# ΜΕΤΑΦΡΑΣΤΕΣ

# Report

Άγγελος Κατσαλήρος, Α.Μ. 2997

Δημήτριος Μπακάλης, Α.Μ.3033

29/05/19

# *1η Φάση ( LEXER – PARSER):*

-Έχουμε την κλάση Lexer η οποία υλοποιεί τον λεκτικό αναλυτή. Η λογική αυτής της κλάσης είναι ότι ελέγχει ένα αρχείο (test.stl) και μέσω τον μεθόδων της δημιουργεί τα tokens, με βάση την εκφώνηση της άσκησης. Αυτά τα tokens αποθηκεύονται σε ένα global πίνακα tokens. Επίσης γίνεται έλεγχος για το αν οι αριθμοί είναι μεταξύ των -32767 και 32767. Τέλος έχουμε και μερικές global μεταβλητές.

Οι μέθοδοι την κλάσης Lexer είναι :

- __init__
- tokenize

Η init φορτώνει το αρχειο ( test.stl)

Η tokenize διαβάζει γραμμή-γραμμή το αρχείο και παράγει τα tokens. Μόλις τελειώσει η tokenize έχουμε έναν πίνακα tokens, με όλα τα token του αρχείου.

Οι global μεταβλητές είναι :

- tokens : πίνακας των token
- counter : μετρητής για τα strings του test.stl
- string : πίνακας από την κάθε γραμμή του αρχείου, χωρίς κενά και tabs μεταξύ των strings.
- special : δισδιάστατος πίνακας που περιέχει όλες τις εντολές [['PROGRAM_DEC',"program"],['ENDPROGRAM_DEC',"endprogram"],['DECLARATIONS',"declare"],['IF_DEC',"if"],['THEN_DEC',"then"],['ELSE_DEC',"else"],['ENDIF_DEC',"endif"],['DOWHILE_DEC',"dowhile"],['WHILE_DEC',"while"],['ENDWHILE',"endwhile"],['LOOP_DEC',"loop"],['ENDLOOP_DEC',"endloop"],['EXIT_DEC',"exit"],['FORCASE_DEC',"forcase"],['ENDFORCASE_DEC',"endforcase"],['INCASE_DEC',"incase"],['ENDINCASE_DEC',"endincase"],['WHEN_DEC',"when"],['ENDWHEN_DEC',"endwhen"],['DEFAULT_DEC',"default"],['ENDDEFAULT_DEC',"enddefault"],['FUNCTION_DEC',"function"],['ENDFUNCTION_DEC',"endfunction"],['RETURN_DEC',"return"],['IN_DEC',"in"],['INOUT_DEC',"inout"],['INANDOUT_DEC',"inandout"],['AND_DEC',"and"],['OR_DEC',"or"],['NOT_DEC',"not"],['INPUT_DEC'

,"input"],['PRINT_DEC',"print"],['LOOP_DEC',"loop"],['ENDDOWHILE_DEC',"enddowhile"]]

-Για τον συντακτικό αναλυτή έχουμε κάποιες μεθόδους που συνεργάζονται για να αναγνωρίζεται το συντακτικό της Starlet. Η ονομασία και η λειτουργία των διάφορων μεθόδων μας δίνονται από την εκφώνηση της άσκησης. Η λογική είναι ότι κάνουμε προσπέλαση τον πίνακα tokens και, ανάλογα τα tokens με την σειρά που μπήκαν και τις αλλαγές γραμμής, κάνουμε τους ελέγχους που ζητάει το συντακτικό της Starlet. Τέλος έχουμε και μερικές global μεταβλητές.

<u>Οι μέθοδοι είναι οι εξής :</u>

- parse : ξεκινάει ελέγχοντας αν το πρόγραμμα ξεκινάει σωστά και αν γίνεται αυτό συνεχίζει με τον υπόλοιπο κώδικα μέσω της block(από την εκφώνηση)
- lines : επιστρέφει την γραμμή στην οποία βρίσκεται το token του ορίσματός της. (δικιά μας μέθοδος)
- endif_stat : ελέγχει αν υπάρχει σκέτο το “endif” . (δικιά μας μέθοδος)
- endfunction_stat : ελέγχει αν υπάρχει σκέτο το “endfunction”. (δικιά μας μέθοδος)
- endloop_stat : ελέγχει αν υπάρχει σκέτο το “endloop”. (δικιά μας μέθοδος)

<u>Οι global μεταβλητές είναι :</u>

- token_index : μετρητής ο οποίος δείχνει σε ποιά θέση στον πίνακα tokens βρισκόμαστε.
- last_token : πίνακας ο οποίος περιέχει, σε κάθε θέση του, το τελευταίο token (σε αριθμό) της κάθε γραμμής του test.stl
- flag_if, flag_while, flag_dowhile, flag_loop, flag_forcase, flag_incase, flag_funciton : ανάλογα την τιμή (0 ή 1) μας δείχνει αν έχουμε εμφωλευμενες εντολές.
- function_name : κρατάει τα ονόματα των συναρτήσεων (για τον εδνιάμεσο κώδικα)

# *2η Φάση (SymbolTable-Ενδιάμεσος Κώδικας)*

-Symbol Table:

Η λογική του symbol table είναι ότι, έχει πληροφορίες για κάθε ένα string του αρχείου test.stl.

<u>Παρακάτω αναφέρουμε την διαδικασία με την οποία υλοποιήθηκε το symbol table :</u>

1.Έχουμε φτιάξει έναν global πίνακα scopes_array ο οποίος κρατάει όλα τα scopes του κώδικα.
2.Τα scopes δημιουργούνται όταν μπαίνουμε σε μια συνάρτηση και κρατάνε πληροφορίες όπως τις παραμέτρους (in,inout,inandout) ,τις μεταβλητές των παραμέτρων, το όνομα της συνάρτησης και λοιπά.
3.Όταν τελειώνει η μετάφραση της συνάρτησης, διαγράφουμε το scope από τον scopes_array με pop.
4.Έχουμε φτιάξει τη κλάση Scope η οποία έχει ως attributes-χαρακτηριστικά:

- nested-level=0 που δείχνει πόσο φωλιασμένο είναι το scope (φωλιασμένο σημαίνει ότι μπορεί να βρίσκεται μέσα σε ένα άλλο scope ,δηλαδή μέσα σε μια άλλη συνάρτηση )
- ---enclosing scope που δειχνει στο scope που βρίσκεται από πάνω του
- ---entities που είναι πίνακας που περιέχει όλα τα entities(παρακάτω θα δείξουμε ποια είναι τα entities )
- ---offset που δείχνει την απόσταση από την κορυφή της στοίβας, αυξάνεται κάθε φορά που βρισκω μεταβλητές και ενημερώνει το framelength της συνάρτησης .

5.Έχουμε φτιάξει μια private method addEntity που προσθέτει ένα entity στο πίνακα entities .
6.Έχουμε μια μέθοδο getoffset που επιστρέφει το current offset και το αυξάνει κατά 4 για να είναι έτοιμο να δείξει τη νέα μεταβλητή.
7.Έχουμε φτιάξει τη κλάση Argument με παραμέτρους :

- το parMode που δείχνει πως περνιέται το argument (με τιμή(CV) ή αναφορα(REF) )
- next_argument που δείχνει στο διπλανό argument

-Και έχουμε μια μέθοδο set_next που ενημερώνει το next_argument .

8.Έχουμε μια κλάση Entity που έχει παραμέτρους name και type ("VARIABLE","FUNCTION","PARAMETER","TMPVAR").
9.Έχουμε μια κλάση Variable που κάνει extend το Entity και έχει offset ανάλογο με αυτό που υπάρχει στο Scope.
10.Έχουμε μια κλάση Function που κάνει extend το Entity και έχει ένα startQuad που αφορά τον ενδιάμεσο κώδικα και θα αρχικοποιεί :

- το quad label της πρώτης quad της function
- ένα πινακα arguments που έχει μέσα όλα τα Arguments objects
- ενα framelength που δείχνει το framelength της συανρτησης.

-Μέσα έχουμε κάποιες μεθόδους που ενημερώνουν τις παραμέτρους.
11.Έχουμε μια κλάση Parameter που κανει extend το Entity, έχει ένα parMode το οποίο θα παίρνει είτε "in","inout","inandout" και ένα offset ανάλογα με το πόσο αρχίζει στο scope .
12.Έχουμε μια κλάση TempVariable που κάνει extend το Entity και έχει offset ανάλογο με το πόσο αρχίζει στο scope.

<u>Φτιάχνουμε κάποιες μεθόδους για την υλοποίηση του Symbol Table:</u>

- add_newScope(): Φτιάχνει νέο Scope μόλις μπούμε σε συνάρτηση
- print_scopes():Τυπώνει κάθε φορά το current scope και αυτό που βρίσκεται από πάνω του, αν υπάρχει.
- add_function_entity(name): Προσθέτει ένα function entity
- add_variable_entity(name): Προσθέτει ένα entity Variable
- add_parameter_entity(name,par_mode): Προσθέτει ένα entity Parameter
- add_function_argument(func_name,parMode): Προσθέτει ένα argument σε μια υπάρχουσα συνάρτηση
- search_entity(name,etype):Ψάχνει να βρει entity με το συγκεκριμένο όνομα και τύπο στο current scope και τα enclosing.
- search_entity_by_name(name): Ψάχνει entity με το συγκεκριμένο όνομα στο current scope και τα enclosing.
- unique_enity(name,etype,nested_level): Ψάχνει να βρει αν υπάρχει κι άλλο entity με το ίδιο όνομα και τύπο.
- var_is_parameter(name,nested_level): Ψάχνει να βρει αν υπάρχει το όνομα ήδη δηλωμένο σαν parameter.
- update_quad_function_entity(name): Ενημερώνει το αρχικό quad label της entity Function
- update_func_entity_framelength(name,framelength): Ενημερώνει το framelength της συνάρτησης.

Προκειμένου να λειτουργήσει ο symbol table θα καλέσουμε κάποιες από αυτές στο parser μας:

1. def parse(self):Πριν κληθεί το block() προσθέτουμε στο πίνακα scopes_array ένα Scope που θα λειτουργήσει σαν global Scope.
2. def block(string): Έχουμε κάνει το block να παίρνει σαν όρισμα ένα string, έτσι ώστε να δουλεύουν σωστά οι μέθοδοι update (update_quad_function_entity και update_func_entity_framelength). Μέσα στη συνάρτηση έχουμε βάλει να καλούνται οι μέθοδοι update_quad_function_entity(name), update_func_entity_framelength (name) και print_scopes(), έτσι ώστε όταν τελειώνει η μετάφραση μιας συνάρτησης, να τυπώνεται το scope.
3. def varlist(): Όταν έχουμε εντολή declare και βρούμε "IDENTIFIER" καλούμε την add_variable_entity έτσι ώστε να προσθέσουμε το νέο entity στο current scope.
4. def subprogram(): Αν βρουμε εντολή function και μετά "IDENTIFIER" , φτιάχνουμε νέο ( scope add_newScope() )και προσθέτουμε το νέο function entity στο current scope με την εντολή (add_function_entity).
5. funcbody(function_name):Βάζουμε να παίρνει όρισμα έτσι ώστε να μπορώ να πάρω το όνομα της συνάρτησης.
6. formalpars(function_name): Το ίδιο με από επάνω.
7. formalparist(function-name): Το ίδιο με από επάνω.
8. formalparitem(function_name): Αν βρούμε argument καλούμε το add_function_argument(func_name,parMode) και add_parameter_entity(name,par_mode).

Έχουμε και κάποιες global μεταβλητές :

- scope_counter: μετράει το level που βρίσκεται το scope.
- scopes_array: πίνακας που έχει τα scopes.
- main_framelenght: μήκος εγγραφήματος δραστηριοποίησης

-Ενδιάμεσος Κώδικας:

Η λογική του ενδιάμεσου κώδικα είναι, ότι δημιουργούνται 4άδες (quad). Το quad αποτελείτε από τα:

- op(πράξη),
- argument1(όρισμα),
- argument2(όρισμα)
- result(αποτέλεσμα)

Τα quads θα χρησιμοποιηθούνε για την παραγωγή εντολών σε assembly. Επίσης μέσο των quads, υλοποιούμε την μετατροπή του αρχείου(test.stl) σε κώδικα C(test.int).

Σε κάθε quad έχουμε θέσει μια κατάλληλη ετικέτα.

Έχουμε κάποιες global μεταβλητές:

- quad_List: πίνακας από quads.
- nextquad: δείχνει σε ποιο στοιχείο του quad βρισκόμαστε.
- var_dict: dictionary όπου κρατάει τον αριθμό των quads.
- next_temp_num_vars: μετρητής για να αυξάνουμε τον αριθμό στο dictionary.

Έχουμε μια κλάση για τον ενδιάμεσο κώδικα, την Quad όπου:

- αρχικοποιείται το: label, op(operator),arg1(όρισμα $1^{ο}$), arg2(όρισμα $2^{ο}$) και το res(result).

-Έχουμε 2 private μεθόδους:

- __str__: μετατρέπει την κλάση Quad στην κατάλληλη μορφή 4άδας.
- tofile: την χρησιμοποιούμε για να γράφουμε στο αρχειο (test.int)

Έχουμε ξεχωριστές συναρτήσεις που υλοποιούν, τις λειτουργίες του ενδιάμεσου κώδικα. Αυτές είναι οι εξής :

- nextQuad: επιστρέφει την global μεταβλητή nextquad .
- generate_quad: δημιουργεί την 4άδα quad.

- new_temp: προσθέτει στο dictionary έναν καινούργιο αριθμό, ανάλογα με τα νέα quad, που θα χρησιμοποιηθεί ως ετικέτα(label) σε κάθε quad
- backpatch: παίρνει σαν όρισμα έναν πίνακα από 4άδες στους οποίους το res(result) δεν είναι συμπληρωμένο και μια ετικέτα (label) η οποία θα συμπληρώσει τις 4άδες αυτές.
- make_list: παίρνει σαν όρισμα ένα label και δημιουργεί μια λίστα ετικετών 4άδων(quad) που περιέχει μόνο την ετικέτα(label).
- merge: παίρνει σαν όρισμα δύο λίστες και δημιουργεί μια λίστα ετικετών 4άδων(quad) από την συγχώνευση των δύο λιστών.
- create_int_file: παίρνει τον πίνακα quad_List και γράφει στο αρχείο test.int ένα-ένα τα quads με τα label(ετικέτες) τους.
- transform_code_to_c: μετατρέπει τις 4άδες (quads) σε κώδικα C.
- create_c_code_file: δημιουργεί το αρχείο C.

-Οι συναρτήσεις αυτές, καλούνται σε διάφορες μεθόδους του parser(συντακτικού αναλυτή). Παρακάτω αναφέρουμε πού καλούνται:

Στην μέθοδο parse καλούμε: generate_quad('begin_block',tokens[1][1])

generate_quad('halt')

generate_quad('end_block',tokens[1][1]).

Στην μέθοδο block καλούμε: update_quad_function_entity(ενημερώνουμε με ποιό quad αρχίζει το block).

Στην μέθοδο subprogram καλούμε:
generate_quad('begin_block',temp_func_name_for_block)

generate_quad('end_block',temp_func_name_for_block).

Στην assignment_stat καλούμε: generate_quad (':=',value,'_',assign_var).

Στην expression καλούμε: generate_quad(διαφορετικό, ανάλογα με το αν κληθεί το optional_sign(μέθοδος) πχ:

generate_quad(op,temp_term,temp_term2,tmpvar)).

Στην term καλούμε:

generate_quad (temp_mul_oper,temp_factor,temp_factor2,tmpvar).

Στην actualparse καλούμε: generate_quad('par',temp_var,'RET'), generate_quad('call',temp_ident)

Στην actualparitem καλούμε: generate_quad (διαφορετικό, ανάλογα το είδος της παραμέτρου {τιμή,αναφορά,αντιγραφή},πχ:

generate_quad('par',temp_string_for_quad,'REF') ).

Στην if_stat καλούμε: nextQuad() ,backpatch( γίνεται η συμπλήρωση όσων 4άδων γίνεται να συμπληρωθούν μέσα στον κανόνα), makelist, generate_quad( jump , "_", "_", "_") με αυτό εξασφαλίζουμε ότι δεν θα πάμε στην κατάσταση else).

Στην condition καλούμε: nextQuad(), backpatch, merge( συσσωρεύουμε στην λίστα true 4αδες που δεν μπορούν να συμπληρωθούν και αντιστοιχούν σε αληθή αποτίμηση της λογικής παράστασης).

Στην bool_term καλούμε: nextQuad() ,backpatch, merge( συσσωρεύουμε στην λίστα false 4αδες που δεν μπορούν να συμπληρωθούν και αντιστοιχούν σε μη αληθή αποτίμηση της λογικής παράστασης).

Στην bool_factor : αν έχουμε "not" αντιστρέφουμε την λίστα που επιστρέφεται από το condition, έτσι ώστε να λειτουργεί σωστά το jump. Καλούμε generate_quad, makelist και για να πάμε στην σωστή εντολή κάνουμε generate_quad ( jump , "_", "_", "_").

Στην while_stat καλούμε: nextQuad() ,backpatch, generate_quad ('jump','_','_',quad_1).

Στην do_while καλούμε: nextQuad() ,backpatch.

Στην loop_stat καλούμε: nextQuad() ,generate_quad ('jump','_','_',quad_for_loop).

Στην exit_stat καλούμε: generate_quad ('jump','_','_','_').

Στην forcase καλούμε: nextQuad() ,backpatch, makelist (nextQuad()),

generate_quad ('jump','_','_', quad_for_loop).

Στην incase καλουμε: nextQuad() ,backpatch, makelist (nextQuad()),

generate_quad ('jump','_','_', temp__quad_incase).

Στην input_stat καλούμε: generate_quad( 'inp',tokens[token_index][1],'_','_').

Στην print_stat καλούμε: generate_quad ('out',temp_expr).

Στην return_stat καλούμε: generate_quad('retv',temp_expr).

# *3η Φάση (Τελικός Κώδικας Assembly)*

-Η λογική αυτής της φάσης είναι ότι γράφουμε σε ένα αρχείο (assembly.asm) κατάλληλες εντολές σε Assembly, ανάλογα με τα quads που έχουν δημιουργηθεί από την δεύτερη φάση και με βάση τα scopes του προγράμματος. Για να γίνει αυτό, χρησιμοποιούμε 4 μεθόδους, οι οποίες είναι οι εξής:

- gnvlcode (variable, register)
- loadvr (variable, register)
- storerv (register, variable)
- from_quad_to_assembly (quad,block)

-Έχουμε και κάποιες global μεταβλητές:

- assembly_file: αρχείο εγγραφής εντολών της Assembly.
- quads_inside_block: πίνακας που αποτελείται από quads όταν το op(operator{όρισμα}) είναι par.
- number_of_functions_mips: μετρητής ,του αριθμού των συναρτήσεων του προγράμματος, για να αρχικοποιήσουμε σωστά το L_... ( σαν label για κάθε συνάρτηση, χρήσιμο για τα jump L).
- L_counter: χρησιμοποιείτε για να γίνει σωστά η εκτέλεση των εντολών assembly σε περίπτωση που έχουμε function.

Η μέθοδος gnvlcode: μεταφέρει στον $t0 την διεύθυνση μιας μη-τοπικής μεταβλητής. Από τον πίνακα συμβόλων(symbol_table) βρίσκει πόσα επίπεδα πάνω βρίσκεται αυτή η μεταβλητή. Μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει.

Η μέθοδος loadvr: ανάλογα με τα ορίσματα γράφει τις αντίστοιχες εντολές στο αρχείο assembly.asm. Αναλαμβάνει το φόρτωμα δεδομένων στους καταχωρητές μέσω των εντολών lw-lb.

Η μέθοδος saverv: ανάλογα με τα ορίσματα γράφει τις αντίστοιχες εντολές στο αρχείο assembly.asm. Αναλαμβάνει την αποθήκευση δεδομένων στους καταχωρητές μέσω των εντολών sw-sb.

Η from_quad_to_assembly: Ανάλογα με το op(operator {ορισμα}) του quad, μετατρέπει τα quad ενός block σε εντολές assembly. Τέλος η συγκεκριμένη μέθοδος καλείτε στην block με ορίσματα (quad, string), όπου το quad είναι στοιχείο του πίνακα quad_list ο οποίος ξεκινάει από το start_quad_in_block μέχρι το τέλος του πίνακα. Η μεταβλητή αυτή είναι το αποτέλεσμα της μεθόδου → update_quad_function_entity, που επιστρέφει το index του quad που αρχίζει το block.

# *Εκτέλεση αρχείου*

Για να λειτουργήσει ο κώδικας και να δημιουργηθούν τα ζητούμε αρχεία (file_name.int, file_name.c και file_name.asm) θα πρέπει να γράψουμε σε ένα αρχείο, με κατάληξη .stl, τον κώδικα που επιθυμούμε σε γλώσσα Starlet. Το αρχείο αυτό θα πρέπει να είναι στον ίδιο φάκελο με την main.py. Για να τρέξει το πρόγραμμα γράφουμε στο terminal

την εντολή → python3 main.py file_name.stl