多个连续的空格或者回车。

评分依据 return 的值是否符合预期。生成的 MIPS 汇编代码在 Spim 上运行。

输入样例：

```
int a ;
int b ;
int d ;
a = 1 ;
b = 2 ;
d = a + b ;
return d ;
```

（预期返回值为 3）

输出样例 x86：

```
.intel_syntax noprefix
.global main
.type main, @function
main:
  push    ebp
  mov     ebp,esp
  sub     esp,16
  mov     eax,1
  mov     DWORD PTR [ebp-4],eax
  mov     eax,2
  mov     DWORD PTR [ebp-8],eax
  mov     eax,DWORD PTR [ebp-4]
  add     eax,DWORD PTR [ebp-8]
  mov     DWORD PTR [ebp-12],eax
  mov     eax,DWORD PTR [ebp-12]
  leave
  ret
```

输出样例 mips：

```
sw $zero, -4($fp)       # int a
sw $zero, -8($fp)       # int b
sw $zero, -12($fp)      # int d

li $v0, 1               # a = 1
sw $v0, -4($fp)

li $v0, 2               # b = 2
sw $v0, -8($fp)

lw $v0, -4($fp)         # d = a + b
sw $v0, 0($sp)
addiu $sp, $sp, -4
lw $v0, -8($fp)
sw $v0, 0($sp)
addiu $sp, $sp, -4
lw $t1, 4($sp)
lw $t0, 8($sp)
add $t0, $t0, $t1
sw $t0, 8($sp)
addiu $sp, $sp, 4
lw $v0, 4($sp)
```

```
sw $v0, -12($fp)
addiu $sp, $sp, 4

lw $v0, -12($fp)        # return d
```

四、初代编译器实现参考

初代编译器的实现基于程序设计基础、算法和数据结构等课程所学知识。在词法分析部分可以使用正则匹配或其他思路实现，而代码生成部分则可以使用栈来完成。

其中词法分析部分的实现思路是：根据空格或者回车将输入源码字符串分割为多个子串，然后判断每个子串属于哪个单词类，整型常量按照对应规则匹配或最长数字串匹配，连续字母考虑匹配 return，单个字母即标识符，其他单词直接与目标字符串比较匹配即可。对于识别到的单词，可以用结构体进行封装，分别标识对应的类型和内码值。所有的单词按序存储在一个列表中。

其中代码生成部分的参考实现思路如下：

1. 变量声明语句

对于变量声明语句，实验的测试程序已经预先在栈帧中留出相关空间，无需自行修改栈顶指针。

```
int a ;
int b ;
```

x86

对应的的汇编是

```
mov DWORD PTR [ebp-4], 0   #第 1 个变量（地址为 DWORD PTR [ebp-4]）赋值 0
mov DWORD PTR [ebp-8], 0   #第 2 个变量（地址为 DWORD PTR [ebp-8]）赋值 0
```

MIPS

与 x86 大致相同。

```
sw $zero, -4($fp)
sw $zero, -8($fp)
```

2. return 语句

一个文件只含有一个 return 语句，遇到 return 语句时执行返回操作。

return d ;

x86

mov eax, DWORD PTR [ebp-12] # 将返回值复制到 eax 寄存器中

MIPS

lw $v0, -12($fp) # 将返回值复制到$v0（即$2）寄存器中。

3. 赋值与表达式语句

不区分运算符优先级的话，按照左结合顺序依次入栈出栈计算即可。

具体指令请参考对应汇编语言手册。

五、初代编译器提交要求

实现语言：C++（语言标准 c++14）

编译环境：g++-11

测试环境：gcc-11，spim

提交内容：单个 cpp 源文件，文件名称为 compilerlab1.cpp

输入输出：实现的编译器有一个命令行参数，用于指明输入文件路径，编译器从该路径读取源码，并向 stdout 输出编译结果。

注：g++用于编译你提交的编译器实验源码。若选择输出 x86 汇编，gcc 用于将你的编译器实验输出的 x86 汇编码编译成可执行文件，用于测试。若选择输出 MIPS 汇编，spim 用于模拟执行你的编译器实验输出的 MIPS 汇编码，用于测试。

六、初代编译器实验测试框架说明

为了方便测试，实验提供了一个测试框架，用于测试你的编译器实验。

x86 汇编的测试框架：

.intel_syntax noprefix # 使用 Intel 语法

.global main # 声明 main 函数为全局符号，这使得链接器能够识别程序的入口点。

.extern printf # 声明外部函数 printf，表示该函数在其他地方定义，通常是 C

标准库中。

```
    .data                          # 开始数据段，用于定义程序中的初始化数据。
    format_str:
      .asciz "%d\n"              # 定义一个用于 printf 的格式字符串，输出整数并换行。

    .text                          # 开始代码段，包含程序的实际指令。
    main:
      push ebp                     # 将基指针寄存器 ebp 的当前值压入堆栈，保存上一个函数栈帧的基指针
      mov ebp, esp                 # 将栈指针 esp 的值复制到基指针 ebp ，设置新的栈帧基指针
      sub esp, 0x100               # 从栈指针 esp 减去 256 字节，为局部变量分配栈空间
    ##################################################
    ##
    ##   你的编译器实验输出的 x86 汇编码将被插入到这里
    ##
    ##################################################
      # 打印 d (当前 eax 的值)
      push eax                       # 将结果 (eax 的值) 作为 printf 的参数
      push offset format_str       # 将格式字符串的地址作为 printf 的参数
      call printf                    # 调用 printf 函数
      add esp, 8                     # 清理栈

      # 恢复 eax 的值并退出 main
      pop eax
      leave
      ret
```

你的编译器实验输出的 x86 汇编码将会被插入到上述框架中。

为了便于评测，本实验框架将会自动调用 C 库的 printf 函数，输出你的编译器实验输出的 x86 汇编码中的返回值（即 eax 的内容）。

MIPS 汇编的测试框架：

```
.text
.globl main              # 声明 main 函数为全局符号，使得模拟器能识别程序的入口点
main:
  move $fp, $sp       # 设置帧指针
  addiu $sp, $sp, -0x100          # 为局部变量分配栈空间
##################################################
```

```
##
##   你的编译器实验输出的 MIPS 汇编码将被插入到这里
##
####################################################
  move $a0, $v0      # 将返回值（$v0）复制到$a0，作为打印整数的系统调用的参数
  li $v0, 1          # 设置系统调用号为 1，即打印整数
  syscall            # 系统调用

  li $v0, 10         # 设置系统调用号为 10，即退出程序
  syscall            # 系统调用
```

你的编译器实验输出的 MIPS 汇编码将会被插入到上述框架中。

为了便于评测，本实验框架将会自动调用 MIPS 模拟器的系统调用，输出你的编译器实验输出的 MIPS 汇编码中的返回值（即$v0 的内容）。

若你想自行执行汇编代码并调试：

x86:

在自行插入完代码后，在终端中执行

```
gcc -m32 -no-pie <输入汇编文件> -o <输出可执行文件>
./<输出可执行文件>
```

即可观察到输出结果。

注：在一些机器上，你可能需要添加 i386 架构的包才能正确执行以上操作，参考命令如下

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libstdc++6:i386
```

MIPS:

在自行插入完代码后，在终端中执行

```
spim -file <输入汇编文件>
```

即可观察到输出结果。

七、初代编译器参考文档附件

对应汇编说明

1、MIPS 汇编

参考网址：https://freeflyingsheep.github.io/posts/mips/assembly/

2、SPIM Simulator

生成的 MIPS 指令可以在模拟器上运行，有两个版本：

（1）命令行版

安装：sudo apt-get install spim

测试：spim -file [汇编代码文件名]

（2）GUI 版（可以单步调试）

下载地址：http://pages.cs.wisc.edu/~larus/spim.html

使用方法：载入程序生成的目标代码文件

MIPS 和 SPIM Simulator 的内容还可以参考《编译原理实践与指导教程》相关部分。