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 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。

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

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，其中表达式语句包含括号及括号嵌套；

三、初代编译器测试样例

测试用例难度分为两个等级：其中第一个等级每个表达式中操作符优先级相同，且无括号；第二个等级同一个表达式中会有不同优先级，且包含有嵌套的括号。测试用例中，第一个等级测试用例占比 90%，第二个等级的测试用例占比 10%。

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，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。

等级一输入样例：

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

（预期返回值为 3）

等级一输出样例 x86：

```
mov DWORD PTR [ebp-4], 0      # int a
mov DWORD PTR [ebp-8], 0      # int b
```

```
mov DWORD PTR [ebp-12], 0     # int d
mov DWORD PTR [ebp-4], 1      # a = 1
mov DWORD PTR [ebp-8], 2      # b = 2

mov eax, DWORD PTR [ebp-4]    # d = a + b
push eax
mov eax, DWORD PTR [ebp-8]
push eax
pop ebx
pop eax
add eax, ebx
push eax
pop eax
mov DWORD PTR [ebp-12], eax

mov eax, DWORD PTR [ebp-12]   # return d
```

等级二输入样例：

```
int a ;
int b ;
int c ;
int d ;
a = 1 ;
b = 2 ;
c = 3 ;
d = ( a + b * 2 ) / c – 3 ;
return d ;
```

（预期返回值为-2）

等级二输出样例 x86：

```
mov DWORD PTR [ebp-4], 0    # int a
mov DWORD PTR [ebp-8], 0    # int b
mov DWORD PTR [ebp-12], 0   # int c
mov DWORD PTR [ebp-16], 0   # int d
mov DWORD PTR [ebp-4], 1    # a = 1
mov DWORD PTR [ebp-8], 2    # b = 2
mov DWORD PTR [ebp-12], 3   # c = 3

mov eax, DWORD PTR [ebp-4]  # d = ( a + b * 2 ) / c - 3
push eax

mov eax, DWORD PTR [ebp-8]
```

```
push eax

mov eax, 2
push eax
pop ebx
pop eax
imul eax, ebx
push eax

pop ebx
pop eax
add eax, ebx
push eax

mov eax, DWORD PTR [ebp-12]
push eax

pop ebx
pop eax
cdq
idiv ebx
push eax

mov eax, 3
push eax
pop ebx
pop eax
sub eax, ebx
push eax

pop eax
mov DWORD PTR [ebp-16], eax

mov eax, DWORD PTR [ebp-16]   # 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，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
```

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
```

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

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，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。

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）。

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
```