ΜΕΤΑΦΡΑΣΤΕΣ

Άσκηση 2024:

Γεώργιος Θεοδωρόπουλος 4967.

Δημήτριος Γώγος 5085.

PART 1

Σε αυτό το part κάναμε τα πρώτα βήματα της εργασίας φτιάχνοντας τον Lex and Yacc. Όταν τρέχουμε το πρόγραμμα εκτυπώνονται τα tokens του εκάστοτε test όπως και το αν περνάει από την συντακτική ανάλυση (αν έχει σωστή γραμματική). Είχαμε προβλήματα στην γραμματική της main και δεν καταφέραμε να τα λύσουμε. Για αυτό τα test που στείλαμε δεν έχουν καλή και ποιοτική main.

with command python Part1_5085_4967.py test.cpy

```
def read_cpy_file(file_path):
    with open(file_path, 'r') as file:
        content = file.read()
    return content
```

- Ορίζει μια συνάρτηση με όνομα read_cpy_file, η οποία παίρνει ως παράμετρο το file_path (διαδρομή αρχείου).
- Χρησιμοποιεί την εντολή with open(file_path, 'r') as file: για να ανοίξει το αρχείο που βρίσκεται στη διαδρομή file_path σε λειτουργία ανάγνωσης ('r'). Η χρήση του with διασφαλίζει ότι το αρχείο θα κλείσει αυτόματα μετά την ανάγνωση, ακόμη κι αν προκύψει κάποιο σφάλμα.
- Χρησιμοποιεί τη μέθοδο file.read() για να διαβάσει ολόκληρο το περιεχόμενο του αρχείου και το αποθηκεύει στη μεταβλητή content.
- Επιστρέφει το περιεχόμενο του αρχείου (content).

Στην συνέχεια φτιάχνουμε έναν πίνακα από tokens που μας ενδιαφέρουν ώστε να τα χρησιμοποιήσουμε στο Lex.

LEX:

Λειτουργία της Συνάρτησης Lex

1. **Προετοιμασία**:

Αρχικοποιεί μια λίστα code_tokens για να αποθηκεύσει τα tokens του κώδικα. Αρχικοποιεί την current_token ως κενή συμβολοσειρά. Ορίζει το in_comment ως False για να παρακολουθεί αν βρίσκεται μέσα σε σχόλιο. Αρχικοποιεί το index στο 0 για να διατρέξει τον κώδικα.

2. **Επανάληψη πάνω στον κώδικα**:

Χρησιμοποιεί μια while επανάληψη για να διατρέξει κάθε χαρακτήρα του κώδικα.

3. **Διαχείριση Σχολίων**:

Αν βρίσκεται σε σχόλιο (in_comment είναι True), παραλείπει τους χαρακτήρες μέχρι να βρει αλλαγή γραμμής ('\n').

4. **Ανάλυση Λευκών Χώρων**:

Αν συναντήσει λευκό χώρο και υπάρχει current_token, προσθέτει το current_token στη λίστα code_tokens με τον κατάλληλο τύπο (RESERVED_WORD, DIGIT, ALPHABET).

5. **Ανάλυση Συμβόλων Ομαδοποίησης**:

Αν συναντήσει σύμβολο ομαδοποίησης, προσθέτει το τρέχον token στη λίστα και προσθέτει το σύμβολο ως GROUPING_SYMBOL.

6. **Ανάλυση Σχολίων**:

Αν συναντήσει #, ελέγχει αν είναι σχόλιο (##) ή αν είναι λέξη-κλειδί #def, #int, #{, #} και προσθέτει τα αντίστοιχα tokens.

7. **Ανάλυση Τελεστών Σχέσης**:

Αν συναντήσει τελεστή σχέσης, ελέγχει αν είναι σύνθετος τελεστής (π.χ. ==, !=) και προσθέτει τα αντίστοιχα tokens.

8. **Ανάλυση Διαχωριστικών**:

Αν συναντήσει διαχωριστικό (π.χ. ,), προσθέτει το τρέχον token και το διαχωριστικό ως DELIMITER.

9. **Ανάλυση Αριθμητικών Τελεστών**:

Αν συναντήσει αριθμητικό τελεστή (π.χ. +, -), προσθέτει το τρέχον token και τον τελεστή ως ARITHMETIC_OPERATOR.

10. **Αποθήκευση Τελευταίου Token**:

Αν υπάρχει current_token μετά το τέλος της επανάληψης, προσθέτει το τελευταίο token στη λίστα code_tokens.

Επιστροφή. Η συνάρτηση επιστρέφει τη λίστα code_tokens που περιέχει όλα τα tokens του κώδικα που αναλύθηκε.

Γραμματική:

- program : global_variables1 functions main_function
  - ;

- global_variables1
  - : ('#int' id_list)*
  - ;

- global_variables2
  - : ('global' 'ALPHABET')*
  - ;

- functions : 'def' 'ALPHABET' formal_pars ':'
  - '#{'
  - block
  - '#}'
  - ;

- formal_pars : '(' ('ALPHABET' (',' 'ALPHABET')* | 'DIGIT' (',' 'DIGIT')*)? ')'
  - ;

- block : (declarations global_variables2 statement)*
  - ;

- declarations : ('#int' id_list)*
  - ;

- statement : assignment_statement
  - | print_statement
  - | return_statement
  - | if_statement
  - | while_statement
  - | functions
  - ;

- assignment_statement

        - : 'ALPHABET' '=' expression
        - ;

- print_statement : 'print' expression
        - ;

- return_statement
        - : 'return' expression
        - ;

- if_statement : 'if' expression condition ':'
        - statement_or_block
        - ('elif' expression condition ':' statement_or_block)*
        - ('else' ':' statement_or_block)?
        - ;

- while_statement : 'while' 'ALPHABET' condition ':'
        - block
        - '#}'
        - ;

- expression : term (('+' | '-') term)*
        - ;

- term : factor (('*' | '//' | '%') factor)*
        - ;

- factor : 'ALPHABET'
        - | 'DIGIT'
        - | '(' expression ')'
        - ;

- condition : bool_term relational_operator
        - ;

- bool_term : 'not' condition

        - ;

- relational_operator
        - : '>' | '<' | '>=' | '<=' | '==' | '!='
        - ;

- main_function : '#def' 'main'
        - global_variables1
        - statement
        - ;

YACC:

Δομή της Συνάρτησης Yacc

1. **Αρχικοποίηση**:

Ορίζει τη συνάρτηση Yacc που παίρνει ως παράμετρο τη λίστα code_tokens, δηλαδή τα tokens του κώδικα που θα αναλυθούν.

Χρησιμοποιεί μια παγκόσμια μεταβλητή index για να παρακολουθεί τη θέση στον πίνακα των tokens.

2. **Συνάρτηση** match:

Ελέγχει αν το τρέχον token ταιριάζει με τον αναμενόμενο τύπο και (προαιρετικά) την αναμενόμενη τιμή.

Αν ταιριάζει, αυξάνει τον δείκτη index για να προχωρήσει στο επόμενο token.

Αν δεν ταιριάζει, καλεί τη συνάρτηση error για να εμφανίσει μήνυμα σφάλματος.

3. **Συνάρτηση** program:

Η κύρια συνάρτηση του προγράμματος που καλεί τις άλλες συναρτήσεις για να αναλύσει τις παγκόσμιες μεταβλητές, τις συναρτήσεις και την κύρια συνάρτηση.

4. **Συναρτήσεις για τις Παγκόσμιες Μεταβλητές**:

global_variables1: Αναλύει τις παγκόσμιες μεταβλητές που δηλώνονται με #int.

global_variables2: Αναλύει τις παγκόσμιες μεταβλητές που δηλώνονται με global.

5. **Συνάρτηση** block:

Αναλύει ένα μπλοκ κώδικα μέχρι να βρει το κλείσιμο του μπλοκ (#}).

6. **Συνάρτηση** functions:

Αναλύει τις συναρτήσεις του προγράμματος, καλώντας τις σχετικές υποσυναρτήσεις για τις παραμέτρους και το σώμα της συνάρτησης.

7. **Συνάρτηση** formal_pars:

Αναλύει τις παραμέτρους μιας συνάρτησης, δεχόμενη γράμματα (ALPHABET) ή ψηφία (DIGIT).

8. **Συνάρτηση** declarations:

Αναλύει τις δηλώσεις μεταβλητών μέσα σε ένα μπλοκ ή συνάρτηση.

9. **Συνάρτηση** statement:

Αναλύει διάφορα είδη δηλώσεων (π.χ. if, while, def, return, print, assignment).

10. **Συναρτήσεις για τις Δομές Ελέγχου**:

if_statement: Αναλύει δηλώσεις if, elif, else.

while_statement: Αναλύει δηλώσεις while.

11. **Συναρτήσεις για τις Απλές Δηλώσεις**:

return_statement: Αναλύει δηλώσεις return.

print_statement: Αναλύει δηλώσεις print.

assignment_statement: Αναλύει δηλώσεις ανάθεσης (assignment).

12. **Συναρτήσεις για τις Εκφράσεις**:

expression: Αναλύει εκφράσεις που περιλαμβάνουν αριθμητικούς τελεστές (+, -).

term: Αναλύει όρους εκφράσεων που περιλαμβάνουν πολλαπλασιασμό, διαίρεση, ή υπόλοιπο (*, //, %).

factor: Αναλύει παράγοντες εκφράσεων που μπορεί να είναι μεταβλητές, ψηφία ή υποεκφράσεις.

13. **Συναρτήσεις για τις Συνθήκες**:

bool_term: Αναλύει λογικούς όρους που περιλαμβάνουν τη λέξη-κλειδί not.

relational_operator: Αναλύει τελεστές σχέσης (>, <, <=, >=, ==, !=).

condition: Αναλύει συνθήκες που περιλαμβάνουν λογικούς τελεστές (and, or).

14. **Συνάρτηση** main_function:

Αναλύει την κύρια συνάρτηση (main), περιλαμβάνοντας τις παγκόσμιες μεταβλητές και μια δήλωση.

15. **Συνάρτηση** error:

Εμφανίζει μήνυμα σφάλματος και σταματά την εκτέλεση σε περίπτωση που βρεθεί μη αναμενόμενο token.

PART 2

Σε αυτό το part πραγματοποιήσαμε την λειτουργία του ενδιάμεσου κώδικα και του πίνακα συμβολών για κάθε ένα από αυτά όταν τρέχουμε το πρόγραμμα βγαίνουν 2 αρχεία με όνομα .int .sym που περιέχουν αντίστοιχα τα αποτελέσματα.

Οι κλάσεις που αναφέρονται στον κώδικα έχουν ως στόχο τη δημιουργία και διαχείριση ενός πίνακα συμβόλων, που χρησιμοποιείται από τον συντακτικό αναλυτή (Yacc) για την αποθήκευση και διαχείριση των συμβόλων που συναντώνται κατά την ανάλυση του κώδικα. Παρακάτω θα εξηγήσουμε τις κλάσεις αυτές και πώς χρησιμοποιούνται μέσα στον Yacc για να παραχθεί ο πίνακας συμβόλων.

Κλάσεις

1. **Entity**: Αυτή είναι η βασική κλάση που αντιπροσωπεύει μια οντότητα στον πίνακα συμβόλων. Κάθε οντότητα έχει ένα όνομα και έναν τύπο.
2. **Variable**: Κληρονομεί από την κλάση Entity και προσθέτει χαρακτηριστικά που αφορούν τις μεταβλητές, όπως ο τύπος δεδομένων και η αντιστάθμιση (offset).
3. FormalParameter: Αντιπροσωπεύει μια παράμετρο σε μια συνάρτηση ή διαδικασία, περιλαμβάνοντας τον τύπο δεδομένων και τον τρόπο περάσματος (π.χ. by value ή by reference).
4. **Parameter**: Κληρονομεί από την κλάση FormalParameter και προσθέτει την αντιστάθμιση.
5. Procedure: Αντιπροσωπεύει μια διαδικασία, περιλαμβάνοντας το σημείο εκκίνησης της, το μήκος του πλαισίου και τις παραμέτρους της.
6. **Function**: Κληρονομεί από την κλάση Procedure και προσθέτει τον τύπο δεδομένων της συνάρτησης.
7. **TemporaryVariable**: Αντιπροσωπεύει μια προσωρινή μεταβλητή που χρησιμοποιείται για ενδιάμεσους υπολογισμούς.
8. **SymbolicConstant**: Αντιπροσωπεύει μια συμβολική σταθερά.
9. **SymbolTable**: Διαχειρίζεται τους διαφορετικούς συμβολικούς πίνακες για κάθε περιοχή μεταβλητών (scope).

Χρήση κλάσεων στον Yacc

Ο Yacc (Yet Another Compiler-Compiler) χρησιμοποιεί τις κλάσεις αυτές για να διαχειρίζεται και να καταγράφει τα σύμβολα που βρίσκει στον κώδικα που αναλύει. Συγκεκριμένα:

Διαχείριση περιοχών (scopes): Η κλάση SymbolTable επιτρέπει την είσοδο και έξοδο από περιοχές μέσω των μεθόδων enter_scope και exit_scope. Κάθε φορά που εισερχόμαστε σε μια νέα περιοχή, δημιουργείται ένα νέο scope, και όταν εξερχόμαστε, το scope αφαιρείται.

Προσθήκη συμβόλων: Μέσω της μεθόδου add_symbol, τα σύμβολα που εντοπίζονται κατά την ανάλυση προστίθενται στον πίνακα συμβόλων. Τα σύμβολα αυτά μπορεί να είναι μεταβλητές, παράμετροι, διαδικασίες ή συναρτήσεις.

Αναζήτηση και ενημέρωση συμβόλων: Η μέθοδος lookup επιτρέπει την αναζήτηση ενός συμβόλου στον πίνακα, ενώ η update_symbol επιτρέπει την ενημέρωση των ιδιοτήτων ενός συμβόλου.

Εκτύπωση πίνακα συμβόλων: Η μέθοδος print_symbol_table_to_file εκτυπώνει τον πίνακα συμβόλων σε ένα αρχείο για σκοπούς εντοπισμού σφαλμάτων και παρακολούθησης της κατάστασης.

Βοηθητικές συναρτήσεις

- **genQuad**

Δημιουργεί μια νέα τετραπλή και την προσθέτει στη λίστα quad_list.
Επιστρέφει τον αριθμό της νέας τετραπλής.

- nextQuad

Επιστρέφει τον αριθμό της επόμενης διαθέσιμης τετραπλής.

- newTemp

Δημιουργεί μια νέα προσωρινή μεταβλητή και επιστρέφει το όνομά της.

- emptyList

Επιστρέφει μια κενή λίστα.

- makeList

Δημιουργεί μια λίστα με ένα στοιχείο (ετικέτα)

- mergeList

Συγχωνεύει δύο λίστες και επιστρέφει τη συνδυασμένη λίστα.

- backpatch

Ενημερώνει τις τετραπλές στις θέσεις που αναφέρονται στη λίστα, αντικαθιστώντας το τελευταίο πεδίο με την ετικέτα.

- printQuads

Εκτυπώνει τις τετραπλές σε ένα αρχείο.

Οι βοηθητικές συναρτήσεις που παρατίθενται χρησιμοποιούνται για την παραγωγή ενδιάμεσου κώδικα σε μορφή τετραπλών (quadruples) από τον συντακτικό αναλυτή (Yacc). Ο ενδιάμεσος κώδικας είναι μια αναπαράσταση του πηγαίου κώδικα που είναι πιο κοντά σε γλώσσα μηχανής, αλλά παραμένει ανεξάρτητος από τη συγκεκριμένη αρχιτεκτονική του επεξεργαστή. Αυτό διευκολύνει τη μετάφραση του πηγαίου κώδικα σε διάφορες γλώσσες στόχους. Ας δούμε τι κάνουν αυτές οι συναρτήσεις και πώς χρησιμοποιούνται:

Χρήση στον Yacc

Οι παραπάνω βοηθητικές συναρτήσεις χρησιμοποιούνται από τον Yacc κατά την παραγωγή ενδιάμεσου κώδικα. Ακολουθούν παραδείγματα χρήσης αυτών των συναρτήσεων σε διάφορα τμήματα του κώδικα Yacc:

- **Παράδειγμα με συνάρτηση**

```
def functions():
  while index < len(code_tokens) and code_tokens[index][1] == 'def':
    match('RESERVED_WORD', 'def')
    func_name = code_tokens[index][1]
    match('ALPHABET')
    params = formal_pars()
    match('DELIMITER', ':')
    genQuad('begin_block', func_name, '_', '_')
    for param in params:
      genQuad('par', param.name, param.mode, '_')
      symbol_table.add_symbol(param)
    match('GROUPING_SYMBOL', '#{')
    symbol_table.enter_scope()
    block()
    match('GROUPING_SYMBOL', '#}')
    symbol_table.exit_scope()
    genQuad('end_block', func_name, '_', '_')
```

- **Παράδειγμα με εντολή εκχώρησης**

```
def assignment_statement():
  id = code_tokens[index][1]
  match('ALPHABET')
  match('ASSIGNMENT_OPERATOR', '=')
  if index < len(code_tokens) and code_tokens[index][1] == 'int':
    input_statement()
  else:
    expression()
    genQuad('=', expression.place, '_', id)
```

- **Παράδειγμα με έκφραση**

```
def expression():
  term()
  expression.place = term.place
  while index < len(code_tokens) and code_tokens[index][1] in ['+', '-']:
    op = code_tokens[index][1]
    match('ARITHMETIC_OPERATOR', op)
    term()
    w = newTemp()
    genQuad(op, expression.place, term.place, w)
    expression.place = w
```

- **Παράδειγμα με δομή ελέγχου**

```
def if_statement():
  match('RESERVED_WORD', 'if')
  B()
  B_true = nextQuad()
  genQuad('jump', '_', '_', '_')
  B_false = nextQuad()
  match('DELIMITER', ':')
  statement()
  while index < len(code_tokens) and code_tokens[index][1] == 'elif':
    backpatch([B_false], nextQuad())
    match('RESERVED_WORD', 'elif')
    B()
    B_true = nextQuad()
    genQuad('jump', '_', '_', '_')
    B_false = nextQuad()
    match('DELIMITER', ':')
    statement()
  if index < len(code_tokens) and code_tokens[index][1] == 'else':
    backpatch([B_false], nextQuad())
    match('RESERVED_WORD', 'else')
    match('DELIMITER', ':')
    statement()
  backpatch([B_true], nextQuad())
```

Για μια εντολή if, οι τετραπλές χρησιμοποιούνται για να δημιουργηθούν οι κατάλληλες εντολές ελέγχου και άλματα, ενώ η backpatch χρησιμοποιείται για την ενημέρωση των άλματων με τις σωστές ετικέτες.

Οι βοηθητικές συναρτήσεις και οι κλάσεις σε συνδυασμό επιτρέπουν στον Yacc να παράγει έναν δομημένο ενδιάμεσο κώδικα, που διευκολύνει την περαιτέρω επεξεργασία και τη μετάφραση σε τελικό εκτελέσιμο κώδικα.

PART 3

Σε αυτό το part κάναμε αλλαγές στα δυο προηγούμενα ώστε να μπορέσουμε να παραγάγουμε σωστά τον τελικό κώδικα και να τον τρέξουμε στην εφαρμογή του risc v δυστυχώς όμως δεν τα καταφέραμε μιας και αντιμετωπίζαμε προβλήματα σε όλο το μήκος του προγράμματος πέρα από αυτό όμως καταφέραμε να φτιάξουμε ένα αρχείο με κατάληξη .asm το οποίο μεταφράζει κάποιες δομές του ενδιάμεσου κώδικα σε τελικό.

- Συνάρτηση add_symbol_to_table.

Λειτουργία: Προσθέτει ένα νέο σύμβολο στον τρέχοντα πίνακα συμβόλων. Αν το σύμβολο ήδη υπάρχει στο ίδιο επίπεδο περιοχής (scope level), εγείρεται ένα σφάλμα.

Χρήση: Χρησιμοποιείται κατά την ανάλυση των δηλώσεων (π.χ., μεταβλητών και συναρτήσεων) για να εισάγει τα σύμβολα στον πίνακα συμβόλων.

- Συνάρτηση gnlvcode

Λειτουργία: Υπολογίζει και φορτώνει τη διεύθυνση μιας μη τοπικής μεταβλητής στο καταχωρητή t0.

Χρήση: Χρησιμοποιείται όταν μια μεταβλητή βρίσκεται σε ανώτερο επίπεδο περιοχής (scope level).

- Συνάρτηση loadvr

Λειτουργία: Φορτώνει την τιμή της μεταβλητής v στον καταχωρητή reg.

Χρήση: Χρησιμοποιείται για την ανάκτηση τιμών μεταβλητών κατά την εκτέλεση εντολών.

- Συνάρτηση storerv

Λειτουργία: Αποθηκεύει την τιμή του καταχωρητή reg στη μεταβλητή v.

Χρήση: Χρησιμοποιείται για την αποθήκευση τιμών μεταβλητών κατά την εκτέλεση εντολών.

- Συνάρτηση translate_quad_to_riscv

Λειτουργία: Μεταφράζει ένα quadruple σε αντίστοιχες εντολές RISC-V.

Χρήση: Χρησιμοποιείται για τη μετάφραση του ενδιάμεσου κώδικα σε τελικό κώδικα συναρμολόγησης.

- Συνάρτηση produce

Λειτουργία: Γράφει μια εντολή στο αρχείο συναρμολόγησης.

Χρήση: Χρησιμοποιείται από τις άλλες συναρτήσεις για να προσθέσουν εντολές στον τελικό κώδικα.

Πώς χρησιμοποιούνται αυτές οι συναρτήσεις στο πρόγραμμα για την παραγωγή τελικού κώδικα:

- **Ανάλυση και Εισαγωγή Συμβόλων**:

  Κατά την ανάλυση του κώδικα, οι δηλώσεις μεταβλητών και συναρτήσεων εισάγονται στον πίνακα συμβόλων χρησιμοποιώντας τη add_symbol_to_table.

- **Μετατροπή Quadruples σε RISC-V**:

  Οι εντολές του ενδιάμεσου κώδικα (quadruples) μεταφράζονται σε RISC-V χρησιμοποιώντας τη translate_quad_to_riscv. Κάθε quadruple μεταφράζεται σε αντίστοιχες εντολές RISC-V.

- **Παραγωγή Τελικού Κώδικα**:

  Η produce χρησιμοποιείται για να γράψει τις μεταφρασμένες εντολές στο αρχείο συναρμολόγησης 5085_4967.asm.

Συνοψίζοντας, αυτές οι συναρτήσεις αποτελούν τον πυρήνα της διαδικασίας μετάφρασης από τον ενδιάμεσο κώδικα στον τελικό κώδικα RISC-V, εξασφαλίζοντας ότι οι δηλώσεις και οι εντολές μεταφράζονται σωστά και γράφονται στο αρχείο εξόδου.