**ΧΡΗΣΤΟΣ ΠΑΝΑΓΙΩΤΗΣ ΠΑΛΑΜΙΔΑΣ**

**1115201800140**

***Αρχεία:***

* SymbolTable.java
* MyFirstVisitor.java
* llvmVisitor.java

Το SymbolTable.java και το MyFirstVisitor έχουν την ίδια χρησιμότητα με τον προηγούμενη εργασία με κάποιες μικρές αλλαγές και προσθήκες (κυρίως στο SymbolTable) έτσι ώστε να αποθηκεύω πιο αποδοτικά κάποιες πληροφορίες. Θα συμπεριλάβω και το προηγούμενο README που εξηγώ τη λειτουργία των 2 αυτών αρχείων.

*llvmVisitor.java:*

O LLVM Visitor είναι αυτός που διαβάζει γραμμή-γραμμή το minijava πρόγραμμα και παράγει LLVM κώδικα, όπως ακριβώς ζητάει η εργασία.

***Σημειώσεις:***

* Λόγω χρόνου, δεν πρόλαβα να υλοποιήσω την υποστήριξη boolean arrays. Γνωρίζω όμως ότι η προσέγγιση μου θα ήταν να αποθηκεύω 0 και 1 σε i32\* και όταν θα ήθελα να χρησιμοποιήσω την τιμή τους ως bool θα έπρεπε να κάνω bitcast σε i1.
* Στην προηγούμενη εργασία λόγω του ότι δε διάβασα σωστά την εκφώνηση, είχα κάνει λάθος τα offsets τα οποία και διόρθωσα, όπως και διόρθωσα ότι το πρόγραμμα μου μπορεί να διαβάσει και να λειτουργήσει για παραπάνω από ένα minijava input.
* Υπάρχει ένας φάκελος που λέγεται test\_files που έχει μέσα όλα τα test cases που έχω ελέγξει. Ανεξάρτητα με το αν θέλετε να τρέξετε αυτά ή άλλα δικά σας αρχεία, πρέπει να φτιάξετε ένα φάκελο (έστω <folder\_name>) μέσα στον οποίο να υπάρχουν 2 υποφάκελοι. Ένας φάκελος με όνομα ‘*java*’ και ένας φάκελος με όνομα ‘*ll*’. Αυτό συμβαίνει γιατί δίνοντας ένα relative/absolute path, το πρόγραμμα διαβάζει ένα αρχείο από τον ‘java’ φάκελο και το αποθηκεύει στον ‘ll’.

***How to run:***

1. make all
2. java Main ./test\_files/java/Add.java ./test\_files/java/Classes.java
3. cd test\_files/ll/
4. clang -o Add Add.ll
5. ./Add

~Το βήμα [2] διαβάζει το minijava αρχείο, έστω Add.java, και γράφει μέσα στο Add.ll τον ισοδύναμο LLVM κώδικα.