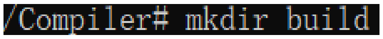
Compiler output is in the form of LLVM assembly.

Program Structure:

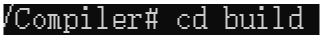
header.h – contains the majority of struct definitions as well as major function declarations
enums.h – contains enumerations used throughout the program
parse_tree.h – contains struct definitions describing the parse tree of a program
code_generator.h – contains function declarations for code generation
extern.cpp – contains global variables
cpp files – contain definitions of functions
runtimelib.c – provides functions for the target language runtime

Build Process:

1. from directory with CMakeLists.txt, create and move to build directory

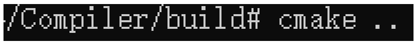


```
/Compiler# mkdir build
```



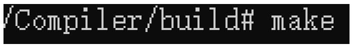
```
/Compiler# cd build
```

2. generate cmake files



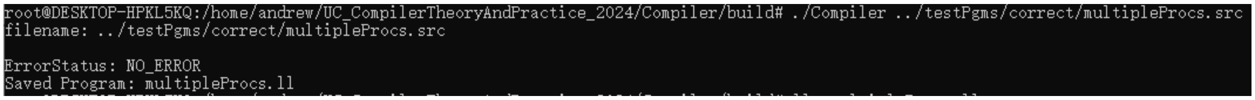
```
/Compiler/build# cmake ..
```

3. build compiler and runtime



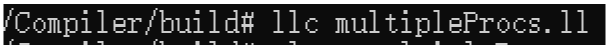
```
/Compiler/build# make
```

4. run compiler on target source file



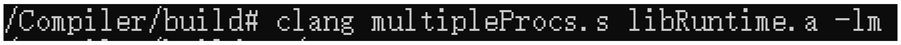
```
root@DESKTOP-HPKL5KQ:/home/andrew/UC_CompilerTheoryAndPractice_2024/Compiler/build# ./Compiler ../testPgms/correct/multipleProcs.src
filename: ../testPgms/correct/multipleProcs.src

ErrorStatus: NO_ERROR
Saved Program: multipleProcs.ll
```

5. generate machine assembly from LLVM assembly using llc



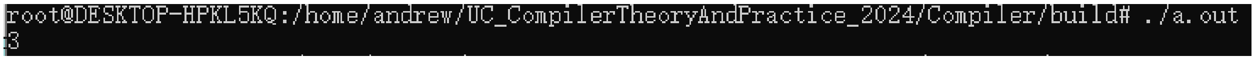
```
/Compiler/build# llc multipleProcs.ll
```

6. build executable, linking with runtime and -lm



```
/Compiler/build# clang multipleProcs.s libRuntime.a -lm
```

7. run executable



```
root@DESKTOP-HPKL5KQ:/home/andrew/UC_CompilerTheoryAndPractice_2024/Compiler/build# ./a.out
3
```

Compiler Features:

The compiler generates valid LLVM assembly for each of the test programs.
The executables created from linking the output with the runtime run successfully.

For array bounds-checking: when an array access is detected to be out of bounds, the runtime prints and throws an exception using c setjmp/longjmp. This is because I was having problems using the LLVM setjmp/longjmp intrinsics.

While code generation is done during parsing, the parse() function also builds and returns a parse tree. The tree contains some information from code generation for passing around registers.