CoPro: Code Generation

T. Rohan - CS20BTECH11064
I. Rajasekhar - CS20BTECH11020
P. Vikas - CS20BTECH11037
S. Satvik - CS20BTECH11048
B. Revanth - CS20BTECH11007
J. Vamsi Preetham - CS20BTECH11058
G. Yagnavalkya - CS20BTECH11019

Code Generation

- **Fourth phase of the compiler project.**
- A mechanism to produce the executable form of our CoPro programs.
- Uses the ASTs built up for the code to produce executables that could be run.

Where we were after Semantic Analysis

- In the semantic analysis stage, we've taken the tree as input from the parser and performed semantic checks on it.
- The output after the semantic analysis stage was the AST.
- In this phase of our compiler, we take the AST as input and generate code while traversing it.
- We are finally expected to produce executables, that when run would produce the output of the program.

Code Generation

- Code generation is the process of taking the semantically checked AST and generating an executable from it.
- It is the fourth phase of building our Compiler.
- This phase of our project traverses the AST to generate the .ll file as output.
- Along with assembly files (.s) that are generated using the .ll files, we also generate gcc object files (.out) and llvm object files (.bc) which could be run to generate the output of the code.

Compile and Run

- We have generated LLVM outputs for our test cases which are in “llvmOutputs”.
- To run these LLVM files use command

```
$ llvm-as -o llvmObjects/2.bc llvmOutputs/2.ll

$ lli llvmObjects/2.bc
```

- To generate assembly files use command

```
$ llc -o AssemblyFiles/2.s llvmOutputs/2.ll
```

- We can run these assembly files using gcc using command

```
$ g++ -no-pie AssemblyFiles/2.s -o gccObjects/2.out

$ ./gccObjects/2.out
```

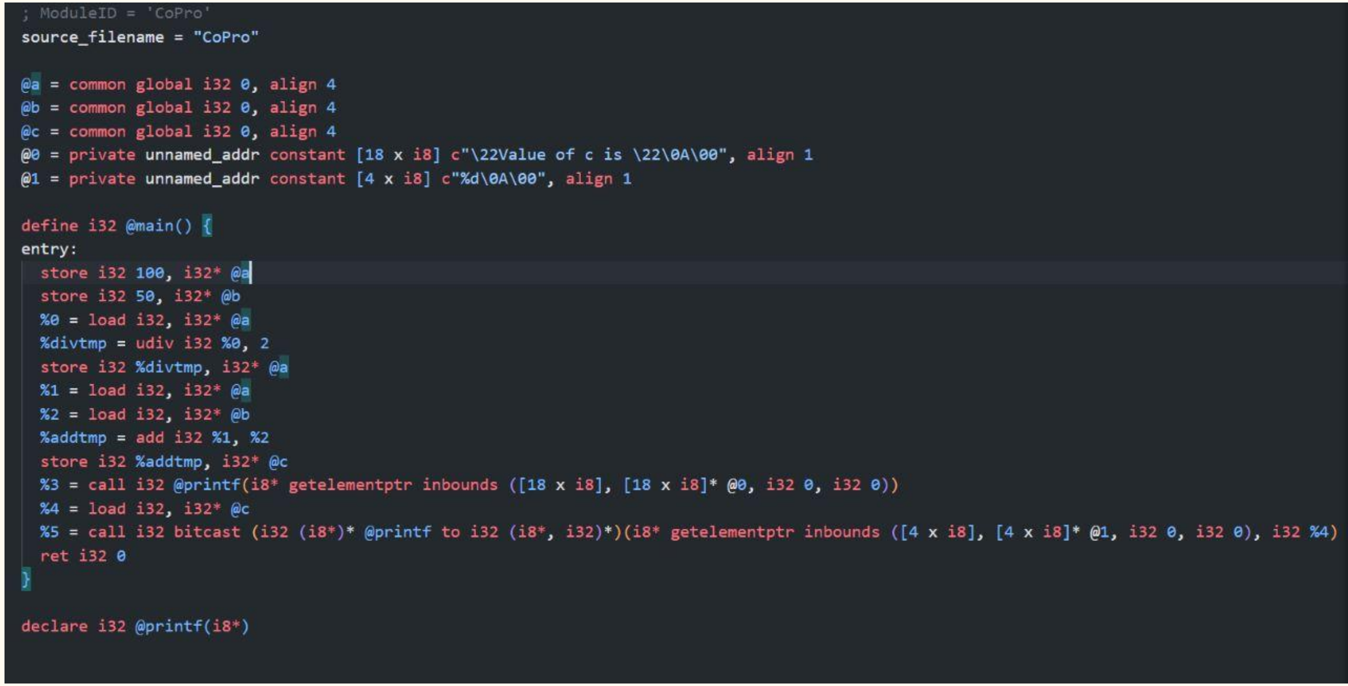
File Hierarchy

In this phase of our compiler, there are multiple new files and folder generated, the additions are(*italic* for folders),:

- **AST.hpp :** Contains updated AST
- ***testCases* :** Contains the test cases
- ***SymbolTables_ASTs* :** contains the symbol tables and AST for the code
- ***Assembly Files* :** contains the .s files generated using the .ll files.
- ***llvmOutputs:*** Contains the .ll files generated
- ***llvmObjects:*** Contains the llvm object files (.bc) generated
- ***gccObjects:*** Contains the gcc object files (.out) generated

Example

```
main->int
<<
        int a=100, b = 50 ,c
        a = a / 2
        c = a + b
        output : “value of c is”
        output : c
>>
```



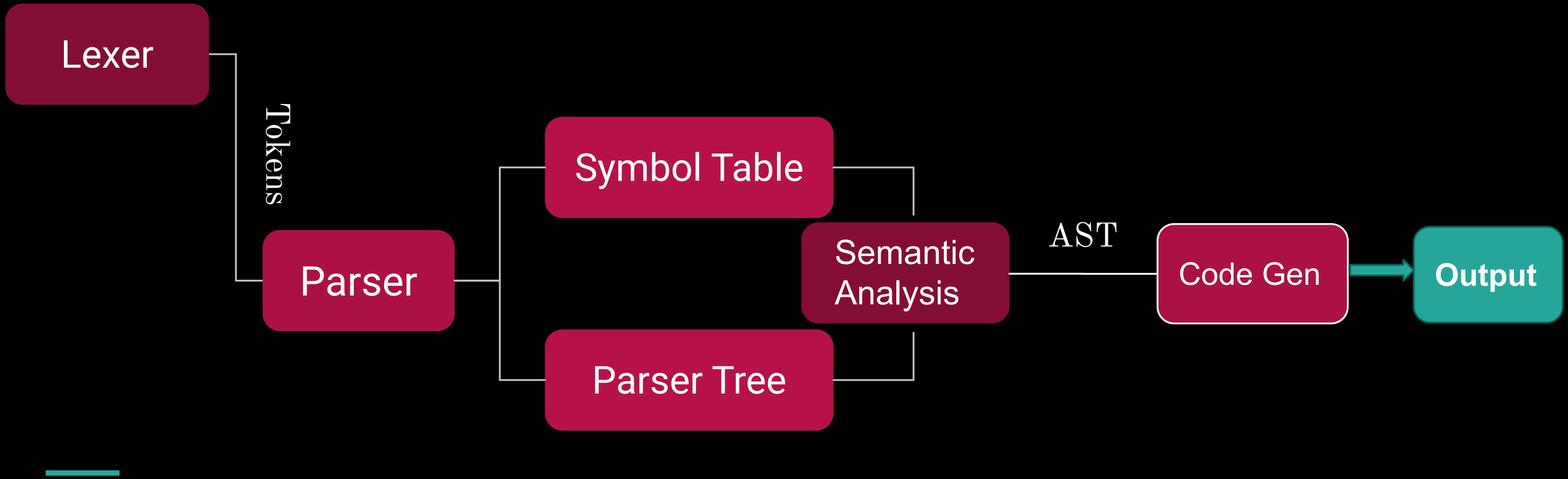
```
; ModuleID = 'CoPro'
source_filename = "CoPro"

@a = common global i32 0, align 4
@b = common global i32 0, align 4
@c = common global i32 0, align 4
@0 = private unnamed_addr constant [18 x i8] c"\22Value of c is \22\0A\00", align 1
@1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

define i32 @main() {
entry:
  store i32 100, i32* @a
  store i32 50, i32* @b
  %0 = load i32, i32* @a
  %divtmp = udiv i32 %0, 2
  store i32 %divtmp, i32* @a
  %1 = load i32, i32* @a
  %2 = load i32, i32* @b
  %addtmp = add i32 %1, %2
  store i32 %addtmp, i32* @c
  %3 = call i32 @printf(i8* getelementptr inbounds ([18 x i8], [18 x i8]* @0, i32 0, i32 0))
  %4 = load i32, i32* @c
  %5 = call i32 bitcast (i32 (i8*)* @printf to i32 (i8*, i32)*)(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @1, i32 0, i32 0), i32 %4)
  ret i32 0
}

declare i32 @printf(i8*)
```

Control Flow



Thank You