Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Πολυτεχνική Σχολή

Πανεπιστήμιο Πατρών

Flex/Bison Project 2024

Χαράλαμπος Αναστασίου, 1093316, 3ο έτος, up1093316@ac.upatras.gr

Κωνσταντίνος Αναστασόπουλος, 1093320, 3ο έτος, up1093320@ac.upatras.gr

Θεόφραστος Παξιμάδης, 1093460, 3ο έτος, up1093460@ac.upatras.gr

Μέρος Α: Περιγραφή της BNF

Ακολουθεί σε BNF ο συντακτικός ορισμός της γραμματικής της φανταστικής αντικειμενοστραφούς γλώσσας προγραμματισμού που μας ζητείται να αναλύσουμε. Κάτω από τους κανόνες υπάρχουν με εναλλακτικό χρώμα σχόλια που τους επεξηγούν και αιτιολογούν την ύπαρξή τους με βάση τα ζητούμενα.

<program> ::= <c> <public_class> <c> | <c> <public_class> <c> <program>

Το πρόγραμμα αποτελείται από μία κλάση ή την παράθεση κλάσεων. Όπου υπάρχει <c> μπορεί προαιρετικά να μπει σχόλιο

<public_class> ::= “public class” <capital_identifier> “{“ <class_block> “}”

Η κλάση ορίζεται ως public και το όνομα ξεκινά με κεφαλαίο.

<class_block> ::= <variable_declarations> <method_declarations> <nested_class>

Προηγείται η δήλωση μεταβλητών, ακολουθεί η δήλωση μεθόδων και τέλος, έπεται η δήλωση ένθετων κλάσεων.

<nested_class> ::= <c> <public_class> <c> | “”

Οι ένθετες κλάσεις είναι προαιρετικές και δηλώνονται όπως όλες οι υπόλοιπες.

<variable_declarations> ::= <variable_declaration>

| <variable_declaration> <variable_declarations>

| “”

Η δήλωση μίας ή περισσότερων μεταβλητών είναι προαιρετική

<variable declaration> ::= <c> <variable_modifier> <data_type><identifier> “;” <c>

| <c> <extended_var_declaration> ";" <c>

| <c> <capital_identifier> <identifier> “=” <object_instance> <c>

Η δήλωση μεταβλητών έχει τη μορφή δήλωσης μίας μεταβλητής, δήλωσης πολλαπλών μεταβλητών ίδιου τύπου ή αρχικοποίησης αντικειμένου κλάσης.

<extended_var_declaration> ::= <int_declaration> | <char_declaration> | <double_declaration> | <boolean_declaration> | <string_declaration>

Η δήλωση πολλαπλών μεταβλητών προϋποθέτει οι μεταβλητές να είναι ίδιου τύπου.

<int_declaration> ::= <variable_modifier> "int" <identifier> “=” <integer_literal> <extra_ints_a>

| <variable_modifier> "int" <identifier> <extra_ints>

Η δήλωση πολλαπλών μεταβλητών είναι της μορφής: τροποποιητής, τύπος δεδομένου, αναγνωριστής, ίσον, τιμή και περισσότερες δηλώσεις με ανάθεση τιμής, ή το παραπάνω αλλά χωρίς ανάθεση τιμών.

<char_declaration> ::= <variable_modifier> "char" <identifier> “=” <char_literal> <extra_chars_a>

| <variable_modifier> "char" <identifier> <extra_chars>

<double_declaration> ::= <variable_modifier> "double" <identifier> “=” <double_literal <extra_doubles_a>

| <variable_modifier> "double" <identifier> <extra_doubles>

<boolean_declaration> ::= <variable_modifier> "boolean" <identifier> “=” <boolean_literal> <extra_booleans_a>

| <variable_modifier> "boolean" <identifier> <extra_booleans>

<string_declaration> ::= <variable_modifier> "string" <identifier> “=” <string_literal> <extra_strings_a>

| <variable_modifier> "string" <identifier> <extra_strings>

<extra_ints_a> ::= <extra_int_a> | <extra_int_a> <extra_ints_a>

Οι έξτρα δηλώσεις με ανάθεση είναι μία ή περισσότερες.

<extra_int_a> ::= “,” <identifier> “=” <integer_literal> | “”

Η έξτρα προαιρετική δήλωση με ανάθεση έχει τη μορφή: κόμμα, αναγνωριστής, ίσον, τιμή.

<extra_ints> ::= <extra_int> | <extra_int> <extra_ints>

Οι έξτρα απλές δηλώσεις είναι μία ή περισσότερες.

<extra_int> ::= “,” <identifier> | “”

Η έξτρα προαιρετική απλή δήλωση έχει τη μορφή: κόμμα, αναγνωριστής.

<extra_chars_a> ::= <extra_char_a> | <extra_char_a> <extra_chars_a>

Ομοίως με πάνω.

<extra_char_a> ::= “,” <identifier> “=” <char_literal> | “”

Ομοίως με πάνω.

<extra_chars> ::= <extra_char> | <extra_char> <extra_chars>

Ομοίως με πάνω.

<extra_char> ::= “,” <identifier> | “”

Ομοίως με πάνω.

<extra_doubles_a> ::= <extra_double_a> | <extra_double_a> <extra_doubles_a>

Ομοίως με πάνω.

<extra_double_a> ::= “,” <identifier> “=” <double_literal> | “”

Ομοίως με πάνω.

<extra_doubles> ::= <extra_double> | <extra_double> <extra_doubles>

Ομοίως με πάνω.

<extra_double> ::= “,” <identifier> | “”

Ομοίως με πάνω.

<extra_booleans_a> ::= <extra_boolean_a> | <extra_boolean_a> <extra_booleans_a>

Ομοίως με πάνω.

<extra_boolean_a> ::= “,” <identifier> “=” <boolean_literal> | “”

Ομοίως με πάνω.

<extra_booleans> ::= <extra_boolean> | <extra_boolean> <extra_booleans>

Ομοίως με πάνω.

<extra_boolean> ::= “,” <identifier> | “”

Ομοίως με πάνω.

<extra_strings_a> ::= <extra_string_a> | <extra_string_a> <extra_strings_a>

Ομοίως με πάνω.

<extra_string_a> ::= “,” <identifier> “=” <string_literal> | “”

Ομοίως με πάνω.

<extra_strings> ::= <extra_string> | <extra_string> <extra_strings>

Ομοίως με πάνω.

<extra_string> ::= “,” <identifier> | “”

Ομοίως με πάνω.

<method_declarations> ::= <c> <method_declaration> <c> | <c> <method_declaration> <c> <method_declarations> | “”

Η δήλωση μεθόδων είναι προαιρετική και αποτελείται από μία δήλωση ή την παράθεση δηλώσεων.

<method_declaration> ::= <method_modifier> <return_type> <identifier> “(“ <parameters> “)” “{“ <method_block> <return> “}”

Η μέθοδος δηλώνεται ως: τροποποιητής, τύπος επιστροφής, αναγνωριστής, παράμετροι μέσα σε παρένθεση και μπλοκ κώδικα μέσα σε άγκιστρα.

<method_modifier> ::= "Public" | "Private" | ""

Ο προαιρετικός τροποποιητής μεθόδου είναι Public ή Private.

<paramaters> ::= <data_type><identifier>

| <data_type><identifier> “,” <parameters>

| “”

Η προαιρετική δήλωση παραμέτρων είναι της μορφής τύπος δεδομένου και αναγνωριστής, και αν οι παράμετροι είναι περισσότερες από μία, τότε χωρίζονται με κόμμα.

<method_block> = <variable_declarations> <commands>

Στο μπλοκ μεθόδου προηγούνται οι δηλώσεις μεταβλητών και ακολουθούν οι εντολές.

<commands> ::= <c> <command> <c> | <c> <command> <c> <commands> | “”

Οι εντολές είναι καμία, μία ή πολλές.

<command> ::= <assignment> | <do_loop> | <for_loop> | <if_else> | <switch_case> | <print> | <break> | <method_call>

Οι πιθανές εντολές είναι: ανάθεση, do, for, if-else, switch, print, break, method-call.

<assignment> ::= <identifier> “=” <expression>

Οι αναθέσεις είναι της μορφής: αναγνωριστής, ίσον, έκφραση.

<expression> ::= <method_call> | <literal>“;”| <object_instance> | <operations>“;”| <member_access>

Η έκφραση μπορεί να είναι κλήση μεθόδου, κυριολεκτικό, αντικείμενο κλάσης, πράξεις ή πρόσβαση σε αντικείμενο κλάσης.

<member_access> ::= <capital_identifier>”.”<identifier> “(”<arguments>”)” “;”

| <capital_identifier> ”.” <identifier> “;”

Η πρόσβαση σε μέλη μίας κλάσης(μεταβλητή, μέθοδος) ακολουθεί την μορφή:

αναγνωριστής / όνομα αντικειμένου , τελεία (.) , αναγνωριστής μέλους κλάσης με έξτρα παρενθέσεις και ορίσματα στην περίπτωση που το μέλος είναι μέθοδος.

<method_call> ::= <identifier> “(” <arguments> ”)” “;”

| <identifier>”.”<identifier> “(”<arguments>”)” “;”

Η κλήση μεθόδου είναι είτε απλή, είτε κλήση μεθόδου που ανήκει σε αντικείμενο κλάσης.

<arguments> ::= <literal> | <literal> “,” <arguments> | <identifier> | <identifier> “,” <arguments> |

Τα ορίσματα είναι κανένα, ένα ή περισσότερα κυριολεκτικά, ή μεταβλητές.

<object_instance> ::= “new” <capital_identifier> “(“ “)” ”;”

Η δημιουργία αντικειμένου κλάσης ορίζεται με new, αναγνωριστή και παρενθέσεις.

<operations> ::= <literal><operator_symbol><operations> | <literal>

| <identifier><operator_symbol><operations> | <identifier>

Ως πράξεις θεωρούνται τα κυριολεκτικά, οι σταθερές και η μία ή περισσότερες πράξεις μεταξύ τους.

<operator_symbol> ::= “-” | “*” | “/” | “+”

Οι τελεστές είναι: πλην, επί, δια, συν.

<do_loop> ::= “do” <c> “{“ <commands> ”}” <c> “while” “(“<conditions>“)” “;”

Το do loop ακολουθεί το πρότυπο της C, do, μπλοκ, while, συνθήκες.

<conditions> ::= <condition> | <condition><logic_operator><conditions>

Η συνθήκες είναι μία ή περισσότερες χωρισμένες με λογικούς τελεστές.

<condition> ::= <literal><compare_symbol><identifier>

| <identifier><compare_symbol><identifier>

| <literal><compare_symbol><literal>

| <identifier><compare_symbol><literal>

Η κάθε συνθήκη κάνει συγκρίσεις ανάμεσα σε κυριολεκτικά ή τιμές μεταβλητών.

<compare_symbol> ::= “>” | “<” | “==” | “!=” | “>=” | “<=”

Οι συγκρίσεις είναι οι ίδιες που βρίσκει κανείς σε σχεδόν κάθε άλλη γλώσσα γενικής χρήσης.

<logic_operators> ::= “&&” | “||”

Οι λογικοί τελεστές είναι οι AND και OR.

<for_loop> ::= “for” “(“ <for_assignment> “;” <conditions> “;” <increment> “)” <c> “{” <commands> “}” <c>

Το for ακολουθεί το πρότυπο της C έχοντας μέσα στην παρένθεση ανάθεση, συνθήκη και επαύξηση.

<for_assignment> ::= <identifier> “=” <literal>| <identifier> “=” <identifier>

Κατά την ανάθεση χρησιμοποιούνται κυριολεκτικά αλλά και τιμές μεταβλητών.

<increment> ::= <identifier> “=” <identifier><operator_symbol><literal>

| <identifier> “=” <identifier><operator_symbol><identifier>

Η επαύξηση δεν ακολουθεί το πρότυπο των περισσοτέρων γλωσσών (++, --) αλλά το: μεταβλητή, ίσον, μεταβλητή και πράξεις.

<if_else> ::= <c> “if” “(” <conditions> “)” <c> “{” <commands> “}” <c> <else_if> <c> <else> <c>

Τα if-else, else-if, else και switch ακολουθούν το πρότυπο της C.

<else_if>::= <c> “else if” “(” <conditions> “)” <c> “{” <commands> “}” <c>

| <c> “else if” “(” <conditions> “)” <c> “{” <commands> “}” <c> <else_if> <c>

| “”

<else>: <c> “else” <c> “{” <commands> “}” | “”

<switch_case> ::= “switch” “(” <identifier> ”)” <c> “{” <c> <cases> <c> “default:” <commands> “}” <c>

|“switch” “(” <identifier> ”)” “{” <cases> “}”

<cases> ::= “case” <literal> “:” | “case” <literal> “:” <cases>

Τα cases είναι ένα ή περισσότερα.

<print> ::= “out.print” “(“ <text><variables> “)” “;”

Το print, όπως και στη C, έχει κείμενο και μεταβλητές.

<text> = <string_literal>

Το κείμενο είναι αλφαριθμητικό κυριολεκτικό.

<variables> ::= “,” <identifier> | “,” <identifier><variables> | “”

Οι μεταβλητές είναι καμία, μία, ή πολλές.

<return> ::= “return” <literal> “;” | “return” <identifier> “;” | “return” <operations> “;”

| “return” <method_call> | “return” <member_access>

Οι μέθοδοι επιστρέφουν κυριολεκτικό, αναγνωριστή, πράξεις, κλήση μεθόδου, ή μέλος κλάσης με πρόσβαση μέσω αντικειμένου.

<break> ::= “break” “;”

<variable_modifier> ::= “public” | “private” | “”

Ο προαιρετικός τροποποιητής μεταβλητής είναι public ή private.

<data_type> ::= "int" | "char" | "double" | "boolean" | "String"

Ο τύπος δεδομένου είναι int, char, double, boolean και String.

<return_type> ::= <data_type> | ""

Ο τύπος επιστροφής είναι κάποιος από τους παραπάνω ή τίποτα.

<literal> ::= <integer_literal> | <char_literal> | <double_literal> | <boolean_literal> | <string_literal>

Τα κυριολεκτικά έχουν έναν από τους παραπάνω τύπους.

<integer_literal> ::= <digit> | <digit><integer_literal>

Οι ακέραιοι αποτελούν παράθεση ενός ή περισσότερων ψηφίων.

<double_literal> ::= <integer_literal> “.” <integer_literal> “d”

Οι doubles είναι της μορφής: ακέραιος, τελεία, ακέραιος, d.

<boolean_literal> ::= "true" | "false"

Τα boolean κυριολεκτικά είναι true ή false.

<string_literal> ::= “”” <char>“”” | “”” <char><string_literal>“””

Τα Strings περιβάλλονται από διπλά εισαγωγικά και αποτελούν παραθέσεις χαρακτήρων.

<no_newline_string_literal> ::= “”” <char>“”” | “”” <char><no_newline_string_literal>“””

Τα Strings αυτά δεν περιλαμβάνουν το newline character.

<char_literal> ::= “”<symbol>“” | “” <escape_sequence>“” | “” <letter>”” | “” <digit>“”

Οι χαρακτήρες είναι σύμβολα, χαρακτήρες αποφυγής, γράμματα ή ψηφία, εγκλεισμένα σε μονά εισαγωγικά.

<no_newline_char_literal> ::= “”<symbol>“” | “” <no_newline_escape>“” | “” <letter>”” | “” <digit>“”

Οι χαρακτήρες αυτοί δεν περιλαμβάνουν το newline character.

<char> ::= <symbol> | <escape_sequence> | <letter> | <digit>

Εδώ δε χρησιμοποιούνται μονά εισαγωγικά γιατί αυτός ο κανόνας είναι το building block των Strings.

<symbol> ::= '+' | '-' | '*' | '/' | '%' | '=' | '&' | '|' | '^' | '~' | '!' | '<' | '>' | '?' | ':' | ';' | ',' | '.' | '(' | ')' | '[' | ']' | '{' | '}' | '#' | '_' | '$' | '@' |

Όλα τα standard σύμβολα του πληκτρολογίου.

<escape_sequence> ::= '\n' | '\t' | '\r' | '\\' | '\'' | '\"' | '\a' | '\b' | '\f' | '\v' | '\?' | '\0' | '\x'

Τα escape sequences της C.

<no_newline_escape> ::= '\t' | '\r' | '\\' | '\'' | '\"' | '\a' | '\b' | '\f' | '\v' | '\?' | '\0' | '\x'

Τα παραπάνω, αλλά χωρίς το newline.

<identifier> ::= <identifier_head><identifier_body>

Ο αναγνωριστής αποτελείται από το πρώτο του σύμβολο και ενδεχομένως άλλα σύμβολα.

<identifier_head> ::= “_” | <letter>

Το πρώτο σύμβολο μπορεί να είναι γράμμα ή underscore.

<identifier_body> ::= <identifier_head> | <digit> | <identifier_head><identifier_body>

| <digit><identifier_body> | “”

Τα υπόλοιπα σύμβολα μπορούν να είναι οποιαδήποτε ψηφία, γράμματα, ή underscores.

<capital_identifier> ::= <capital_letter><identifier_body>

Οι αναγνωριστές κλάσης ξεκινούν πάντοτε με κεφαλαίο.

<letter> ::= <lowercase_letter> | <capital_letter>

Τα γράμματα διακρίνονται σε πεζά και κεφαλαία.

<lowercase_letter> ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

Πεζά λατινικά γράμματα.

<capital_letter> ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

Κεφαλαία λατινικά γράμματα.

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Ψηφία από το μηδέν έως το εννιά.

<c> ::= <comment> | <comment> <c> | “”

Τα σχόλια, κάθε φορά που μπορούν να μπουν, είναι κανένα, ένα, ή περισσότερα και ο κανόνας λέγεται <c> γιατί επαναλαμβάνεται πολλές φορές και εναλλακτικά, με μεγαλύτερο όνομα θα έπιανε πολύ χώρο.

<comment> ::= <one_line_comment> | <multiple_line_comment>

Τα σχόλια είναι μίας ή πολλών γραμμών.

<one_line_comment> :== “//” <no_newline_string_literal>

Τα σχόλια μίας γραμμής, όπως και στη C, ξεκινούν με // και σταματούν όταν διαβάσουν newline.

<multiple_line_comment> ::= “/*” <string_literal> “*/”

Τα σχόλια πολλών γραμμών, όπως και στη C, περιβάλλονται από /* και */, και διαβάζουν οτιδήποτε βρίσκεται ενδιάμεσα.

Ορισμός Συναρτήσεων

Στα επόμενα Μέρη Β, Γ αναφέρονται ορισμένες συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση των Ερωτημάτων. Στην ενότητα αυτή περιγράφονται όλες οι συναρτήσεις με την σειρά που αναφέρονται στα Μέρη, για τη διευκόλυνση του αναγνώστη.

`void addVar(char *name, char *expr_type)`

Προσθέτει, κατά τη δήλωση μεταβλητών, στον πίνακα data_table μία εγγραφή με το όνομα της μεταβλητής και τον τύπο της έκφρασης που της ανατίθεται.

`int compareAll(ExprEntry *data_table)`

Ενεργεί πάνω στα περιεχόμενα του πίνακα data_table και ελέγχει εάν ο τύπος των τιμών που έχουν ανατεθεί στις μεταβλητές είναι ο ίδιος. Αν δεν είναι ίδιος για όλες τις μεταβλητές, επιστρέφει 0, αλλιώς επιστρέφει 1.

`void clearTable(ExprEntry *data_table)`

Συνάρτηση η οποία “καθαρίζει” τον πίνακα data_table θέτοντας τα πεδία name και expr_type κάθε εγγραφής του πίνακα ίσα με την κενή γραμματοσειρά “” .

`void add_method(char *modifier, char *type, char *name)`

Κάθε φορά που ολοκληρώνεται κάποιο method declaration αποθηκεύονται στον πίνακα m_table ο modifier, ο τύπος επιστροφής τιμής και το όνομα της εκάστοτε μεθόδου.

`void check_method(const char *name)`

Η συνάρτηση αυτή εξετάζει κάθε φορά που γίνεται κλήση μεθόδου, αν αυτή υπάρχει στον πίνακα m_table, δεχόμενη ως όρισμα το όνομά της. Αν δεν υπάρχει, το πρόγραμμα τερματίζει και εμφανίζεται το αντίστοιχο ενημερωτικό μήνυμα.

`void add_declaredVar(char *name, char *expr_type)`

Δέχεται ως ορίσματα ένα όνομα μεταβλητής και τον τύπο της και κάνει εισαγωγή αυτών στον πίνακα declaredVar_table. Καλείται στην περίπτωση που έχω δήλωση πολλαπλών μεταβλητών χωρίς ανάθεση τιμής (π.χ. int x,y,z). Η υλοποίησή της είναι ίδια με την addVar.

`char *searchVariable(char *varName)`

Πραγματοποιεί αναζήτηση στον πίνακα declaredVar_table για την εύρεση του τύπου της μεταβλητής, το όνομα της οποίας δέχεται ως όρισμα.

`void add_id(char *modifier, char *name)`

Κατά την δήλωση μιας μεταβλητής προσθέτει στον πίνακα id_table μία εγγραφή με το όνομα και τον modifier της μεταβλητής αυτής.

`void add_extra(char *name)`

Στην περίπτωση δηλώσεων πολλών μεταβλητών με ή χωρίς ανάθεση, προσθέτει προσωρινά στον πίνακα ex_table μόνο το όνομα των έξτρα μεταβλητών (μετά το πρώτο κόμμα). Αυτό γίνεται επειδή αρχικά δεν γνωρίζω τον modifier των έξτρα δηλώσεων, καθώς εκτελούνται πρώτα οι εμφωλευμένοι κανόνες extra_variables και extra_assigned_variables και μετά ο κανόνας variable_declaration.

`void clear_extra()`

Συνάρτηση η οποία “καθαρίζει” τον πίνακα ex_table θέτοντας το πεδίο name κάθε εγγραφής του πίνακα ίσο με την κενή γραμματοσειρά “” .

`void check_var_private(char *name)`

Η συνάρτηση αυτή δέχεται ως όρισμα το όνομα μιας μεταβλητής και εξετάζει εάν αυτή υπάρχει στον πίνακα id_table. Αν δεν υπάρχει, το πρόγραμμα τερματίζει και εμφανίζεται το αντίστοιχο ενημερωτικό μήνυμα.

`void clear_var_private()`

Συνάρτηση η οποία “καθαρίζει” τον πίνακα id_table θέτοντας τα πεδία name και modifier κάθε εγγραφής του πίνακα ίσα με την κενή γραμματοσειρά “” .

`void clear_private_methods()`

Συνάρτηση η οποία “καθαρίζει” τον πίνακα m_table θέτοντας τα πεδία name type και modifier κάθε εγγραφής του πίνακα ίσα με την κενή γραμματοσειρά “” .

`void add_assign(char *name, char *value)`

Καλείται κάθε φορά που συναντάται μία ανάθεση αριθμητικής τιμής τύπου INT_NUM ή DOUBLE_NUM σε μεταβλητή. Προσθέτει μία εγγραφή στον πίνακα assign_table με πεδία ονόματος μεταβλητής και τιμής αυτά που δέχεται ως ορίσματα.

`char *findOperationValue(char *name)`

Συνάρτηση η οποία εκτελεί αναζήτηση (bottom up) στον πίνακα assign_table το όνομα της μεταβλητής που δέχεται ως όρισμα και εάν το βρει επιστρέφει την τιμή της, αλλιώς ενημερώνει με κατάλληλο μήνυμα σφάλματος ότι στην μεταβλητή δεν έχει ανατεθεί αριθμητική τιμή. Χρησιμοποιείται στον κανόνα <operations> οταν στην αριθμητική έκφραση υπεισέρχονται και μεταβλητές.

Μέρος Β: Δηλώσεις Μεταβλητών

Ερώτημα Α: Ανάθεση τιμής κατά τη δήλωση μιας μεταβλητής

Από το Μέρος Α, ο κανόνας για την δήλωση μεταβλητών είναι ο ακόλουθος:

```
variable_declaration: c variable_modifier data_type variable QMARK c
| c CLASS_NAME variable EQUAL object_instance QMARK c
;
```

ο οποίος επιτρέπει μόνο την δήλωση μεταβλητών της μορφής int x;. Για να υπάρχει η δυνατότητα αναγνώρισης ανάθεσης τιμής κατά την δήλωση, αρκεί να προσθέσουμε τον παρακάτω κανόνα στο variable_declaration:

```
: variable_modifier data_type variable EQUAL expression QMARK c
```

Ο παραπάνω κανόνας επιτρέπει, κατά την δήλωση μιας μεταβλητής, την ανάθεση τιμής σε αυτή. Δηλαδή επιτρέπει δηλώσεις της μορφής int x = 5;, double z = 21.2d; κ.λπ. Ωστόσο, ο κανόνας αυτός δεν ελέγχει αν η τιμή που ανατίθεται στην μεταβλητή είναι ***σύμφωνη με τον τύπο της***. Με άλλα λόγια, επιτρέπονται αναθέσεις της μορφής: boolean z = 2; και String z = true;. Ο έλεγχος της σωστής ανάθεσης τιμής περιλαμβάνεται στο επόμενο ερώτημα.

Ερώτημα Β: Δήλωση πολλαπλών μεταβλητών ίδιου τύπου (με ή χωρίς ανάθεση)

Στο παρόν ερώτημα έχει προστεθεί η δυνατότητα πολλαπλών δηλώσεων μεταβλητών ίδιου τύπου με ή χωρίς ανάθεση, καθώς και ένας έλεγχος ότι η ανάθεση τιμών στις μεταβλητές κατά την δήλωση τους είναι σύμφωνη με τον τύπο τους.

Για να υποστηρίζονται πολλαπλές δηλώσεις στην ίδια γραμμή, πρέπει να γίνουν οι αντίστοιχες αλλαγές στον κανόνα για την δήλωση μεταβλητών. Ο κανόνας για την δήλωση μεταβλητών μέχρι στιγμής είναι ο ακόλουθος:

```
variable_declaration: c variable_modifier data_type variable QMARK c
        | c CLASS_NAME variable EQUAL object_instance QMARK c
        | c variable_modifier data_type variable EQUAL expression QMARK c
        ;
```

Προφανώς, επειδή το πλήθος των μεταβλητών σε κάθε δήλωση είναι αυθαίρετο, οι πρόσθετες μεταβλητές πρέπει να ορίζονται με αναδρομή. Για αυτό τον λόγο, έχουν γίνει οι παρακάτω αλλαγές στον κανόνα variable_declaration:

```
variable_declaration:   c variable_modifier data_type variable extra_variables QMARK c
| c CLASS_NAME variable EQUAL object_instance QMARK c
| c variable_modifier data_type variable EQUAL normal_type extra_assigned_variables QMARK c
;


extra_variables: COMMA variable extra_variables
        | ;

extra_assigned_variables: | COMMA variable EQUAL normal_type extra_assigned_variables
```

Οι κανόνες extra_variables και extra_assigned_variables προσθέτουν αναδρομικά επιπλέον μεταβλητές στις αντίστοιχες δηλώσεις μεταβλητών. Για παράδειγμα, αν υπάρχει η δήλωση int x; τότε καλείται ο πρώτος κανόνας του variable_declaration και ο κανόνας extra_variables επιστρέφει το κενό. Αν όμως υπάρχει η δήλωση int x,y,z; , τότε ο κανόνας variable_declaration περιλαμβάνει μόνο την μεταβλητή x ενώ οι μεταβλητές y, και z, ενεργοποιούν τον κανόνα extra_variables. Ομοίως, για την περίπτωση που υπάρχει δηλώσεις της μορφής: int x = 1, y = 2, z = 3;.

Το δεύτερο σκέλος του ερωτήματος είναι ο έλεγχος πως σε κάθε δήλωση μεταβλητής, αν υπάρχει ανάθεση τιμής, τότε αυτή είναι σύμφωνη με τον τύπο της. Για τις δηλώσεις της μορφής int x; και int x,y,z; δεν χρειάζεται κάποιος έλεγχος καθώς δεν ορίζονται τιμές για τις μεταβλητές. Ο έλεγχος θα γίνει στην περίπτωση όπου η δήλωση συνοδεύεται από ανάθεση τιμής, δηλαδή μόνο στην περίπτωση όπου καλούνται ο τρίτος κανόνας του variable_declaration και ο κανόνας extra_assigned_variables. Στο συγκεκριμένο ερώτημα δεν έχει γίνει αλλαγή στους κανόνες του bison αλλά έχουν προστεθεί blocks κώδικα C (**actions**) στους παραπάνω κανόνες.

Ο έλεγχος σωστής ανάθεσης τιμής, αποτελείται από δύο μέρη. Το πρώτο σκέλος είναι να εξετάζει αν οι τιμές που ανατίθενται είναι του ίδιου τύπου σε κάθε γραμμή και το δεύτερο αν οι τιμές αυτές συμπίπτουν με τον τύπο της μεταβλητής.

Στο πρώτο σκέλος, σε κάθε δήλωση μεταβλητής με ανάθεση τιμής, εισάγονται στον πίνακα data_table οι μεταβλητές που ορίζονται στην ίδια δήλωση και ο τύπος των τιμών τους, με την χρήση της συνάρτησης **addVar**. Δηλαδή αν υπάρχει η δήλωση: int x = 1, int x = 2, int x = 3;, εισάγονται τα ονόματα των μεταβλητών και ο τύπος των τιμών τους που είναι INT_NUM (το αρχείο flex επιστρέφει INT_NUM κάθε φορά που διαβάζει integer αριθμό. Ομοίως με τις υπόλοιπες μεταβλητές). Επειδή το bison εκτελείται bottom-up, πρώτα καλείται ο κανόνας extra_assigned_variables και έπειτα ο variable_declaration. Άρα αν υπάρχουν δύο οι περισσότερες μεταβλητές, πρώτα θα εισάγει ο κανόνας extra_assigned_variables τις μεταβλητές και τον τύπο τους στον πίνακα data_table και στην συνέχεια θα κληθεί ο κανόνας variable_declaration που θα εισάγει την δική του μεταβλητή και τον τύπο της στον ίδιο πίνακα. Μετά, ο κανόνας variable_declaration θα καλέσει την συνάρτηση **compareAll** για να συγκρίνει τον τύπο των τιμών που έχουν εισαχθεί στις μεταβλητές. Αν δεν είναι ίδιος για όλες τις μεταβλητές, τότε το πρόγραμμα τερματίζει και εμφανίζει το αντίστοιχο ενημερωτικό μήνυμα.

Μέχρι το συγκεκριμένο σημείο, γίνεται έλεγχος μόνο στην ανάθεση ίδιου τύπου τιμών στις μεταβλητές. Δηλαδή δεν επιτρέπεται η δήλωση: int x = 1, z = true; αλλά επιτρέπεται η δήλωση int x = false, z = true;. Μένει να γίνει έλεγχος μεταξύ του τύπου των τιμών με το data_type της κάθε δήλωσης.

Αυτό επιτυγχάνεται εύκολα με συνεχόμενα if. Δηλαδή, αν ο τύπος των μεταβλητών είναι INT_NUM, τότε πρέπει και το data_type να είναι int, αλλιώς το πρόγραμμα τερματίζει. Ομοίως γίνεται ο έλεγχος για κάθε είδος μεταβλητών.

Έτσι, εξασφαλίζεται η σωστή ανάθεση τιμών κατά την δήλωση μίας ή περισσότερων μεταβλητών.

Ακολουθεί παράδειγμα με διάφορες δηλώσεις μεταβλητών, όπως ορίζει το συγκεκριμένο ερώτημα. Ο κώδικας που εξετάζεται είναι ο ακόλουθος:

```
public class Germanios {

// Single Declaration Assignments
int x = 2;
double y = 2.2d;
char x = '\n';
char symbol2 = 'A';
String name = "my_name";
private boolean valid = true;
boolean invalid = false;
// Multiple Declaration Assignments
int x=2, y=4, w=2;
double p=2.2d, q = 2.3d;
double r = 2.2d, s = 3.4d;
int a=2, b=3, c=45;
char symbol1 = 'a', symbol2 = 'B', symbol3 = 'c';
String name = "my_name", name2 = "hisname", name__3 = "hername";
boolean valid = true, humidity = false, temperature = true;


// Uncomment to raise errors
//int  z = 2.2d;
//double w = 2;


// Uncomment to raise errors
//int x=2, y=true, w=2;
//double p=2.2d, q = 'A';
//double r = "somestring", s = 3.4d;
//double a=2, b=2, c=45;

}
```

Οι λάθος δηλώσεις μεταβλητών έχουν τοποθετηθεί σε σχόλια, τα οποία θα αφαιρεθούν στην συνέχεια. Ακολουθεί το σωστό παράδειγμα χρήσης:

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
   // Single Declaration Assignments
int x = 2;
double y = 2.2d;
char x = '\n';
char symbol2 = 'A';
String name = "my_name";
private boolean valid = true;
boolean invalid = false;
// Multiple Declaration Assignments
int x=2, y=4, w=2;
double p=2.2d, q = 2.3d;
double r = 2.2d, s = 3.4d;
int a=2, b=3, c=45;
char symbol1 = 'a', symbol2 = 'B', symbol3 = 'c';
String name = "my_name", name2 = "hisname", name__3 = "hername";
boolean valid = true, humidity = false, temperature = true;
// Uncomment to raise errors
//int  z = 2.2d;
//double w = 2;
// Uncomment to raise errors
//int x=2, y=true, w=2;
//double p=2.2d, q = 'A';
//double r = "somestring", s = 3.4d;
//double a=2, b=2, c=45;
}

Theo@DESKTOP-P1UVAKR ~/project
$
```

Τα παρακάτω αποτελούν παραδείγματα λανθασμένων δηλώσεων:

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
   // Single Declaration Assignments
int x = 2;
double y = 2.2d;
char x = '\n';
char symbol2 = 'A';
String name = "my_name";
private boolean valid = true;
boolean invalid = false;
// Multiple Declaration Assignments
int x=2, y=4, w=2;
double p=2.2d, q = 2.3d;
double r = 2.2d, s = 3.4d;
int a=2, b=3, c=45;
char symbol1 = 'a', symbol2 = 'B', symbol3 = 'c';
String name = "my_name", name2 = "hisname", name__3 = "hername";
boolean valid = true, humidity = false, temperature = true;
// Uncomment to raise errors
int  z = 2.2d;
//double w = 2;
// Uncomment to raise errors
//int x=2, y=true, w=2;
//double p=2.2d, q = 'A';
//double r = "somestring", s = 3.4d;
//double a=2, b=2, c=45;
}
Error: Type mismatch

Theo@DESKTOP-P1UVAKR ~/project
$
```

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
   // Single Declaration Assignments
int x = 2;
double y = 2.2d;
char x = '\n';
char symbol2 = 'A';
String name = "my_name";
private boolean valid = true;
boolean invalid = false;
// Multiple Declaration Assignments
int x=2, y=4, w=2;
double p=2.2d, q = 2.3d;
double r = 2.2d, s = 3.4d;
int a=2, b=3, c=45;
char symbol1 = 'a', symbol2 = 'B', symbol3 = 'c';
String name = "my_name", name2 = "hisname", name__3 = "hername";
boolean valid = true, humidity = false, temperature = true;
// Uncomment to raise errors
//int  z = 2.2d;
//double w = 2;
// Uncomment to raise errors
//int x=2, y=true, w=2;
double p=2.2d, q = 'A';
//double r = "somestring", s = 3.4d;
//double a=2, b=2, c=45;
}
Error: Expressions are not all of the same type.

Theo@DESKTOP-P1UVAKR ~/project
$
```

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
   // Single Declaration Assignments
int x = 2;
double y = 2.2d;
char x = '\n';
char symbol2 = 'A';
String name = "my_name";
private boolean valid = true;
boolean invalid = false;
// Multiple Declaration Assignments
int x=2, y=4, w=2;
double p=2.2d, q = 2.3d;
double r = 2.2d, s = 3.4d;
int a=2, b=3, c=45;
char symbol1 = 'a', symbol2 = 'B', symbol3 = 'c';
String name = "my_name", name2 = "hisname", name__3 = "hername";
boolean valid = true, humidity = false, temperature = true;
// Uncomment to raise errors
//int  z = 2.2d;
//double w = 2;
// Uncomment to raise errors
//int x=2, y=true, w=2;
//double p=2.2d, q = 'A';
//double r = "somestring", s = 3.4d;
double a=2, b=2, c=45;
}
Error: Type mismatch

Theo@DESKTOP-P1UVAKR ~/project
$
```

Μέρος Γ: Έλεγχος Εγκυρότητας

Ερώτημα Α: Έλεγχος σωστής δήλωσης μεταβλητών και μεθόδων

Για τον έλεγχο της σωστής χρήσης μεταβλητών, πρέπει να εξετάσουμε πως κάθε φορά που αναθέτουμε τιμή σε κάποια μεταβλητή, αυτή έχει οριστεί και η τιμή που της αναθέτουμε είναι σύμφωνη με τον τύπο της. Για το έλεγχο των μεθόδων, εξετάζουμε μόνο ότι όταν κάποια μέθοδος καλείται, αυτή έχει οριστεί προηγουμένως.

Αρχικά για τις μεθόδους, κάθε φορά που ολοκληρώνεται κάποιο method declaration και καλείται ο αντίστοιχος κανόνας του αρχείου bison, αποθηκεύεται το όνομα της μεθόδου στον πίνακα m_table με την χρήση της συνάρτησης **add_method**. Στην συνέχεια, όταν καλείται μία μέθοδος με τον κανόνα method_call, εξετάζουμε με την συνάρτηση **check_method** αν αυτή υπάρχει στον πίνακα m_table. Αν δεν υπάρχει, το πρόγραμμα τερματίζει και εμφανίζεται το αντίστοιχο ενημερωτικό μήνυμα.

Οι μεταβλητές από την άλλη, μοιράζονται την ίδια λογική με τις μεθόδους, δηλαδή αποθήκευση σε έναν πίνακα μαζί με τον τύπο τους και έλεγχος του πίνακα αυτού όταν γίνεται assignment. Η εισαγωγή των μεταβλητών με τον τύπο τους στον πίνακα declaredVar_table γίνεται με την κλήση της **add_declaredVar** στα πλαίσια του κανόνα variable_declaration. Ο έλεγχος γίνεται στον κανόνα assignment, όπου εξετάζεται αν η μεταβλητή στην οποία γίνεται ανάθεση τιμής έχει οριστεί, και αν η τιμή που της ανατίθεται είναι σύμφωνη με τον τύπο της (κλήση της **searchVariable** σε αυτή την περίπτωση για εύρεση του τύπου της μεταβλητής στον πίνακα declaredVar_table).

Ένα πρόβλημα που αντιμετωπίστηκε στην αποθήκευση των μεταβλητών, οφείλεται στο γεγονός πως η δήλωση των μεταβλητών αντίθετα από τις μεθόδους, εκτείνονται σε περισσότερους κανόνες από έναν και χρειάζεται να αποθηκεύονται μαζί με τον τύπο τους. Το πρόβλημα που εμφανίστηκε ήταν όταν υπήρχαν δηλώσεις τις μορφής int x,y,z;. Οι κανόνες που ενεργοποιούνται στην εξής περίπτωση είναι οι ακόλουθοι:

variable_declaration: c variable_modifier data_type variable extra_variables QMARK c

extra_variables: | COMMA variable extra_variables

Από την εντολή int x,y,z;, το κομμάτι ,y και ,z ενεργοποιούν τον κανόνα extra_variables ενώ το int x ;, τον κανόνα variable_declaration. Επειδή το αρχείο bison εκτελείται bottom-up, πρώτα καλείται ο κανόνας των extra_variables και μετά ο κανόνας variable_declaration. Όταν εκτελείται ο κανόνας extra_variables, δεν υπάρχει πρόσβαση στον τύπο των μεταβλητών και δεν μπορεί να γίνει κλήση της add_declaredVar. Για αυτόν τον λόγο, στον κανόνα extra_variables αποθηκεύονται οι μεταβλητές που διαβάζει ο κανόνας σε έναν ***πίνακα public_name***. Στην συνέχεια, όταν εκτελεστεί ο κανόνας variable_declaration αποθηκεύονται τα περιεχόμενα του πίνακα public_name μαζί με τον τύπο της μεταβλητής

που ενεργοποίησε τον κανόνα variable_declaration. Για παράδειγμα, έστω ότι υπάρχει η εντολή int x,y,z;

1. Οι y, και z, καλούν τον κανόνα extra_variables και αποθηκεύεται μόνο το όνομα τους στον πίνακα public_name.
2. Η υπόλοιπη εντολή, int x; , ενεργοποιεί τον κανόνα variable_declaration.
3. Αποθηκεύεται η μεταβλητή x μαζί με τον τύπο της στον πίνακα declaredVar_table με κλήση της **add_declaredVar**.
4. Αποθηκεύονται τα περιεχόμενα του public_name μαζί με τον τύπο της μεταβλητής x στον πίνακα declaredVar_table.

Για τις υπόλοιπες εντολές αρχικοποίησης μεταβλητών δεν αντιμετωπίστηκε κάποιο πρόβλημα. Οι int x; και int x = 2; δεν εμφανίζουν πρόβλημα γιατί δεν καλούν επιπλέον κανόνες ενώ αν υπάρχει εντολή της μορφής int x = 1, y = 2, z = 3; , τότε ο τύπος των μεταβλητών αποθηκεύεται από τις τιμές που ανατίθενται στις μεταβλητές.

Ακολουθεί ο κώδικας για την επίδειξη της σωστής λειτουργίας του ερωτήματος:

```
public class Germanios {

     int x = 1, y = 2, z = 3;

     char a, b, c;


     Private void main() {return 1;}


     Public void main2() {return 1;}


     Public void main3() {return 1;}
     public class C {


     Private void main4() {return 1;}
```

```
            Public void method() {

                  String w = "good morning";

                  double x = 1.0d;

                  x = 1.55d;


                  // x = 'a';
                  main();

                  // x = 1;
                  y = 2;
                  z = 3;
                  // z = "hello";
                  main2();

                  main3();

                  main4();

                  //main10();

                  w = "asd";

                  return 1;
      }

}

public class D {


      Public void method1() {

            boolean k = true;

            k = false;

            // k = 1;

            //e = 3;
            main();

            main4();
```

```
            //main10();

            return 1;
}

     public class E {Public void method3() {main();  return 1;}}

     }
}

public class B {


     Public void method2() {

            method();

            method1();

            main3();

            return 1;
     }

}
```

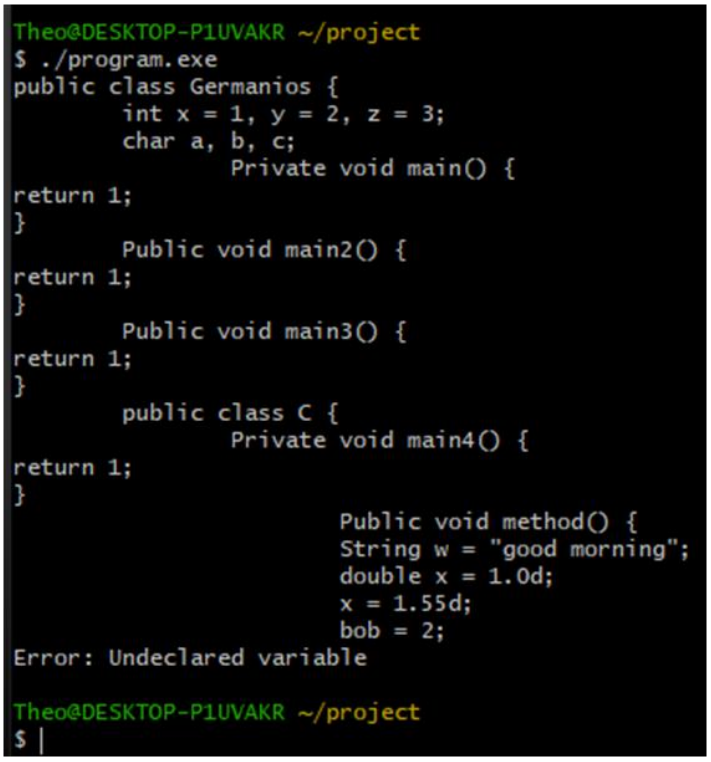
Στον παραπάνω κώδικα τηρούνται όλοι οι περιορισμοί του συγκεκριμένου ερωτήματος. Οι μέθοδοι που καλούνται έχουν οριστεί, ενώ οι μεταβλητές στις οποίες γίνεται ανάθεση τιμής υπάρχουν και είναι του ίδιου τύπου με της ανάθεσης. Σε σχόλια έχουν τοποθετηθεί τα λανθασμένα παραδείγματα χρήσης που θα ακολουθήσουν. Ακολουθεί το σωστό παράδειγμα χρήσης:

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
        int x = 1, y = 2, z = 3;
        char a, b, c;
                Private void main() {
return 1;
}
        Public void main2() {
return 1;
}
        Public void main3() {
return 1;
}
        public class C {
                Private void main4() {
return 1;
}
                        Public void method() {
                        String w = "good morning";
                        double x = 1.0d;
                        x = 1.55d;
                        //bob = 2;
                        // x = 'a';
                                        main();
                        // x = 1;
                        y = 2;
                        z = 3;
                        //z = "hello";
                                        main2();
                                                main3();
                        main4();
                        //main10();
                        w = "asd";
                                return 1;
                }
        }
                public class D {
                Public void method1() {
                        boolean k = true;
                        k = false;
                        // k = 1;
                        //e = 3;
                                        main();
                                        main4();
                        //main10();
                                                return 1;
                }
                        public class E {
Public void method3() {
main();
  return 1;
}
}
        }
}
public class B {
                Public void method2() {
                                        method();
                        method1();
                        main3();
                                                return 1;
                }
}

Theo@DESKTOP-P1UVAKR ~/project
$
```

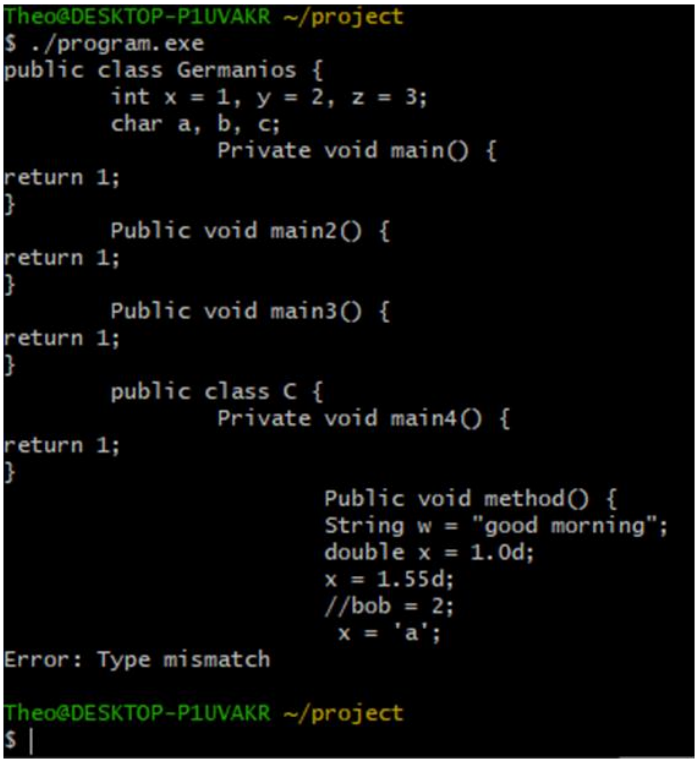
Θα ακολουθήσουν τρία παραδείγματα λανθασμένης χρήσης. Ένα όπου καλείται μία μέθοδος που δεν έχει οριστεί, ένα όπου γίνεται ανάθεση τιμής σε μεταβλητή που δεν έχει οριστεί και ένα όπου γίνεται λανθασμένη ανάθεση τιμής. Παρακάτω είναι το παράδειγμα όπου καλείται μία μη ορισμένη μέθοδος, η main10.

```
Theo@DESKTOP-P1UVAKR ~/project
$ nano file.txt

Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
public class Germanios {
        int x = 1, y = 2, z = 3;
        char a, b, c;
                Private void main() {
return 1;
}
        Public void main2() {
return 1;
}
        Public void main3() {
return 1;
}
        public class C {
                Private void main4() {
return 1;
}
                        Public void method() {
                        String w = "good morning";
                        double x = 1.0d;
                        x = 1.55d;
                        //bob = 2;
                        // x = 'a';
                                        main();
                        // x = 1;
                        y = 2;
                        z = 3;
                        //z = "hello";
                                        main2();
                                                main3();
                        main4();
Method 'main10' does not exist!
                        main10()
Theo@DESKTOP-P1UVAKR ~/project
$
```

Το επόμενο παράδειγμα θα είναι για την ανάθεση τιμής σε μία μεταβλητή που δεν έχει οριστεί (μεταβλητή bob).



Τέλος, θα γίνει μία λανθασμένη ανάθεση τιμής στην μεταβλητή x που είναι τύπου int.



Ερώτημα Β: Εμβέλεια μεταβλητών και μεθόδων

Στο συγκεκριμένο ερώτημα όπου απαιτείται ο έλεγχος της εμβέλειας των μεταβλητών και των μεθόδων, έχει εφαρμοστεί η ακόλουθη λογική. Όλες οι μεταβλητές και οι μέθοδοι που ορίζονται αποθηκεύονται me κλήση των συναρτήσεων **add_id** και **add_method** στους πίνακες *id_table* και *m_table* αντίστοιχα, μαζί με τα modifiers τους. Όταν καλείται μία μεταβλητή ή μέθοδος, γίνεται έλεγχος αν αυτή υπάρχει στους πίνακες αυτούς με την κλήση των **check_var_private** και **check_method**, ανεξαρτήτως των modifier τους. Έπειτα, κάθε φορά που ολοκληρώνεται μία κλάση πατέρας (όχι εμφωλευμένη), διαγράφονται οι μέθοδοι και μεταβλητές με modifier private, ώστε να μην είναι προσπελάσιμες από μετέπειτα κλάσεις. Οι public παραμένουν στους πίνακες και μπορούν να χρησιμοποιηθούν από επόμενες κλάσεις.

Η διαγραφή των private μεθόδων και μεταβλητών γίνεται μόνο από την κλάση πατέρα, με την βοήθεια της μεταβλητής method_counter που εξομοιώνει μία στοίβα. Πιο συγκεκριμένα, η μεταβλητή method_counter αυξάνεται κατά ένα όταν το αρχείο flex διαβάζει “public class”, δηλαδή όταν αρχικοποιείται μία κλάση, και μειώνεται κατά ένα όταν η κλάση αυτή ολοκληρώνεται και καλείται ο αντίστοιχος κανόνας του αρχείου bison public_class. Όταν η μεταβλητή γίνει ίση με 0, ή με άλλα λόγια η πατρική κλάση τελειώσει, τότε διαγράφονται όλες οι private μέθοδοι και μεταβλητές.

Η χρήση της μεταβλητής αυτή θα γίνει πιο κατανοητή με ένα παράδειγμα. Έστω ότι σαν είσοδο έχουμε το παρακάτω κομμάτι κώδικα:

```
1   public class Germanios {
2
3     public class D {
4
5     }
6
7     public class E {
8
9     }
10
11  }
12
13  public class B {
14
15  }
```

Αρχικά ο method_counter είναι 0.
Στην γραμμή 1 ο counter γίνεται 1 επειδή αρχικοποιείται μία κλάση.

Στην γραμμή 3 ο counter αυξάνεται πάλι κατά ένα και γίνεται 2.

Στην γραμμή 5 όπου τελειώνει η κλάση D, ο counter μειώνεται κατά 1 και είναι πλέον 1.

Στην γραμμή 7 ο counter αυξάνεται πάλι κατά 1 και γίνεται 2.

Στην γραμμή 9 ο counter μειώνεται κατά 1 και γίνεται 1.

Στην γραμμή 11 τελειώνει η αρχική κλάση και ο counter γίνεται 0. Επειδή ο counter έγινε 0, διαγράφονται όλες οι private μέθοδοι και μεταβλητές για να μην είναι προσπελάσιμες από τις επόμενες κλάσεις.

Για την εμβέλεια των μεθόδων δεν απαιτείται κάτι περισσότερο. Αν μία μέθοδος οριστεί ως private μέσα στην κλάση Germanios, τότε αυτή

δεν θα είναι προσπελάσιμη από την κλάση B που δεν ανήκει στο σώμα της παρά μόνο από τις εμφωλευμένες της. Οι public κλάσεις παραμένουν προσπελάσιμες από όλες τις κλάσεις.

Οι μεταβλητές ωστόσο, μπορούν να οριστούν μέσα σε μία κλάση καθώς και εντός μίας μεθόδου. Οι μεταβλητές που αρχικοποιούνται μέσα σε μία μέθοδο, δεν μπορούν να προσπελαθούν από οποιαδήποτε άλλη μέθοδο ή κλάση ενώ παράλληλα δεν έχουν κάποιο modifier. Για αυτόν τον λόγο, για τις μεταβλητές που ορίζονται εντός μιας μεθόδου, αποθηκεύεται ως modifier η λέξη “method”. Όταν τελειώσει μία μέθοδος, διαγράφονται όλες οι μεταβλητές από τον πίνακα id_table με modifier “method” και δεν είναι προσπελάσιμες από άλλες μεθόδους ή κλάσεις.

Για την διαφοροποίηση της αποθήκευσης μεταβλητών εντός μεθόδων και εκτός από αυτών, χρησιμοποιείται η μεταβλητή var_id_count. Η μεταβλητή αυτή δουλεύει όπως η μεταβλητή method_counter που περιγράφηκε παραπάνω. Όταν αρχικοποιείται μία μέθοδος αυξάνεται κατά ένα ενώ όταν τελειώνει μία μέθοδος μειώνεται κατά ένα. Όταν η μεταβλητή είναι 0 τότε οι μεταβλητές είναι εκτός μιας μεθόδου ενώ όταν είναι 1 είναι εντός.

Ακολουθούν δύο παραδείγματα χρήσης, ένα για το εύρος των μεθόδων και ένα για το εύρος των μεταβλητών. Όποιες γραμμές αρχικά βρίσκονται σε σχόλια, είναι αυτές που παραβιάζουν την εμβέλεια και θα ενεργοποιηθούν έπειτα.

Αρχικά, χρησιμοποιείται ο ακόλουθος κώδικας για την εμβέλεια των μεθόδων:

```
public class Germanios {

     Private void main() {return 1;}

     Public void main2() {return 1;}

     Public void main3() {return 1;}

     public class C {


          Private void main4() {return 1;}

          Public void method() {

               main();

               main2();

               main3();
```

```
                main4();

                return 1;
            }
        }
        public class D {

            Public void method1() {

                main();

                main4();

                return 1;
            }

            public class E {Public void method3() {main();  return 1;}}

        }

}

public class B {

        Public void method2() {

            method();

            method1();

            main3();

            //main();

            //main4();

            return 1;
        }
}
```

Οι μέθοδοι main2, main3, method, method1, method3 είναι public και προσπελάσιμες από όλες τις κλάσεις. Οι μέθοδοι main, main4 είναι private και δεν είναι προσπελάσιμες από την κλάση B που δεν ανήκει στην αρχική κλάση. Ακολουθεί το παράδειγμα σωστής χρήσης:

```
$ ./program.exe
//This file is for testing scopes of methods
public class Germanios {
        Private void main() {
return 1;
}
        Public void main2() {
return 1;
}
        Public void main3() {
return 1;
}
        public class C {
                Private void main4() {
return 1;
}
                        Public void method() {
                                        main();
                                        main2();
                        main3();
                        main4();
                        return 1;
                }
        }
                public class D {
                Public void method1() {
                                        main();
                                        main4();
                                                return 1;
                }
                        public class E {
Public void method3() {
main();
  return 1;
}
}
        }
}
public class B {
                Public void method2() {
                                        method();
                        method1();
                        main3();
                                        //main();
                                                //main4();
                                                return 1;
                }
}

Theo@DESKTOP-P1UVAKR ~/project
$
```

Με τον παραπάνω κώδικα δεν εμφανίζεται κάποιο σφάλμα επειδή οι κλήσεις των main, main4 στην κλάση Β έχουν τοποθετηθεί σε σχόλιο. Παρακάτω, θα αφαιρεθούν τα σχόλια στην κλήση της main της κλάσης Β.

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
//This file is for testing scopes of methods
public class Germanios {
        Private void main() {
return 1;
}
        Public void main2() {
return 1;
}
        Public void main3() {
return 1;
}
        public class C {
                Private void main4() {
return 1;
}
                        Public void method() {
                                        main();
                                        main2();
                        main3();
                        main4();
                        return 1;
                }
        }
                public class D {
                Public void method1() {
                                        main();
                                        main4();
                                                return 1;
                }
                        public class E {
Public void method3() {
main();
  return 1;
}
}
        }
}
public class B {
                Public void method2() {
                                        method();
                        method1();
                        main3();
Method 'main' does not exist!
                                        main()
Theo@DESKTOP-P1UVAKR ~/project
$
```

Σωστά, η εκτέλεση του προγράμματος σταματάει όταν γίνεται κλήση private μεθόδου από διαφορετική κλάση.

Ακολουθεί ο κώδικας για την επίδειξη της εμβέλειας των μεταβλητών:

```
public class Germanios {

private int x;

public int y,z;
```

```
int l = 5;
int o;
	Private void main() {
	int bob = 5;
	bob = 1;
	x = 1;
	y = 2;
	z = 3;
	l = 4;
	return 1;
	}

	Private void main10() {
	//bob = 10;
	return 1;
	}

	public class C {
		Private void main2() {
		x = 1;
		//bob = 5;
		y = 2;
		z = 3;
		l = 4;
		return 1;
		}
	}
}
```

```
public class B {
	Public void method2() {
	y = 5;
	l = 2;
	//x = 1;
	//bob = 5;
	return 1;
	}
}
```

Στο παραπάνω παράδειγμα, η μεταβλητή x είναι private, οι y, z, l, o είναι public ενώ η μεταβλητή bob ορίζεται μόνο στην μέθοδο main.

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
//This file is for testing scopes of variables
public class Germanios {
private int x;
public int y,z;
int l = 5;
int o;
        Private void main() {
                int bob = 5;
        bob = 1;
        x = 1;
        y = 2;
        z = 3;
        l = 4;
        return 1;
        }
        Private void main10() {
        //bob = 10;
        return 1;
        }
        public class C {
                Private void main2() {
                                                x = 1;
                //bob = 5;
                y = 2;
                z = 3;
                l = 4;
                                        return 1;
                }
        }
}
public class B {
                Public void method2() {
                                        y = 5;
                        l = 2;
                //x = 1;
                //bob = 5;
                                                return 1;
                }
}

Theo@DESKTOP-P1UVAKR ~/project
$
```

Αρχικά δεν παραβιάζεται κάποια εμβέλεια. Στην συνέχεια, θα γίνει προσπέλαση της μεταβλητής bob που ανήκει μόνο στην μέθοδο main από διαφορετικές μεθόδους:

```
Theo@DESKTOP-P1UVAKR ~/project
$ ./program.exe
//This file is for testing scopes of variables
public class Germanios {
private int x;
public int y,z;
int l = 5;
int o;
        Private void main() {
                int bob = 5;
        bob = 1;
        x = 1;
        y = 2;
        z = 3;
        l = 4;
        return 1;
        }
        Private void main10() {
        bob = 10;
Public 'bob' does not exist!

Theo@DESKTOP-P1UVAKR ~/project
$
```

```
$ ./program.exe
//This file is for testing scopes of variables
public class Germanios {
private int x;
public int y,z;
int l = 5;
int o;
        Private void main() {
                int bob = 5;
        bob = 1;
        x = 1;
        y = 2;
        z = 3;
        l = 4;
        return 1;
        }
        Private void main10() {
        //bob = 10;
        return 1;
        }
        public class C {
                Private void main2() {
                                        x = 1;
                bob = 5;
Public 'bob' does not exist!

Theo@DESKTOP-P1UVAKR ~/project
$
```

Τέλος, η κλάση B θα προσπαθήσει να προσπελάσει την μεταβλητή x που είναι private μεταβλητή διαφορετικής κλάσης:

```
$ ./program.exe
//This file is for testing scopes of variables
public class Germanios {
private int x;
public int y,z;
int l = 5;
int o;
        Private void main() {
                int bob = 5;
        bob = 1;
        x = 1;
        y = 2;
        z = 3;
        l = 4;
        return 1;
        }
        Private void main10() {
        //bob = 10;
        return 1;
        }
        public class C {
                Private void main2() {
                                        x = 1;
                //bob = 5;
                y = 2;
                z = 3;
                l = 4;
                                return 1;
                }
        }
}
public class B {
                Public void method2() {
                                y = 5;
                        l = 2;
                x = 1;
Public 'x' does not exist!

Theo@DESKTOP-P1UVAKR ~/project
$
```

Επιβεβαιώνονται όλα τα δυνατά σενάρια χρήσης.

Ερώτημα Γ: Υπολογισμός και ανάθεση αριθμητικής έκφρασης σε μεταβλητή

Προσθήκες στο αρχείο Flex

Για την υλοποίηση του συγκεκριμένου ερωτήματος κρίθηκαν απαραίτητες ορισμένες προσθήκες στον κώδικα του Flex αρχείου, χωρίς αλλαγές σε σχέση με τα προηγούμενα ερωτήματα:

```
// Variable to store yytext from INT_NUM and DOUBLE_NUM

char numval[50];


[0-9]+          { printf("%s", yytext); strcpy(numval, yytext); yylval.str = strdup("INT_NUM");  return INT_NUM;}

[0-9]+"."[0-9]+"d" { printf("%s", yytext); strcpy(numval, yytext); yylval.str = strdup("DOUBLE_NUM"); return DOUBLE_NUM; }
```

Παραπάνω φαίνεται η δημιουργία μίας μεταβλητής numval σε μορφή char array για την αποθήκευση της τιμής yytext. Κάθε φορά που αναγνωρίζεται από το Flex ένας ακέραιος αριθμός ή αριθμός διπλής ακρίβειας, με την εντολή strcpy(numval, yytext); αποθηκεύεται ο αριθμός αυτός στη μεταβλητή numval ώστε να χρησιμοποιηθεί αργότερα από τον parser. Προφανώς, στο bison αρχείο πρέπει να δηλώσουμε:

```
extern char numval[50];
```

Αλλαγές στο αρχείο Bison

Η υλοποίηση αυτού του ζητούμενου μπορεί να διακριθεί σε 3 μέρη:

- ✓ Υπολογισμός αριθμητικών εκφράσεων στο δεξιό μέρος της εντολής ανάθεσης με βάση την προτεραιότητα των πράξεων,
- ✓ Σωστός χειρισμός/προτεραιότητα παρενθέσεων,
- ✓ Έλεγχος, στην περίπτωση μεταβλητών, εάν έχουν λάβει αριθμητικές τιμές σε προηγούμενο χρόνο.

Για την υλοποίηση των 2 πρώτων, προσθέτουμε αρχικά τις ***ντιρεκτίβες***:

```
%left PLUS MINUS
%left MUL DIV
%nonassoc LEFT_BRACKET RIGHT_BRACKET
```

Οι παρενθέσεις δεν είναι συσχετισμένες, που σημαίνει ότι οι εκφράσεις μέσα σε παρενθέσεις αξιολογούνται πρώτες. Η πρόσθεση και η αφαίρεση έχουν την ίδια προτεραιότητα και είναι αριστερά συσχετισμένα. Ο πολλαπλασιασμός και η διαίρεση έχουν μεγαλύτερη προτεραιότητα από την πρόσθεση και την αφαίρεση και είναι επίσης αριστερά συσχετισμένα.

Ακολουθούν οι αλλαγές στον κανόνα <operations> :

```
operations: LEFT_BRACKET operations RIGHT_BRACKET    {$$=$2;}
| operations PLUS operations  {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) + strtod($3, &endptr2); sprintf($$, "%lf", temp);}
| operations MINUS operations {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) - strtod($3, &endptr2); sprintf($$, "%lf", temp);}
| operations MUL operations   {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) * strtod($3, &endptr2); sprintf($$, "%lf", temp);}
| operations DIV operations   {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) / strtod($3, &endptr2); sprintf($$, "%lf", temp);}
```

```
| num                    { strcpy($$, numval); }
| variable          { strcpy($$, findOperationValue($$)); }
;
```

Όλα τα tokens που συσχετίζονται με τους κανόνες <operations> και <num> είναι τύπου <str> (αλφαριθμητικά). Επομένως, για την εκτέλεση των πράξεων μεταξύ των τελεστών ακολουθείται η εξής διαδικασία:

A. Μετατροπή των num (είτε είναι INT_NUM είτε DOUBLE_NUM) από <str> σε double και εκτέλεση των πράξεων

```
double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) + strtod($3, &endptr2);
```

B. Ξανά μετατροπή σε <str> και αποθήκευση του αποτελέσματος στην $$

```
sprintf($$, "%lf", temp);
```

Στην περίπτωση των παρανθέσεων στον κανόνα <operations>, το $2 αναφέρεται στο αποτέλεσμα της ανάλυσης της εσωτερικής έκφρασης, ενώ το $$ αναφέρεται στο αποτέλεσμα της ολόκληρης πράξης. Έτσι, η εντολή `$$=$2;` επιβάλλει ότι όταν βρεθούν παρενθέσεις, το αποτέλεσμα της πράξης είναι το αποτέλεσμα της εσωτερικής έκφρασης χωρίς τις παρενθέσεις.

Το μόνο που μένει είναι η υλοποίηση του τρίτου μέρους που αφορά στις μεταβλητές.

```
typedef struct {
char name[256];
char value[50];
} Assign_record;
```

Assign_record **assign_table**[MAX_SIZE];

int ass_count = 0;

Οι παραπάνω γραμμές κώδικα δημιουργούν έναν πίνακα από εγγραφές με πεδία το όνομα μεταβλητής και την τιμή της. Κάθε φορά που θα συναντάται μία ανάθεση αριθμητικής τιμής τύπου INT_NUM ή DOUBLE_NUM σε μεταβλητή, μία εγγραφή θα προστίθεται στον πίνακα με την κλήση της συνάρτησης **add_assign** .

Για αυτό το σκοπό, στον κανόνα <assignment> κάνουμε την εξής προσθήκη στο action του υποκανόνα:

```
| VAR EQUAL normal_type QMARK
{
        if (strcmp($3, "INT_NUM") == 0 | strcmp($3, "DOUBLE_NUM") == 0){
add_assign($1, numval)
}
....
}
```

όπου numval η μεταβλητή που έχει αποθηκευμένη κάθε φορά το yytext των tokens INT_NUM , DOUBLE_NUM

Επιστρέφοντας στον κανόνα <operations> :

```
...
| variable          { strcpy($$, findOperationValue($$)); }
...
```

Με αυτή τη γραμμή κώδικα πραγματοποιούμε, με κλήση της συνάρτησης **findOperationValue**, μία **bottom up** αναζήτηση στον πίνακα assign_table με όρισμα την μεταβλητή variable ($$) και, εάν της έχει ανατεθεί τιμή σε προηγούμενο χρόνο, η τιμή αυτή ανατίθεται στην ίδια($$) ώστε να χρησιμοποιηθεί αργότερα στην εκτέλεση πράξεων.

Διαφορετικά η εκτέλεση του προγράμματος τερματίζει με αντίστοιχο μήνυμα σφάλματος:

Variable '%s' must have an assigned integer or double value!

Ακολουθεί ο κώδικας για την επίδειξη της σωστής λειτουργίας του ερωτήματος:

```
//This file is for testing operations
public class Germanios {

int temperature;
int l=5;
public double result1;
public double result2;
public double result3;
public double result4;
public double result5;
public double result6;

Private void main() {
int bob;
bob=7;
result1 = bob + l * 5 / 4;
result2 = 7 * (5 + 2) ;
result3 = (7 + 5) * 2 ;
result4 = bob * (l + 2);
result5 = (bob + l) * 2 ;
```

```
//Uncomment to raise errors
//result6 = (temperature + 1) * 2 ;

return 1;
}
}
```

Στον παραπάνω κώδικα τηρούνται όλοι οι περιορισμοί του συγκεκριμένου ερωτήματος. Οι μεταβλητές που έχουν χρησιμοποιηθεί στις αριθμητικές παραστάσεις έχουν λάβει τιμή σε προηγούμενο χρόνο. Σε σχόλια έχει τοποθετηθεί το λανθασμένο παράδειγμα χρήσης που θα ακολουθήσει.

Ακολουθεί το σωστό παράδειγμα χρήσης:

```
c:\DevSoft\Cygwin\home\mpamp>program file3c.txt
//This file is for testing operations
public class Germanios {
   int temperature;
 int l=5;
 public double result1;
 public double result2;
 public double result3;
 public double result4;
 public double result5;
 public double result6;
        Private void main() {
        int bob;
        bob=7;
                        result1 = bob + l * 5 / 4;
Value of result1 is:  13.250000
        result2 = 7 * (5 + 2) ;
Value of result2 is:  49.000000
               result3 = (7 + 5) * 2 ;
Value of result3 is:  24.000000
               result4 = bob * (l + 2);
Value of result4 is:  49.000000
        result5 = (bob + l) * 2 ;
Value of result5 is:  24.000000
    //Uncomment to raise errors
//result6 = (temperature + l) * 2 ;
        return 1;
        }
 }
```

Και το λανθασμένο όπου η μεταβλητή temperature χρησιμοποιείται χωρίς να της έχει ανατεθεί κάποια τιμή:

```
c:\DevSoft\Cygwin\home\mpamp>program file3c.txt
//This file is for testing operations
public class Germanios {
   int temperature;
 int l=5;
 public double result1;
 public double result2;
 public double result3;
 public double result4;
 public double result5;
 public double result6;
        Private void main() {
        int bob;
        bob=7;
                        result1 = bob + l * 5 / 4;
Value of result1 is:  13.250000
        result2 = 7 * (5 + 2) ;
Value of result2 is:  49.000000
               result3 = (7 + 5) * 2 ;
Value of result3 is:  24.000000
               result4 = bob * (l + 2);
Value of result4 is:  49.000000
        result5 = (bob + l) * 2 ;
Value of result5 is:  24.000000
    //Uncomment to raise errors
result6 = (temperature  Variable 'temperature' must have an assigned integer or double value!
```

Μέρος Δ: Error Recovery

Αρκεί να προσθέσω τον κανόνα:

| error '\n' {yyerrok;}

στον υπερκανόνα <program> του bison αρχείου.

ΕΞΗΓΗΣΗ:

Σε έναν υπερκανόνα, ο κανόνας error θα εκτελείται όταν οποιοσδήποτε από τους υποκανόνες που περιέχονται σε αυτόν τον υπερκανόνα αποτύχει να αναγνωρίσει την είσοδο. Έτσι, εξασφαλίζεται ότι η εκτέλεση του προγράμματος δεν θα διακόπτεται από ένα μη αναμενόμενο λάθος, αλλά αντίθετα θα συνεχίζει να αναγνωρίζει το λάθος και να εξετάζει την είσοδο για άλλες πιθανές ερμηνείες.

Το action block απλά εκτελεί την μακροεντολή yyerrok η οποία είναι predefined από την Bison και σηματοδοτεί την ολοκλήρωση του error recovery.

(Δες Παράγραφο 2.3 “Simple Error Recovery” σελ. 35 του Bison manual)

Ας πάρουμε το δοκιμαστικό αρχείο του προηγούμενου Ερωτήματος με τα operations και να προσθέσουμε την λανθασμένη συντακτικά γραμμή κώδικα:

double error = 2.2d

(χωρίς semicolon ;)

```
public class Germanios {
```

```
int temperature;
int l=5;

public double result1;
public double result2;
public double result3;
public double result4;
public double result5;
public double result6;

double error = 2.2d

Private void main() {
int bob;
bob=7;

result1 = bob + l * 5 / 4;
result2 = 7 * (5 + 2) ;
result3 = (7 + 5) * 2 ;
result4 = bob * (l + 2);
result5 = (bob + l) * 2 ;

      //Uncomment to raise errors
      result6 = (temperature + l) * 2 ;
return 1;
}
}
```

Αποτέλεσμα μετά την εκτέλεση:

```
C:\DevSoft\Cygwin\home\mpamp>program file3c.txt
//This file is for testing operations
public class Germanios {
   int temperature;
 int l=5;
 public double result1;
 public double result2;
 public double result3;
 public double result4;
 public double result5;
 public double result6;
  double error = 2.2d   PrivateError at line 25: syntax error
 void main() {
                int bob;
        bob=7;
                        result1 = bob + l * 5 / 4;
        result2 = 7 * (5 + 2) ;
                result3 = (7 + 5) * 2 ;
                result4 = bob * (l + 2);
        result5 = (bob + l) * 2 ;
    //Uncomment to raise errors
result6 = (temperature + l) * 2 ;
        return 1;
        }
 }

C:\DevSoft\Cygwin\home\mpamp>_
```

Παρατηρούμε ότι όντως το πρόγραμμα μας δεν τερματίζει λόγω του syntax error αλλά τυπώνει ενημερωτικό μήνυμα και συνεχίζει την εκτέλεση κανονικά.

Κώδικας

Bison file (parser.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 1000   // Maximum number of normal_types (adjust as needed)

extern int yylex();
extern FILE* yyin;
extern int line_number;
extern int method_count;
extern int var_id_count;
extern char numval[50];    // Variable to store yytext from INT_NUM and DOUBLE_NUM

int declaration_flag = 0;
int var_num = 0;




//Structs

typedef struct {            //Data structure to store variable name & expression type
   char name[256];                  // Assuming maximum variable name length of 255 characters
   char expr_type[20];              // Data type: INT_NUM, DOUBLE_NUM, CHAR_VAR, STRING_VAR,
BOOLEAN
} ExprEntry, DeclaredVar;
```

```
//Data structure for a method
typedef struct {
char method_name[256];
char method_type[256];
char method_modifier[256];
} M_call;


typedef struct {
char modifier[256];
char name[256];
} Var_check;


typedef struct {
char name[256];
} Extra_var;


typedef struct {
char name[256];
char value[50];
} Assign_record;



//Tables
ExprEntry data_table[MAX_SIZE];
int expr_count = 0;

DeclaredVar declaredVar_table[MAX_SIZE];
```

```
int var_count = 0;

char public_name[256][MAX_SIZE];

M_call m_table[MAX_SIZE];
int m_count = 0;

Var_check id_table[MAX_SIZE];
int id_count = 0;

Extra_var ex_table[MAX_SIZE];
int ex_count = 0;

Assign_record assign_table[MAX_SIZE];
int ass_count = 0;



//Function Signatures
void addVar(char *name, char *expr_type);
int compareAll(ExprEntry *data_table);
void clearTable(ExprEntry *data_table);
void add_method(char *modifier, char *type, char *name);
void check_method(const char *name);
void add_declaredVar(char *name, char *expr_type);
char *searchVariable(char *varName);
void add_id(char *modifier, char *name);
void add_extra(char *name);
void clear_extra();
void check_var_private(char *name);
void clear_var_private();
```

```
void clear_private_methods();
void add_assign(char *name, char *value);
char *findOperationValue(char *name);




// Function to add a normal_type to the normal_types table
void addVar(char *name, char *expr_type) {
    if (expr_count < MAX_SIZE) {
        strcpy(data_table[expr_count].name, name);
        strcpy(data_table[expr_count].expr_type, expr_type);
        expr_count++;
    } else {
        fprintf(stderr, "Error: Expressions table full\n");
        exit(EXIT_FAILURE);
    }
}



//Function to compare entries of normal_types table
int compareAll(ExprEntry *data_table) {



// Get the value of the first entry
char *firstValue = data_table[0].expr_type;

// Iterate over the remaining entries and compare their values
for (int i = 1; i < MAX_SIZE; i++) {
```

```
if (strcmp(data_table[i].expr_type, "") != 0){

// If any value is different from the first one, return 0
if (strcmp(firstValue, data_table[i].expr_type) != 0) {

return 0;
}
}
else{continue;}
}

// If all values are equal, return 1
return 1;

}



// Function to clear the normal_types table
void clearTable(ExprEntry *data_table){
 for (int i = 1; i < MAX_SIZE; i++) {
if (strcmp(data_table[i].expr_type, "") != 0){
 strcpy(data_table[i].name, "");
 strcpy(data_table[i].expr_type, "");
}
}
}
```

```
// Adds methods to the table
void add_method(char *modifier, char *type, char *name) {

if (m_count < MAX_SIZE) {
        strcpy(m_table[m_count].method_name, name);
strcpy(m_table[m_count].method_type, type);
strcpy(m_table[m_count].method_modifier, modifier);
        m_count++;
    } else {
        fprintf(stderr, "Error: Methods table full\n");
        exit(EXIT_FAILURE);
    }
}



// Checks whether the method called exists
void check_method(const char *name) {
    int flag = 0;

    for (int i = 0; i < MAX_SIZE; i++) {
        if (strcmp(m_table[i].method_name, name) == 0) {
            flag = 1;
            break;
        }
    }

    if (flag == 0) {
        fprintf(stderr, "Method '%s' does not exist!\n", name);
        exit(EXIT_FAILURE);
    }
```

```
}



// Function to add a variable to the declared variables table
void add_declaredVar(char *name, char *expr_type) {
    if (expr_count < MAX_SIZE) {
        strcpy(declaredVar_table[var_count].name, name);
        strcpy(declaredVar_table[var_count].expr_type, expr_type);
        var_count++;
    } else {
        fprintf(stderr, "Error: Declared variables' table full\n");
        exit(EXIT_FAILURE);
    }
}



// Function to search for a variable in the array bottom up
char *searchVariable(char *varName) {
    char foundVar[256] = "";
    int found = 0;

    // Start searching from the end of the array
    for (int i = var_count - 1; i >= 0; i--) {
        if (strcmp(varName, declaredVar_table[i].name) == 0) {
            strcpy(foundVar, declaredVar_table[i].name); // Store the found variable name
            found = 1;
            return declaredVar_table[i].expr_type;
        }
    }
```

```
    if(!found) {
return "";
    }
}




void add_id(char *modifier, char *name) {

if (id_count < MAX_SIZE) {
        strcpy(id_table[id_count].name, name);
strcpy(id_table[id_count].modifier, modifier);
        id_count++;
    } else {
        fprintf(stderr, "Error: table full\n");
        exit(EXIT_FAILURE);
    }
}




void add_extra(char *name) {

if (ex_count < MAX_SIZE) {
        strcpy(ex_table[ex_count].name, name);
        ex_count++;
    } else {
        fprintf(stderr, "Error: table full\n");
        exit(EXIT_FAILURE);
    }
}
```

```
void clear_extra() {

for(int i = 0; i < MAX_SIZE; i++){
strcpy(ex_table[ex_count].name, "");
}

ex_count = 0;
}




void check_var_private(char *name) {
   int flag = 0;

   for (int i = 0; i < MAX_SIZE; i++) {
      if (strcmp(id_table[i].name, name) == 0) {
         flag = 1;
         break;
      }
   }

   if (flag == 0) {
      fprintf(stderr, "Public '%s' does not exist!\n", name);
      exit(EXIT_FAILURE);
   }
}
```

```
void clear_var_private() {
   for (int i = 0; i < MAX_SIZE; i++) {
if (strcmp(id_table[i].modifier, "private") == 0)
{
      strcpy(id_table[i].name, "");
strcpy(id_table[i].modifier, "");
}
   }
}



// Removes the private methods from the table
void clear_private_methods() {
   for (int i = 0; i < MAX_SIZE; ++i) {
if (strcmp(m_table[i].method_modifier, "Private") == 0)
{
      strcpy(m_table[i].method_name, "");
strcpy(m_table[i].method_type, "");
strcpy(m_table[i].method_modifier, "");
}
   }
   //m_count = 0;
   //printf("Methods Table for private cleared.\n");
}



void add_assign(char *name, char *value) {
```

```
if (ass_count < MAX_SIZE) {
        strcpy(assign_table[ass_count].name, name);
strcpy(assign_table[ass_count].value, value);
        ass_count++;
    } else {
        fprintf(stderr, "Error: table full\n");
        exit(EXIT_FAILURE);
    }
}
```

```
// Function to search (bottom up) for the most recently assigned value (of type INT_NUM or DOUBLE_NUM) to a variable --> used in <opearations> rule
char *findOperationValue(char *name) {
    char foundValue[256] = "";
    int found = 0;

    // Start searching from the end of the array
    for (int i = ass_count - 1; i >= 0; i--) {
        if (strcmp(name, assign_table[i].name) == 0) {
            found = 1;
return assign_table[i].value;  // Store the found variable's name

        }
    }
    if(!found) {
fprintf(stderr, "  Variable '%s' must have an assigned integer or double value!\n", name);
exit(EXIT_FAILURE);
    }
}
```

```
void yyerror(char *s) {
  fprintf(stderr, "Error at line %d: %s\n", line_number, s);
}


%}



%union {
char *str;
}

%token PUBLIC_CLASS CLASS_NAME LEFT_BRACE RIGHT_BRACE QMARK
%token <str> PUBLIC PRIVATE
%token <str> VAR
%token <str> INT DOUBLE CHAR BOOLEAN STRING
%token <str> M_PUBLIC M_PRIVATE VOID
%token <str> INT_NUM DOUBLE_NUM CHAR_VAR STRING_VAR TRUE FALSE
%token DOT LEFT_BRACKET RIGHT_BRACKET COMMA  MUL DIV PLUS MINUS
%token EQUAL NEW
%token RETURN
%token DO WHILE AND OR CHECK_EQUAL LESS GREATER NOT_EQUAL FOR SWITCH CASE
%token DOUBLE_DOT DEFAULT BREAK PRINT IF ELSE ELSE_IF
%token SINGLE_COMMENT MULTILINE_COMMENT

%left PLUS MINUS
%left MUL DIV
```

```
%nonassoc LEFT_BRACKET RIGHT_BRACKET



%type <str> variable

%type <str> expr_type

%type <str> normal_type                 //only normal_type assignments with same line variable declarations

%type <str> method_modifier

%type <str> return_type

%type <str> variable_modifier

%type <str> operations

%type <str> num




%%

program: c public_class c

            | program c public_class c

            ;


public_class: PUBLIC_CLASS CLASS_NAME LEFT_BRACE class_block RIGHT_BRACE


{

method_count = method_count - 1;


if (method_count == 0) {


// printf("\n original method \n");


clear_private_methods();

clear_var_private();
```

```
}
}
                    ;

class_block: variable_declarations method_declarations nested_class;
 nested_class: public_class | nested_class c public_class | ;

variable_declarations: variable_declaration | variable_declaration variable_declarations | ;

variable_declaration:   c variable_modifier expr_type variable extra_variables QMARK c
{
char temp_type[20];
strcpy(temp_type, "");

if (strcmp($3, "int") == 0) {
                                    strcpy(temp_type, "INT_NUM");
} else if (strcmp($3, "double") == 0) {
  strcpy(temp_type, "DOUBLE_NUM");
} else if (strcmp($3, "boolean") == 0) {
  strcpy(temp_type, "BOOLEAN");
} else if (strcmp($3, "String") == 0) {
  strcpy(temp_type, "STRING_VAR");
} else if (strcmp($3, "char") == 0) {
                                     strcpy(temp_type, "CHAR_VAR");
} else {
  strcpy(temp_type, "");
}

if(strcmp(temp_type, "") == 0) {
 fprintf(stderr, "Error: Unknown type\n");
```

```
 exit(EXIT_FAILURE);
} else {
add_declaredVar($4, temp_type); }

if(declaration_flag == 1) {
 for(int i = 0; i < var_num; i++) {
  add_declaredVar(public_name[i], temp_type); } }

var_num = 0;
declaration_flag = 0;



if(var_id_count == 0) {
add_id($2, $4);

for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id($2, ex_table[i].name);
}
}

}



if(var_id_count > 0) {
add_id("method", $4);

for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id("method", ex_table[i].name);
}
```

```
}

}

clear_extra();

}

| c CLASS_NAME variable EQUAL object_instance QMARK c
| c variable_modifier expr_type variable EQUAL normal_type extra_assigned_variables QMARK c
{
 if (strcmp($6, "INT_NUM") == 0 | strcmp($6, "DOUBLE_NUM") == 0){

 add_assign($4, numval);

 }


// Add the first variable and its expression to the symbol table
addVar($4, $6);
add_declaredVar($4, $6);


if(var_id_count == 0) {
add_id($2, $4);

for (int b = 0; b < 100; b++){
if (strcmp(ex_table[b].name, "") != 0) {
add_id($2, ex_table[b].name);
}
}
```

```
}

if(var_id_count > 0) {
add_id("method", $4);

for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id("method", ex_table[i].name);
}
}

}

clear_extra();


}
c
{

if (!compareAll(data_table)) {
printf("Error: Expressions are not all of the same type.\n");
exit(EXIT_FAILURE);
}
else {
```

```
expr_count=0;
clearTable(data_table);
}


}
c
{



                    if(strcmp($3, "int") == 0) {
if(strcmp($6, "INT_NUM") != 0) {
                                                                    fprintf(stderr,
"Error: Type mismatch\n");
exit(EXIT_FAILURE);
}
}
else if(strcmp($3, "double") == 0)
{
if(strcmp($6, "DOUBLE_NUM") != 0)  {
fprintf(stderr, "Error: Type mismatch\n");
exit(EXIT_FAILURE);
}
}
else if(strcmp($3, "char") == 0)
{
if(strcmp($6, "CHAR_VAR") != 0)  {
fprintf(stderr, "Error: Type mismatch\n");
        exit(EXIT_FAILURE);
}
}
else if(strcmp($3, "boolean") == 0)
{
```

```
if(strcmp($6, "BOOLEAN") != 0)  {
                                                fprintf(stderr, "Error: Type mismatch\n");
                                                exit(EXIT_FAILURE);
}
}
else if(strcmp($3, "String") == 0)
{
if(strcmp($6, "STRING_VAR") != 0)   {
fprintf(stderr, "Error: Type mismatch\n");
exit(EXIT_FAILURE);
}
}

}

| c expr_type variable extra_variables QMARK c
{
char temp_type[20];
strcpy(temp_type, "");

if (strcmp($2, "int") == 0) {
                                    strcpy(temp_type, "INT_NUM");
} else if (strcmp($2, "double") == 0) {
  strcpy(temp_type, "DOUBLE_NUM");
} else if (strcmp($2, "boolean") == 0) {
  strcpy(temp_type, "BOOLEAN");
} else if (strcmp($2, "String") == 0) {
  strcpy(temp_type, "STRING_VAR");
} else if (strcmp($2, "char") == 0) {
                                     strcpy(temp_type, "CHAR_VAR");
} else {
```

```
   strcpy(temp_type, "");
}

if(strcmp(temp_type, "") == 0) {
 fprintf(stderr, "Error: Unknown type\n");
 exit(EXIT_FAILURE);
} else {
add_declaredVar($3, temp_type); }

if(declaration_flag == 1) {
 for(int i = 0; i < var_num; i++) {
  add_declaredVar(public_name[i], temp_type); } }

var_num = 0;
declaration_flag = 0;



if(var_id_count == 0) {
add_id("public", $3);

for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id("public", ex_table[i].name);
}
}

}



if(var_id_count > 0) {
add_id("method", $3);
```

```
for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id("method", ex_table[i].name);
}
}

}
                                        clear_extra();
 }
| c expr_type variable EQUAL normal_type extra_assigned_variables QMARK c
{

 if (strcmp($5, "INT_NUM") == 0 | strcmp($5, "DOUBLE_NUM") == 0){

add_assign($3, numval);

}

// Add the first variable and its expression to the symbol table
addVar($3, $5);
add_declaredVar($3, $5);


if(var_id_count == 0) {
add_id("public", $3);

for (int b = 0; b < 100; b++){
if (strcmp(ex_table[b].name, "") != 0) {
add_id("public", ex_table[b].name);
}
```

```
}

}


if(var_id_count > 0) {
add_id("method", $3);

for (int i = 0; i < 100; i++){
if (strcmp(ex_table[i].name, "") != 0) {
add_id("method", ex_table[i].name);
}
}

}

clear_extra();



}
c
{

if (!compareAll(data_table)) {
printf("Error: Expressions are not all of the same type.\n");
exit(EXIT_FAILURE);
}
else {

expr_count=0;
clearTable(data_table);
```

```
}
}
c
{

if(strcmp($2, "int") == 0) {
if(strcmp($5, "INT_NUM") != 0) {
fprintf(stderr, "Error: Type mismatch\n");
        exit(EXIT_FAILURE);
}
}
else if(strcmp($2, "double") == 0) {
if(strcmp($5, "DOUBLE_NUM") != 0) {

                        fprintf(stderr, "Error: Type mismatch\n");
                        exit(EXIT_FAILURE);
}
}
else if(strcmp($2, "char") == 0) {
if(strcmp($5, "CHAR_VAR") != 0) {
fprintf(stderr, "Error: Type mismatch\n");
        exit(EXIT_FAILURE);
}
}
else if(strcmp($2, "boolean") == 0) {
if(strcmp($5, "BOOLEAN") != 0) {
fprintf(stderr, "Error: Type mismatch\n");
        exit(EXIT_FAILURE);
}
}
else if(strcmp($2, "String") == 0) {
if(strcmp($5, "STRING_VAR") != 0) {
fprintf(stderr, "Error: Type mismatch\n");
```

```
exit(EXIT_FAILURE);
}
}

}
;


extra_variables: | COMMA variable extra_variables
{
declaration_flag = 1;
strcpy(public_name[var_num], $2);
var_num++;

add_extra($2);

};


extra_assigned_variables: | COMMA variable EQUAL normal_type extra_assigned_variables
                    {

if (strcmp($4, "INT_NUM") == 0 | strcmp($4, "DOUBLE_NUM") == 0){

add_assign($2, numval);

}


// Add extra variables and their assigned expressions to data_table
addVar($2, $4);
```

```
add_declaredVar($2, $4);


add_extra($2);


                    };



method_declarations: c method_declaration c | c method_declaration c method_declarations | ;



method_declaration: method_modifier return_type variable LEFT_BRACKET parameters
RIGHT_BRACKET LEFT_BRACE method_block return RIGHT_BRACE
{
add_method($1, $2, $3);
var_id_count = var_id_count - 1;



for (int i = 0; i < 100; i++) {
if (strcmp(id_table[i].modifier, "method") == 0)
{
                    strcpy(id_table[i].name, "");
strcpy(id_table[i].modifier, "");
}
          }


}
;



parameters: expr_type variable
```

```
| expr_type variable COMMA parameters
|;


method_block: c variable_declarations c commands c ;


commands: c command c | c command c commands | ;


command: assignment | do_loop | for_loop | switch_case | break | method_call QMARK | print |
if_else ;


assignment: VAR EQUAL expression QMARK
   | VAR EQUAL operations QMARK
  {
// Print result of operation as a double
char *eptr;
printf("Value of %s is:  %lf \n", $1, strtod($3, &eptr));
  }

  | VAR EQUAL normal_type QMARK
  {
       if (strcmp($3, "INT_NUM") == 0 | strcmp($3, "DOUBLE_NUM") == 0){

add_assign($1, numval);

}
```

```
check_var_private($1);

char* foundVariableType = searchVariable($1);
if(strcmp(foundVariableType, "") == 0)
{
fprintf(stderr, "Error: Undeclared variable\n");
exit(EXIT_FAILURE);
}
else
{
if(strcmp(foundVariableType, $3) != 0)
{
fprintf(stderr, "Error: Type mismatch\n");
exit(EXIT_FAILURE);
}
}
};


member_access: variable DOT variable LEFT_BRACKET arguments RIGHT_BRACKET
| variable DOT variable ;


method_call: variable LEFT_BRACKET arguments RIGHT_BRACKET

{
check_method($1);
}
;
```

arguments: variable | variable COMMA arguments | normal_type | normal_type COMMA arguments | ;

expression: member_access | method_call ;

//operations and num rules are of type <str>. To perform operations i convert string values to double and then convert back to string again to store the result in $$

operations: LEFT_BRACKET operations RIGHT_BRACKET {$$=$2;}

| operations PLUS operations {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) + strtod($3, &endptr2); sprintf($$, "%lf", temp);}

| operations MINUS operations {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) - strtod($3, &endptr2); sprintf($$, "%lf", temp);}

| operations MUL operations {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) * strtod($3, &endptr2); sprintf($$, "%lf", temp);}

| operations DIV operations {double temp; char *endptr1; char *endptr2; temp = strtod($1, &endptr1) / strtod($3, &endptr2); sprintf($$, "%lf", temp);}

| num { strcpy($$, numval); }

| variable { strcpy($$, findOperationValue($$)); }

;

do_loop: DO c LEFT_BRACE commands RIGHT_BRACE c WHILE LEFT_BRACKET conditions RIGHT_BRACKET QMARK ;

conditions: condition | condition logic_operators conditions ;

condition: variable compare_symbol normal_type

| normal_type compare_symbol variable

| variable compare_symbol VAR

| normal_type compare_symbol normal_type ;

logic_operators: AND | OR ;

compare_symbol: LESS | GREATER | NOT_EQUAL | CHECK_EQUAL ;

operator_symbol: DIV | MUL | PLUS | MINUS ;

for_loop: FOR LEFT_BRACKET for_assignment QMARK conditions QMARK increment RIGHT_BRACKET c LEFT_BRACE commands RIGHT_BRACE c ;

for_assignment: variable EQUAL num | variable EQUAL variable ;

increment: variable EQUAL variable operator_symbol num

| variable EQUAL variable operator_symbol variable ;

switch_case: SWITCH LEFT_BRACKET variable RIGHT_BRACKET c LEFT_BRACE c cases c DEFAULT commands RIGHT_BRACE

| SWITCH LEFT_BRACKET variable RIGHT_BRACKET c LEFT_BRACE c cases c RIGHT_BRACE ;

cases: CASE normal_type DOUBLE_DOT commands | CASE normal_type DOUBLE_DOT commands cases ;

if_else: c IF LEFT_BRACKET conditions RIGHT_BRACKET c LEFT_BRACE commands RIGHT_BRACE c else_if c else c ;

else_if: c ELSE_IF LEFT_BRACKET conditions RIGHT_BRACKET c LEFT_BRACE commands RIGHT_BRACE c

|c ELSE_IF LEFT_BRACKET conditions RIGHT_BRACKET c LEFT_BRACE commands RIGHT_BRACE c else_if c

| ;

else: c ELSE c LEFT_BRACE commands RIGHT_BRACE | ;

```
normal_type: DOUBLE_NUM | INT_NUM | TRUE | FALSE | CHAR_VAR | STRING_VAR ;

object_instance: NEW CLASS_NAME LEFT_BRACKET RIGHT_BRACKET ;

return: RETURN normal_type QMARK
|RETURN expression QMARK
| ;

break: BREAK QMARK ;

 print: PRINT LEFT_BRACKET STRING_VAR print_vars RIGHT_BRACKET QMARK ;


print_vars: COMMA variable | COMMA variable print_vars | ;

 variable_modifier: PUBLIC | PRIVATE ;

return_type: INT | DOUBLE | CHAR | BOOLEAN | STRING | VOID ;

expr_type: INT | DOUBLE | CHAR | BOOLEAN | STRING ;

method_modifier: M_PUBLIC | M_PRIVATE ;

variable: VAR ;

num: INT_NUM | DOUBLE_NUM ;

c: comment | comment c | ;

comment: SINGLE_COMMENT | MULTILINE_COMMENT ;
```

```
%%



int main(int argc, char** argv) {
   char filename[] = "file.txt";
   if(argc == 2)
   {
strcpy(filename, argv[1]);
   }
   FILE *file = fopen(filename, "r");
   if (!file) {
      perror(filename);
      return 1;
   }
   yyin = file;
   yyparse();
   fclose(file);
   return 0;
}
```

Flex file (lexer.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.tab.h"
```

```
%}

%option noyywrap

%{
int line_number = 1;      // Variable to store line number
int method_count = 0;
int var_id_count = 0;

char numval[50];        // Variable to store yytext from INT_NUM and DOUBLE_NUM
%}




%%

[\n]    { line_number++; }   // Update line number on newline

"public class"  { printf("%s", yytext); method_count++; return PUBLIC_CLASS; }
"int" { printf("%s", yytext); yylval.str = strdup(yytext); return INT; }
"double" { printf("%s", yytext); yylval.str = strdup(yytext); return DOUBLE; }
"char" { printf("%s", yytext); yylval.str = strdup(yytext); return CHAR; }
"boolean" { printf("%s", yytext); yylval.str = strdup(yytext); return BOOLEAN; }
"String" { printf("%s", yytext); yylval.str = strdup(yytext); return STRING; }
"private" { printf("%s", yytext); yylval.str = strdup(yytext); return PRIVATE; }
"public" { printf("%s", yytext); yylval.str = strdup(yytext); return PUBLIC; }
"Private" { printf("%s", yytext); var_id_count++; yylval.str = strdup(yytext); return M_PRIVATE; }
"Public" { printf("%s", yytext); var_id_count++; yylval.str = strdup(yytext); return M_PUBLIC; }
```

```
"void" { printf("%s", yytext); yylval.str = strdup(yytext); return VOID; }
"return" { printf("%s", yytext); return RETURN; }
"true" { printf("%s", yytext); yylval.str = strdup("BOOLEAN"); return TRUE; }
"false" { printf("%s", yytext); yylval.str = strdup("BOOLEAN"); return FALSE; }
"do" { printf("%s", yytext); return DO; }
"while" { printf("%s", yytext); return WHILE; }
"for" { printf("%s", yytext); return FOR; }
"switch" { printf("%s", yytext); return SWITCH; }
"case" { printf("%s", yytext); return CASE; }
"if" { printf("%s", yytext); return IF; }
"else if" { printf("%s", yytext); return ELSE_IF; }
"else" { printf("%s", yytext); return ELSE; }
"break" { printf("%s", yytext); return BREAK; }
":" { printf("%s", yytext); return DOUBLE_DOT; }
"default:" { printf("%s", yytext); return DEFAULT; }
"out\.print" { printf("%s", yytext); return PRINT; }
"==" { printf("%s", yytext); return CHECK_EQUAL; }
"=" { printf("%s", yytext); return EQUAL; }
"new" { printf("%s", yytext); return NEW; }
"(" { printf("%s", yytext); return LEFT_BRACKET; }
")" { printf("%s", yytext); return RIGHT_BRACKET; }
"&&" { printf("%s", yytext); return AND; }
"||" { printf("%s", yytext); return OR; }
"<" { printf("%s", yytext); return LESS; }
">" { printf("%s", yytext); return GREATER; }
"!=" { printf("%s", yytext); return NOT_EQUAL; }
"/" { printf("%s", yytext); return DIV; }
"\*" { printf("%s", yytext); return MUL; }
"+" { printf("%s", yytext); return PLUS; }
"-" { printf("%s", yytext); return MINUS; }
"\." { printf("%s", yytext); return DOT; }
```

```
\'[^\']\' { printf("%s", yytext); yylval.str = strdup("CHAR_VAR"); return CHAR_VAR; }

\'\\n\' { printf("%s", yytext); yylval.str = strdup("CHAR_VAR"); return CHAR_VAR; }

\"[^\"]*\" { printf("%s", yytext); yylval.str = strdup("STRING_VAR"); return STRING_VAR; }

[A-Z_][a-zA-Z0-9_]*   { printf("%s", yytext); return CLASS_NAME; }

[a-zA-Z_][a-zA-Z0-9_]*   { printf("%s", yytext); yylval.str = strdup(yytext); return VAR; }

[0-9]+            { printf("%s", yytext); strcpy(numval, yytext); yylval.str = strdup("INT_NUM");  return INT_NUM;}

[0-9]+"."[0-9]+"d" { printf("%s", yytext); strcpy(numval, yytext); yylval.str = strdup("DOUBLE_NUM"); return DOUBLE_NUM; }

"{"   { printf("%s\n", yytext); return LEFT_BRACE; }

","   { printf("%s", yytext); return COMMA; }

"}"   { printf("%s\n", yytext); return RIGHT_BRACE; }

";"   { printf("%s\n", yytext); return QMARK; }

"\/\/".*    { printf("%s\n", yytext); return SINGLE_COMMENT; }

"\/\*"[^\*]*\*+([^\*\/][^\*]*\*+)*"\/"    { printf("%s\n", yytext); return MULTILINE_COMMENT; }

.   { printf("%s", yytext); }



%%
```