The Cimple Language

Εργασία του μαθήματος των Μεταφραστών

Ακαδημαϊκό έτος 2020-2021

Περιεχόμενα:

Εισαγωγή

Η γλώσσα Cimple είναι μια μικρή εκπαιδευτική γλώσσα προγραμματισμού γραμμένη σε Python. Θυμίζει τη γλώσσα C από την οποία αντλεί ιδέες και δομές αλλά είναι αρκετά πιο μικρή τόσο στις υποστηριζόμενες δομές όσο φυσικά και σε προγραμματιστικές δυνατότητες. Η Cimple υποστηρίζει δομές όπως η while, η if-else, forcase, incase καθώς και συναρτήσεις, διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, φώλιασμα στη δήλωση συναρτήσεων και διαδικασιών και αναδρομικές κλήσεις. Η κατάληξη των προγραμμάτων Cimple είναι «.ci» και η τελική γλώσσα που θα παραχθεί θα είναι σε Assembly Mips.

Μέρη Υλοποίησης:

- Λεκτικός αναλυτής
- Συντακτικός αναλυτής
- Ενδιάμεσος κώδικας
- Πίνακας συμβολών
- Τελικός κώδικας

Η γραμματική της Cimple της συντακτικής ανάλυσης

```
#"program" is the starting symbol
program          :        program ID block .

#a block with declarations, subprograms and statements
Block            :        declarations subprograms statements

#declaration of variables, zero or more "declare" allowed
declarations     :        ( declare varlist ; )*

#a list of variables following the declaration keyword
varlist          :        ID ( , ID )*
                          | ε

#zero or more subprograms allowed
subprograms      :        ( subprogram )*

#a subprogram is a function or a procedure, followed by parameters
and block
Subprogram       :        function ID ( formalparlist ) block
                          | procedure ID ( formalparlist ) block

#list of formal parameters
formalparlist    :        formalparitem ( , formalparitem )*
                          | ε

#a formal parameter (" in ": by value , " inout " by reference)
formalparitem    :        in ID
                          | inout ID

# one or more statements
statements       :        statement ;
                          | { statement ( ; statement )* }

# one statement
statement        :        assignStat
                          | ifStat
                          | whileStat
                          | switchcaseStat
                          | forcaseStat
                          | incaseStat
                          | callStat
                          | returnStat
                          | inputStat
                          | printStat
                          | ε

#assignment statement
assignStat       :        ID := expression
```

```
#ifstatement
ifStat          :       if ( condition ) statements elsepart
elsepart        :       else statements
                        | ε

#whilestatement
whileStat       :       while ( condition ) statements


#switchstatement
switchcaseStat  :       switchcase
                        ( case ( condition ) statements )*
                        default statements

#forcasestatement
forcaseStat     :       forcase
                        ( case ( condition ) statements )*
                        default statements

#incasestatement
incaseStat      :       incase
                        ( case ( condition ) statements )*

#returnstatement
returnStat      :       return( expression )

#callstatement
callStat        :       call ID( actualparlist )

#printstatement
printStat       :       print( expression )

#inputstatement
inputStat       :       input( ID )

#listofactualparameters
actualparlist   :       actualparitem ( , actualparitem )*
                        | ε

#an actualparameter(" in ": by value , " inout " by reference)
actualparitem   :       in expression
                        | inout ID

#booleanexpression
condition       :       boolterm ( or boolterm )*

#terminbooleanexpression
boolterm        :       boolfactor ( and boolfactor )*

#factorinbooleanexpression
boolfactor      :       not [ condition ]
                        | [ condition ]
                        | expression REL_OP expression

#arithmeticexpression
expression      :       optionalSign term ( ADD_OP term )*
```

```
#terminarithmeticexpression
term            :   factor ( MUL_OP factor )*

#factorinarithmeticexpression
factor          :   INTEGER
                |   ( expression )
                |   ID idtail

#followsafunctionofprocedure(parethnesisandparameters)
idtail          :   ( actualparlist )
| ε

#sumbols"+"and"-"(areoptional)
optionalSign    :   ADD_OP
                |   ε

#lexerrules:relational,arithenticoperations,integersandids
REL_OP          :   = | <= | >= | > | < | <>

ADD_OP          :   + | -

MUL_OP          :   * | /

INTEGER         :   [0-9]+

ID              :   [a-zA-Z][a-zA-Z0-9]*
```

Μορφή προγράμματος

Κάθε πρόγραμμα ξεκινάει με τη λέξη κλειδί program. Στη συνέχεια ακολουθεί ένα αναγνωριστικό (όνομα) για το πρόγραμμα αυτό και τα τρία βασικά μπλοκ του προγράμματος: οι δηλώσεις μεταβλητών (declarations), οι συναρτήσεις και διαδικασίες (subprograms), οι οποίες μπορούν και να είναι φωλιασμένες μεταξύ τους, και οι εντολές του κυρίως προγράμματος (statements). Η δομή ενός προγράμματος Cimple φαίνεται παρακάτω. Προσέξτε την τελεία στο τέλος.

```
program id
        declarations
        subprograms
        statements
```

Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η Cimple είναι οι ακέραιοι αριθμοί. Η δήλωση γίνεται με την εντολή declare. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται

να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της declare.

Δομές της γλώσσας

Εκχώρηση

ID := expression
Χρησιμοποιείται για την ανάθεση της τιμής μιας μεταβλητής ή μιας σταθεράς, ή μιας έκφρασης
σε μία μεταβλητή.

Απόφαση if

if (condition)
    statements1
[else
    statements2]

Η εντολή απόφασης if εκτιμά εάν ισχύει η συνθήκη condition και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές statements1 που το ακολουθούν. Το else δεν αποτελεί υποχρεωτικό τμήμα
της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές statements2 που ακολουθούν το else
εκτελούνται εάν η συνθήκη condition δεν ισχύει

Επανάληψη while

while (condition)
    statements

Η εντολή επανάληψης while επαναλαμβάνει τις εντολές statements, όσο η συνθήκη condition
ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η condition το αποτέλεσμα της αποτίμησης είναι
ψευδές, τότε οι statements δεν εκτελούνται ποτέ.

Επιλογή switchcase

switchcase
    (case (condition) statements1)*
    default statements2

Η δομή switchcase ελέγχει τις condition που βρίσκονται μετά τα case. Μόλις μία από αυτές
βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες statements1 (που ακολουθούν το condition). Μετά

ο έλεγχος μεταβαίνει έξω από την switchcase. Αν, κατά το πέρασμα, καμία από τις case δεν
ισχύσει, τότε ο έλεγχος μεταβαίνει στην default και εκτελούνται οι statements2. Στη συνέχεια ο
έλεγχος μεταβαίνει έξω από την switchcase.

Επανάληψη forcase

forcase
    (case (condition) statements1)*
    default statements2

Η δομή επανάληψης forcase ελέγχει τις condition που βρίσκονται μετά τα case. Μόλις μία από
αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες statements1 (που ακολουθούν το condition). Μετά ο έλεγχος μεταβαίνει στην αρχή της forcase. Αν καμία από τις case δεν ισχύει, τότε
ο έλεγχος μεταβαίνει στη default και εκτελούνται οι statements2. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την forcase.

Επανάληψη incase

incase
    (case (condition) statements1)*

Η δομή επανάληψης incase ελέγχει τις condition που βρίσκονται μετά τα case, εξετάζοντας τες
κατά σειρά. Για κάθε μία για τις οποίες η αντίστοιχη condition ισχύει εκτελούνται οι statements
που ακολουθούν το condition. Θα εξεταστούν με τη σειρά όλες οι condition και θα εκτελεστούν
όλες οι statements των οποίων οι condition ισχύουν. Αφότου εξεταστούν όλες οι case, ο έλεγχος
μεταβαίνει έξω από τη δομή incase, εάν καμία από τις statements δεν έχει εκτελεστεί, ή μεταβαίνει στην αρχή της incase, έαν έστω και μία από τις statements έχει εκτελεστεί.

Επιστροφή τιμής συνάρτησης

    return (expression)

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης, το
οποίο είναι το αποτέλεσμα της αποτίμησης του expression.

Έξοδος δεδομένων

print (expression)

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του expression.

Είσοδος δεδομένων

input (ID)

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο. Η τιμή που θα δώσει θα
μεταφερθεί στην μεταβλητή ID.

Κλήση διαδικασίας

call functionName(actualParameters)

Καλεί μία διαδικασία.

Συναρτήσεις και διαδικασίες

Η Cimple υποστηρίζει συναρτήσεις και διαδικασίες. Για τις συναρτήσεις η σύνταξη είναι:

```
function ID (formalPars)
{
        declarations
        subprograms
        statements
}
```

ενώ για τις διαδικασίες:

```
procedure ID (formalPars)
{
        declarations
        subprograms
        statements
}
```

Η formalPars είναι η λίστα των τυπικών παραμέτρων. Οι συναρτήσεις και οι διαδικασίες μπορούν να φωλιάσουν η μία μέσα στην άλλη. Οι κανόνες εμβέλειας ακολουθούν τους κανόνες της
PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την return.

Η κλήση μιας συνάρτησης, γίνεται μέσα από τις αριθμητικές παραστάσεις σαν τελούμενο. π.χ.

D = a + f(in x)

όπου f η συνάρτηση και x παράμετρος που περνάει με τιμή.
Η κλήση μιας διαδικασίας γίνεται με την call. π.χ.

call f(inout x)

όπου f η διαδικασία και x παράμετρος που περνάει με αναφορά.

Μετάδοση παραμέτρων

Η Cimple υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

• με τιμή: Δηλώνεται με τη λεκτική μονάδα in. Αλλαγές στην τιμή της δεν επιστρέφονται
στο πρόγραμμα που κάλεσε τη συνάρτηση
• με αναφορά: Δηλώνεται με τη λεκτική μονάδα inout. Κάθε αλλαγή στη τιμή της μεταφέρεται αμέσως στο πρόγραμμα που κάλεσε τη συνάρτηση
Στην κλήση μίας συνάρτησης οι πραγματικοί παράμετροι συντάσσονται μετά από τις λέξεις κλειδιά in και inout, ανάλογα με το αν περνούν με τιμή ή αναφορά.

❖ Λεκτικός Αναλυτής (1η φάση)

Ο λεκτικός αναλυτής διαβάζει χαρακτήρα-χαρακτήρα το πρόγραμμα έτσι ώστε να μπορεί να αναγνωρίζει αν το πρόγραμμα μας είναι σε γλώσσα CImple. Διαβάζοντας κάθε χαρακτήρα ελέγχει αν είναι μεταβλητή, σταθερά ή κάποιος αριθμητικός τελεστής. Συγκεκριμένα, ο λεκτικός αναλυτής αναγνωρίζει τα παρακάτω

Λεκτικά Tokens:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,…,Z και a,…,z),
• τα αριθμητικά ψηφία (0,…,9),
• τα σύμβολα των αριθμητικών πράξεων (+, -, *, /),
• τους τελεστές συσχέτισης (<, >, =, <=, >=, <>)
• το σύμβολο ανάθεσης (:=)
• τους διαχωριστές (;, “,”, :)
• τα σύμβολα ομαδοποίησης ([,], (,) , { , })
• του τερματισμού του προγράμματος (.)
• και διαχωρισμού σχολίων (#)

Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

• Πολλαπλασιαστικοί: *, /

• Προσθετικοί: +, -

• Σχεσιακοί: =, <, >, <>, <=, >=

• Λογικοί: not

• Λογική σύζευξη: and

• Λογική διάζευξη: or

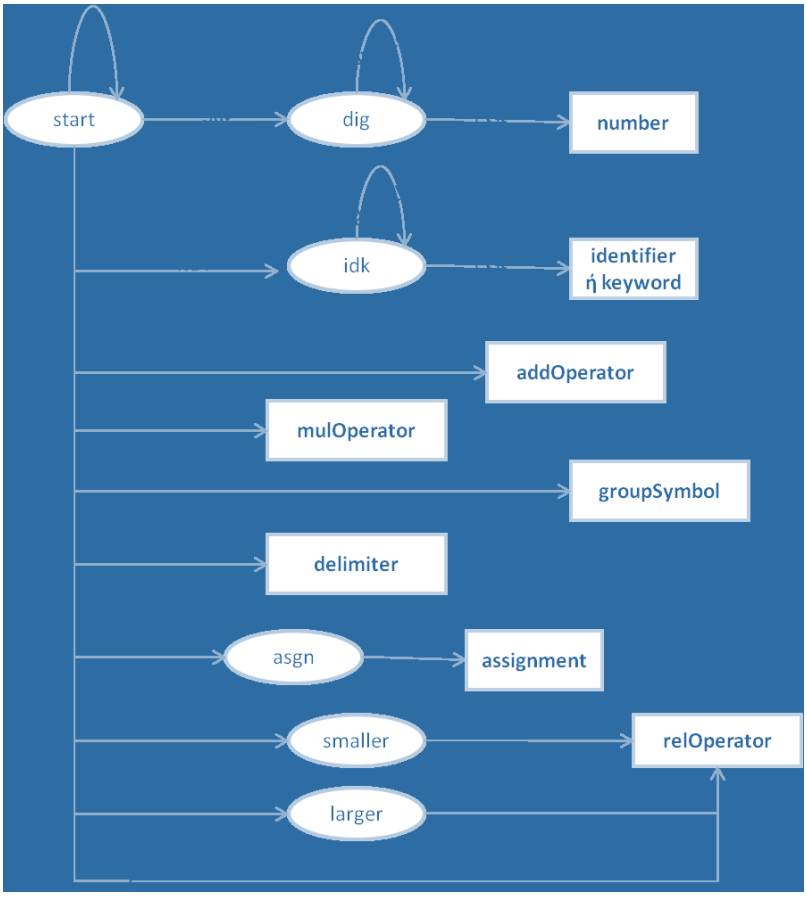
Δεσμευμένες λέξεις

program	case
declare	default
if	not
else	and
while	or
switchcase	function
forcase	procedure
incase	call
return	in
inout	input
print	

Η διαδικασία είναι η εξής: ο λεκτικός αναλυτής είναι μία συνάρτηση με όνομα **lex()** και, κάθε φορά που καλείται , εξάγει μια λέξη την οποία αναγνωρίζει και αρχειοθετεί σε μια προσωρινή λίστα ώστε να ελέγξουμε αν υπακούει στη γραμματική της Cimple. Αν υπακούει και είναι έγκυρη τότε την κρατάμε και επιστρέφουμε το αποτέλεσμα ειδάλλως εκτυπώνουμε σχετικό μήνυμα σφάλματος στο σημείο της ανάλυσης που εντοπίστηκε.

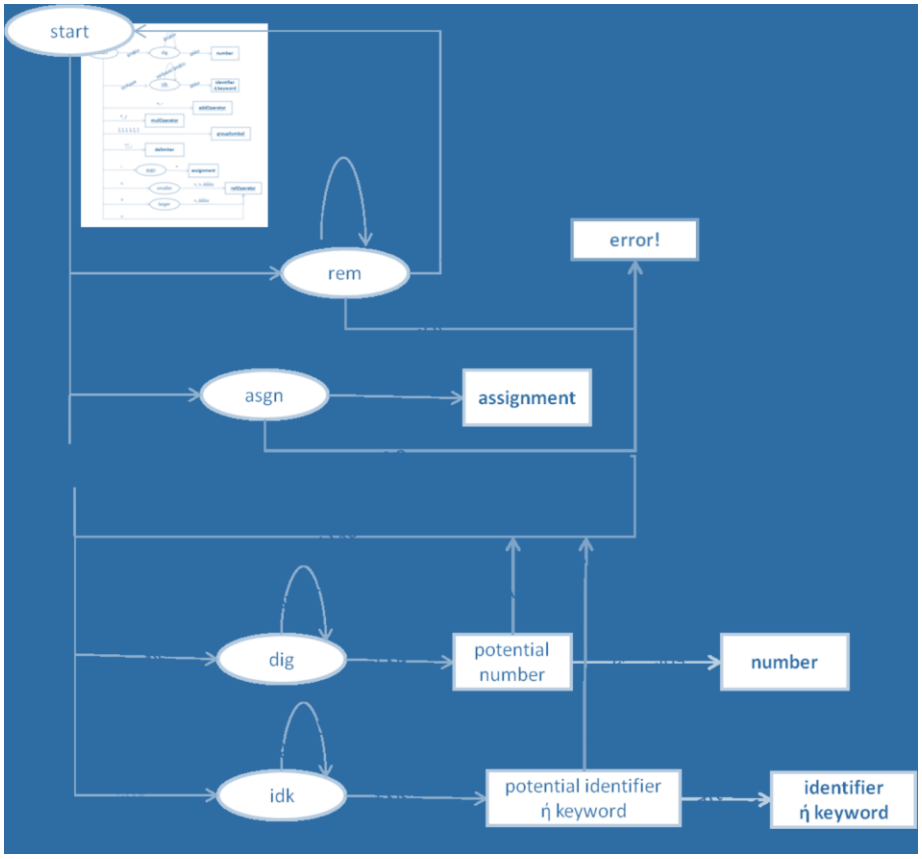
Η συνάρτηση lex() κάθε φορά που καλείται επιστρέφει ένα αντικείμενο τύπου Token, το οποίο ορίζουμε στην αρχή του προγράμματος. Το πεδία της κλάσης αυτής είναι:

1. TokenType : ο τύπος του αντικειμένου (πχ. String)
2. String : ο συγκεκριμένος χαρακτήρας που διαβάστηκε τελευταία
3. lineNo : Η γραμμή στην οποία βρίσκεται ο χαρακτήρας.



Εικόνα : Πεπερασμένο αυτόματο λεκτικών μονάδων

Τέλος, ο λεκτικός αναλυτής είναι να εντοπίζει λάθη κατά το διάβασμα ενός προγράμματος. Ελέγχει την ορθότητα του προγράμματος όπως: των αναγνωριστικών μεταβλητών που δίνει ο χρήστης, το μέγεθος των ακέραιων αριθμός, το κλείσιμο των σχολίων και συμβολών όπως το assign symbol (:=).



❖ Συντακτικός Αναλυτής (1η φάση)

Η συνάρτηση lex() καλείται από τις συναρτήσεις που κάνουν την δουλειά του λεκτικού αναλυτή. Με την σειρά του ο συντακτικός αναλυτής ελέγχει αν το token που επέστρεψε ο lex() είναι μια έγκυρη εντολή της Cimple ή αν βρίσκεται στην σωστή θέση. Κάθε μια από τις συναρτήσεις ελέγχει ένα συγκεκριμένο κομμάτι και αν το πρόβλημα υπακούει τους κανόνες τις γλώσσας. Αν υπάρχει συντακτικό λάθος, ο συντακτικός αναλυτής τυπώνει ένα μήνυμα σφάλματος μαζί με την γραμμή στην οποία βρίσκεται.

Οι συναρτήσεις που έχουμε υλοποιήσει πραγματοποιούν έλεγχο προκειμένου να διαπιστώσουν αν ακολουθείται σωστά η γραμματική και η σύνταξη της γλώσσας Cimple. Αν συμβαίνει το προηγούμενο τότε εκτελούνται οι κατάλληλες ενέργειες που απαιτούνται, διαφορετικά τυπώνονται αντίστοιχα μηνύματα λάθους και τερματίζεται η διαδικασία μεταγλώττισης.

❖ Ενδιάμεσος κώδικας (2η φάση)

Η δεύτερη φάση περιλαμβάνει βοηθητικές συναρτήσεις οι οποίες παράγουν ένα σύνολο από τετράδες, δηλαδή τον ενδιάμεσο κώδικα. Οι τετράδες αυτές αποτελούνται από έναν τελεστή και τρία τελούμενα.

Οι τετράδες είναι αριθμημένες. Κάθε τετράδα έχει μπροστά της έναν μοναδικό αριθμό που τη χαρακτηρίζει. Μόλις τελειώσει η εκτέλεση μίας τετράδας εκτελείται η τετράδα που έχει τον αμέσως μεγαλύτερο αριθμό, εκτός εάν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό.

Τελεστής Αριθμητικής πράξης

τετράδες της μορφή : op x , y , z
ο τελεστής op μπορεί να είναι: + , - , * , /
τα τελούμενα x, y μπορεί να είναι: είτε ονόματα μεταβλητών είτε αριθμητικές σταθερές.
το τελούμενο z μπορεί να είναι όνομα μεταβλητής.
Εφαρμόζεται ο τελεστής op στα τελούμενα x , y και το αποτέλεσμα τοποθετείται στο τελούμενο z.

Τελεστής Εκχώρησης

τετράδες της μορφής : := , x , _ , z
το τελούμενο x μπορεί να είναι όνομα μεταβλητής ή αριθμητική σταθερά.
το τελούμενο z μπορεί να είναι όνομα μεταβλητής.
Η τιμή του x εκχωρείται στο τελούμενο z.

Τελεστής άλματος χωρίς συνθήκη

τετράδες της μορφής : jump , _ , _ , z.

Μεταπήδηση χωρίς όρους στη θέση z.

Τελεστής άλματος με συνθήκη

τετράδες της μορφής: relop , x , y , z
ο τελεστής relop μπορεί να είναι ενας από τους τελεστές: =, >, <, <>, >=, <=
Μεταπήδηση στη θέση z αν ισχύει (x relop y)

Αρχή και τέλος ενότητας

τετράδες για αρχή της μορφής: beginBlock , name , _ , _
Αρχή υποπρογράμματος ή προγράμματος με το όνομα name
τετράδες για τέλος της μορφής: endBlock , name , _ , _
Τέλος υποπρογράμματος ή προγράμματος με το όνομα name
τετράδες για τερματισμό της μορφής: halt , _ , _ , _
Τερματισμός προγράμματος

Συναρτήσεις – Διαδικασίες

τετράδες της μορφής : par , x , m , _
όπου x παράμετρος συνάρτησης
και m ο τρόπος μετάδοσης που μπορεί να είναι:
CV : μετάδοση με τιμή
REF: μετάδοση με αναφορά
RET: επιστροφή τιμής συνάρτησης
τετράδες της μορφής : call , name , _ , _
κλήση συνάρτησης name
τετράδες της μορφής: ret , x , _ , _
επιστροφή τιμής συνάρτησης

Βοηθητικές Υπορουτινες:

1. **nextquad():** επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
2. **genquad(op,x,y,z)**: δημιουργεί την επόμενη τετράδα.
3. **newtemp():** δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή, οι προσωρινες μεταβλητες είναι της μορφης T_1,T_2,T_3..
4. **emptylist():** δημιουργεί μία κενή λίστα ετικετών τετράδων.
5. **makelist(label):** δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x.
6. **merge(list1 , list2):** δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών 1 και 2

7. **backpatch(labelList, z):** η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η backpatch επισκέπτεται μία-μία τις τετράδες αυτες και τις συμπληρώνει με την ετικέτα z.

Υλοποίηση Συναρτήσεων Ενδιάμεσου Κώδικα Αρχή και τέλος Block

*Οι γραμμές αναφέρονται στο σημείο που καλούνται στο αρχείο phase2.py που παραδώσαμε στην δεύτερη φάση.

(LINES 232)

<program> ::= program name

<program_block> (name)

(LINES 252)

<program_block> (name) ::= <declarations>

<subprograms>

genquad("begin_block",name,"_","_")

< block>

genquad("halt","_","_","_")

genquad("end_block",name,"_","_")

(LINES 346 & 379)

<subprograms> ::= function id <formalpars>

{ genquad("begin_block",id,"_","_")

<block>

genquad("end_block",id,"_","_")

}

Αριθμητικές Παραστάσεις

E -> T1(+ T2{P1})* {P2}

{P1}: w = newTemp()

genquad("+",T1.place,T2.place,w)

T1.place=w

{P2}: E.place=T1.place

- Νέα προσωρινή μεταβλητή που θα κρατήσει το μέχρι στιγμής αποτέλεσμα
- Παραγωγή τετράδας που προσθέτει το μέχρι στιγμής αποτέλεσμα στο νέο T2
- Το μέχρι στιγμής αποτέλεσμα τοποθετείται στην T1 ώστε να χρησιμοποιηθεί αν υπάρξει επόμενο T2
- Όταν δεν υπάρχει άλλο T2 το αποτέλεσμα είναι στο T1

Για παράδειγμα:

```
procedure E (E.place)
        begin
                T ( T1.place )
                while token=plustk do begin
                        lex();
                        T (T2.place)
                        w:=newTemp()
                        genquad( "+", T1.place, T2.place, w)
                        T1.place :=w
        end
        E.place := T1.place
end
```

Λογικές Παραστάσεις - OR

B -> Q1{P1} (or {P2} Q2 {P3})*

{P1}: B.true = Q1.true

B.false = Q1.false

{P2}: backpatch(B.false, nextquad())

{P3}: B.true = merge(B.true, Q2.true)

B.false = Q2.false

- Μεταφορά των τετράδων από τη λίστα Q1 στη λίστα B
- Συμπλήρωση όσων τετράδων μπορούν να συμπληρωθούν μέσα στον κανόνα
- Συσσώρευση στη λίστα true των τετράδων που δεν μπορούν να συμπληρωθούν και αντιστοιχούν σε αληθή αποτίμηση λογικής παράστασης
- Η λίστα false περιέχει την τετράδα η οποία αντιστοιχεί σε στη μη αληθή αποτίμηση της λογικής παράστασης

Λογικές Παραστάσεις - AND

Q -> R1{P1} (and {P2} R2 {P3})*

{P1}: Q.true = R1.true

Q.false = R .false

Q.false = R1.false

{P2}: backpatch(Q.true, nextquad())

{P3}: Q.false = merge(Q.false, R2.false)

Q.true = R2.true

- Μεταφορά των τετράδων από τη λίστα R1 στη λίστα Q
- Συμπλήρωση όσων τετράδων μπορούν να συμπληρωθούν μέσα στον κανόνα
- Συσσώρευση στη λίστα false των τετράδων που δεν μπορούν να συμπληρωθούν και αντιστοιχούν σε μη αληθή αποτίμηση λογικής παράστασης
- Η λίστα true περιέχει την τετράδα η οποία αντιστοιχεί σε στην αληθή αποτίμηση της λογικής παράστασης

Λογικές Παραστάσεις - not

R -> not (B) {P1}

{P1}: R.true=B.false

R.false=B.true

- Αντιστροφή και μεταφορά τετράδων από τη λίστα B στη λίστα R

Λογικές Παραστάσεις - relop

R -> E1 relop E2{P1}

{P1}: R.true=makelist(nextquad())

genQuad(relop, E1.place, E2.place, “_”)

R.false=makelist(nextquad())

genQuad(“jump” , “_” , “_” , “_”)

- Δημιουργία μη συμπληρωμένης τετράδας και εισαγωγή στη λίστα μη συμπληρωμένων τετράδων για την αληθή αποτίμηση της relop
- Δημιουργία μη συμπληρωμένης τετράδας και εισαγωγή στη λίστα μη συμπληρωμένων τετράδων για τη μη αληθή αποτίμηση της relop

Κλήση Υποπρογραμμάτων

Κλήση διαδικασίας:

call assign_v (in a, inout b)

par, a, CV, _

par, b, REF, _

call, assign_v , _, _

Κλήση συνάρτησης:

error = assign_v (in a, inout b)

par, a, CV, _

par, b, REF, _

w = newTemp()

par, w, RET, _

call, assign_v , _, _

Εντολή return

S -> return (E) {P1}

{P1}: genquad(“retv”,E.place,”_”,”_”)

Εκχώρηση

S -> id := E {P1};

{P1} : genQuad(“:=“,E.place,”_”,id)

Δομή while

S -> while {P1} B do {P2} S1{P3}

{P1}: Bquad:=nextquad()

{P2}: backpatch(B.true,nextquad())

{P3}: genquad(“jump”,”_”,”_”,Bquad)

backpatch(B.false,nextquad())

- Συμπλήρωση των τετράδων που έχουν μείνει ασυμπλήρωτες και γνωρίζουμε τώρα ότι πρέπει να συμπληρωθούν με την επόμενη τετράδα, το true πάνω στην S και το false έξω από τη δομή
- Μετάβαση στην αρχή της συνθήκης ώστε να ξαναγίνει έλεγχος

Δομή Repeat…Until

S -> repeat {P1} S1 until (cond) {P2}

{P1}: sQuad:=nextquad()

{P2}: backpatch(cond.False,sQuad)

backpatch(cond.True,nextquad())

- Οι τετράδες αυτές πρέπει να μεταβούν στην αρχή της συνθήκης για να επανελεγχθεί
- Συμπλήρωση των τετράδων που έχουν μείνει ασυμπλήρωτες και γνωρίζουμε τώρα ότι πρέπει να συμπληρωθούν με την επόμενη τετράδα, δηλαδή έξω από τη δομή

Δομή if

S -> if B then {P1} S1{P2} TAIL {P3}

{P1}: backpatch(B.true,nextquad())

{P2}: ifList=makelist makelist(nextquad nextquad())

genquad(“jump”,”_”,”_”,”_”)

backpatch(B.false,nextquad())

{P3}: backpatch(ifList,nextquad())

TAIL -> else S2| TAIL -> ε

- Συμπλήρωση των τετράδων που έχουν μείνει ασυμπλήρωτες και και γνωρίζουμε τώρα ότι πρέπει να συμπληρωθούν με την επόμενη τετράδα, στο if και else αντίστοιχα
- Εξασφαλίζουμε ότι εάν εκτελεστούν οι εντολές του if δε θα εκτελεστούν στη συνέχεια οι εντολές του else

Δομή switch

S -> switch {P1}

((cond): {P2} S1 break {P3})*

default: S2{P4}

{P1} : exitlist = emptylist()

{P2} : backpatch(cond.true,nextquad())

{P3} : e = makelist(nextquad())

genquad('jump', '_', '_', '_')

mergelist(exitlist,e)

backpatch(cond.false,nextquad())

{P4} : backpatch(exitlist,nextquad())

Είσοδος - Έξοδος

S -> input (id) {P1}

{P1}: genquad(“inp”,id.place,”_”,”_”)

S -> print (E) {P2}

{P2}: genquad(“out”,E.place,”_”,”_”)

❖ Πίνακας Συμβόλων (3η φάση)

Βοηθητικές Συναρτήσεις

record_arguments(parMode ,type):

Αυτή η συνάρτηση σκοπεύει στο να καταγράφει της τυπικές παραμέτρους των διαδικασιών και των συναρτήσεων μέσα σε μία λίστα η οποία αρχικοποιείται κάθε φορά που ο λεκτικός αναλυτής επιστρέφει function ή procedure. Προστίθεται στο Scope της συνάρτησης/διαδικασίας όταν δεν έχει άλλες τυπικές παραμέτρους στην δήλωσή της. Η parMode δηλώνει τον τρόπο περάσματος και το type τον τύπο της.

recordEntity(entity_name, type, entity_list):

Το πεδίο entity_name δηλώνει το όνομα της μεταβλητής. Το πεδίο type δηλώνει τον τύπο της μεταβλητής. Το πεδίο type μπορεί να πάρει τις τιμές id,subprogram,σταθερά, την προσωρινή μεταβλητη temp. Το πεδίο entity_list μπορεί να είναι το offset(η απόσταση από την αρχή του εγγραφήματος δραστηριοποίσης), starQuad(ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης), λίστα παραμέτρων, framelength(μήκος εγγραφήματος δραστηριοποίησης),value(τιμή της σταθεράς) και τέλος το parMode(τρόπος περάσματος).

Η συνάρτηση αυτή χρησιμοποιείται στο Scope του κάθε block είτε αυτό είναι το κυρίως πρόγραμμα είτε είναι μια διαδικασία είτε είναι μία συνάρτηση:

- όταν συναντάμε δήλωση μεταβλητής
- όταν δημιουργείται νέα προσωρινή μεταβλητή
- όταν συναντάμε δήλωση νέας συνάρτησης
- όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

Κάθε φορά που ξεκινάει ένα καινούργιο block αυξάνεται το βάθος φωλιάσματος και όταν τελειώνει ένα block τότε μειώνεται. Όταν τελειώσει το block τότε παράγεται τελικός κώδικας και διαγράφεται το Scope της συγκεκριμένης συνάρτησης/διαδικασίας.

❖ Τελικός Κώδικας (4η φάση)

Το συγκεκριμένο κομμάτι κώδικα παράγει ένα αρχείο σε γλώσσα Assembly με κατάληξη «.asm» έτσι ώστε ο αρχικός κώδικας της γλώσσας Cimplena μπορεί να μεταφραστεί και να εκτελεστεί σε γλώσσα μηχανής με τη βοήθεια του Mips. Για κάθε εντολή του ενδιάμεσου κώδικα, παράγουμε τις αντίστοιχες εντολές του τελικού κώδικα γιατί σε αυτή την φάση θα απεικονίσουμε τις μεταβλητές στην μνήμη (στοίβα του προγράμματος), θα περάσουμε τις παραμέτρους και θα γίνει η κλήση των διαδικασιών/συναρτήσεων.

Βοηθητικές Συναρτήσεις

gnlvcode(v):

Το όρισμα v αντικαθίσταται από τις παραμέτρους του αρχικού προγράμματος

Η συνάρτηση αυτή μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής.. Από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει.

```
lw $t0,-4($sp)          στοίβα του γονέα
```

όσες φορές χρειαστεί:

```
lw $t0,-4($t0)          στοίβα του προγόνου που έχει τη μεταβλητή
addi $t0,$t0,-offset    διεύθυνση της μη τοπικής μεταβλητής
```

loadvr(v , r):

Τα ορίσματα v είναι το όνομα της μεταβλητής και το r είναι ο καταχωρητής.

Μεταφορά δεδομένων στον καταχωρητή $r. Η μεταφορά μπορεί να γίνει από τη μνήμη (στοίβα) Διακρίνουμε περιπτώσεις:

1. εάν η v είναι σταθερά.(const).

2. εάν η v είναι καθολική μεταβλητή (global) . Η μεταβλητή έχει βάθος φωλιάσματος ίσο με μηδέν.

3. εάν η v είναι τοπική μεταβλητή ή τυπική παράμετρος με τιμή(in) ή είναι προσωρινή μεταβλητή(T_i) και έχει βάθος φωλιάσματος ίσο με το τρέχων.

4. εάν η v είναι τυπική παράμετρος με αναφορά (inout) και έχει βάθος φωλιάσματος ίσο με το τρέχων.

5. εάν η v είναι τυπική παράμετρος με αναφορά (inout) και έχει βάθος φωλιάσματος μικρότερο από το τρέχων.

6. εάν η v είναι τοπική μεταβλητή ή τυπική παράμετρος με τιμή(in) και έχει βάθος φωλιάσματος μικρότερο από το τρέχων

storerv(r,v):

Μεταφορά δεδομένων από τον καταχωρητή $r στη μνήμη (μεταβλητή v)

Διακρίνουμε περιπτώσεις:

1. εάν η v είναι καθολική μεταβλητή (global) . Η μεταβλητή έχει βάθος φωλιάσματος ίσο με μηδέν και υπάρχει η τιμή της στην στοίβα του κυρίως προγράμματος.

2. εάν η v είναι τοπική μεταβλητή ή τυπική παράμετρος με τιμή(in) ή είναι προσωρινή μεταβλητή(T_i) και έχει βάθος φωλιάσματος ίσο με το τρέχων. Η τιμή της μεταβλητής υπάρχει στην στοίβα μας.

3. εάν η v είναι τυπική παράμετρος με αναφορά (inout) και έχει βάθος φωλιάσματος ίσο με το τρέχων. Η διεύθυνση της υπάρχει στην στοίβα μας.

4. εάν η v είναι τυπική παράμετρος με αναφορά (inout) και έχει βάθος φωλιάσματος μικρότερο από το τρέχων. Η διεύθυνσή της υπάρχει σε στοίβα προγόνου μας.

5. εάν η v είναι τοπική μεταβλητή ή τυπική παράμετρος με τιμή(in) και έχει βάθος φωλιάσματος μικρότερο από το τρέχων. Η τιμή της υπάρχει σε στοίβα προγόνου μας.

parameterOffset(i,quads_number):

Η συνάρτηση αυτή ελέγχει αν έχουμε την πρώτη τετράδα ‘par’ που πρόκειται να μεταφραστεί σε τελικό κώδικα. Αν ναι, τότε ψάχνει να βρει προς τα κάτω την τετράδα ‘call’ έτσι ώστε να πάρει το framelentgh της καλούσας συνάρτησης/διαδικασίας.

(όπου framelentgh = offset + 4).

locateVariable(v):

Η συνάρτηση αυτή αναζητά από κάθε nesting level μέχρι το επίπεδο του κυρίως προγράμματος να βρει το entity έτσι ώστε να επιστρέψει το entity level της και σε ποια θέση βρίσκεται.

(θέση*4 = offset της μεταβλητής).

Ενδιάμεσος κώδικάς	Κώδικας MIPS
jump, “_”, “_”, label	b label
relop(?),x,y,z	loadvr(x,$t1) loadvr(y, $t2) branch(?),$t1,$t2,z
:=, x, “_”, z	loadvr(x, $t1) storerv($t1, z)
op x,y,z	loadvr(x, $t1) loadvr(y, $t2) op $t1,$t1,$t2 storerv($t1,z)
out “_”, “_”, x	li $v0,1 loadvr(x,$a0) syscall
in “_”, “_”, x	li $v0,5 syscall storerv($v0,x)

retv “_”, “_”, x	loadvr(x, $t1)
	lw $t0,-8($sp)
	sw $t1,($t0)

par,x,CV, _	loadvr(x, $t0) sw $t0, -(12+4i)($fp)
par,x,REF, _ • **αναφορά,** ίδιο βάθος	addi $t0,$sp,-offset sw $t0,-(12+4i)($fp)
par,x,REF, _ • **τιμή,** ίδιο βάθος	lw $t0,-offset($sp) sw $t0,-(12+4i)($fp)

par,x,REF, _ • **διαφορετικό βάθος, τιμή**	gnlvcode(x) sw $t0,-(12+4i)($fp)

par,x,REF, _
• **διαφορετικό βάθος, αναφορά**
	gnlvcode(x)
	lw $t0,($t0)
	sw $t0,-(12+4i)($fp)

par,x,RET, _	addi $t0,$sp,-offset sw $t0,-8($fp)
call, _, _, f • **ίδιο βάθος φωλιάσματος, ίδιος γονέας**	lw $t0,-4($sp) sw $t0,-4($fp)
call, _, _, f • **διαφορετικό βάθος κληθείσας είναι γονέας**	sw $sp,-4($fp)

Σημασιολογική ανάλυση

Η σημασιολογική ανάλυση στον κώδικα μας γίνεται με την βοήθεια της συνάρτησης semantic_analysis(). Συγκεκριμένα, ελέγχεται αν υπάρχει ένα τουλάχιστον return token μέσα σε κάθε block συνάρτησης και πως δεν υπάρχει κάποιο return έξω από συνάρτηση, διαδικασία η κυρίως πρόγραμμα.

Παραδείγματα

1ο παράδειγμα - Ενδιάμεσος κώδικας

```
program factorial
#declarations#
declare x;
declare i,fact;
#main#
{
	input(x);
	fact := 1;
	i := 1;
	while (i<=x)
	{
		fact := fact*i;
		i := i+1;
	};
	print(fact);
}.
```

```
0:['begin_block', 'factorial', '_', '_']
1:['input', 'x', '_', '_']
2:[':=', '1', '_', 'fact']
3:[':=', '1', '_', 'i']
4:['<=', 'i', 'x', 6]
5:['jump', '_', '_', 11]
6:['*', 'fact', 'i', 'T_0']
7:[':=', 'T_0', '_', 'fact']
8:['+', 'i', '1', 'T_1']
9:[':=', 'T_1', '_', 'i']
10:['jump', '_', '_', '4']
11:['print', 'fact', '_', '_']
12:['halt', '_', '_', '_']
13:['end_block', 'factorial', '_', '_']
```

2ο παράδειγμα - Ενδιάμεσος κώδικας

```
program fibonacci
declare x;
function fibonacci(in x)
{
	return (fibonacci(in x-1)+fibonacci(in x-2));
}
#main#
{
	input(x);
	print(fibonacci(in x));
}.
```

```
0:['begin_block', 'fibonacci', '_', '_']
1:['-', 'x', '1', 'T_0']
2:['par', 'T_0', 'CV', '_']
3:['call', 'fibonacci', '_', '_']
4:['-', 'x', '2', 'T_1']
5:['par', 'T_1', 'CV', '_']
6:['call', 'fibonacci', '_', '_']
7:['+', 'fibonacci', 'fibonacci', 'T_2']
8:['retv', 'T_2', '_', '_']
9:['halt', '_', '_', '_']
10:['end_block', 'fibonacci', '_', '_']
11:['begin_block', 'fibonacci', '_', '_']
12:['input', 'x', '_', '_']
13:['par', 'x', 'CV', '_']
14:['call', 'fibonacci', '_', '_']
15:['print', 'fibonacci', '_', '_']
16:['halt', '_', '_', '_']
17:['end_block', 'fibonacci', '_', '_']
```

3ο παράδειγμα – Ενδιάμεσος κώδικας

```
program countDigits
declare x, count;
#main#
{
	input(x);
	count := 0;
	while (x>0)
	{
		x := x/10;
		count := count+1;
	};
```

```
0:['begin_block', 'countDigits', '_', '_']
1:['input', 'x', '_', '_']
2:[':=', '0', '_', 'count']
3:['>', 'x', '0', 5]
4:['jump', '_', '_', 10]
5:['/', 'x', '10', 'T_0']
6:[':=', 'T_0', '_', 'x']
7:['+', 'count', '1', 'T_1']
8:[':=', 'T_1', '_', 'count']
9:['jump', '_', '_', '3']
10:['print', 'count', '_', '_']
11:['halt', '_', '_', '_']
12:['end_block', 'countDigits', '_', '_']
```

```
        print(count);
}.
```

- 4ο παράδειγμα – Λεκτικός αναλυτής

```
program absolutevalue

   declare x;

   function absvalue(in x)
   {
      forcase
         case (x >= 0) x := x;
         case (x < 0) x := - x;
         default x := 0;
       ;
   };
   # m a i n #
   {
      input(x);
      print(absvalue(in x));
   }.
```

Τα λεκτικά tokens / αποτελέσματα του λεκτικού αναλυτή:

Σειρά > από αριστερά προς δεξιά

Program	absolutevalue	declare	x	;	function	absvalue	(	in	x
)	{	forcase	case	(	x	>=	0	)	x
:=	x	;	case	(	x	<	0	)	x
:=	-	x	;	default	x	:=	0	;	;
}	;	{	input	(	x	)			

Τέλος αναφοράς