

ΜΥΕ020 – ΜΕΤΑΦΡΑΣΤΕΣ 2

“Προγραμματιστική άσκηση: Μετάφραση από αντικειμενοστραφή γλώσσα
προγραμματισμού σε γλώσσα διαδικασιακού προγραμματισμού”

Γκαβαρδίνας Όθωνας, ΑΜ: 2620

Μπουρλή Στυλιανή, ΑΜ: 2774

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ	3
1.1 Ο ορισμός του μεταγλωττιστή.	3
1.2 Οι απαιτήσεις από ένα μεταγλωττιστή.	3
1.3 Οι φάσεις της μεταγλώττισης.	4
1.4 Χαρακτηριστικά αντικειμενοστρεφών γλωσσών.	5
2. Η ΓΛΩΣΣΑ ΜΑΣ	8
2.1 Η περιγραφή της γλώσσας μας	8
2.2 Η γραμματική της γλώσσας μας.	12
3. ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ	23
3.1 Ο ορισμός και η λειτουργία του λεκτικού αναλυτή.	23
3.2 Το εργαλείο flex.	24
3.3 Η λειτουργία του flex στη γλώσσα μας.	29
4. ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ	32
4.1 Ο ορισμός και η λειτουργία του συντακτικού αναλυτή.	32
4.2 Το εργαλείο bison.	33
4.3 Η λειτουργία του bison στη γλώσσα μας.	35
5. ΠΑΡΑΓΩΓΗ ΚΩΔΙΚΑ ΣΕ ΓΛΩΣΣΑ C	35
5.1 Προσθήκες στον κώδικα του flex.	35
5.2 Προσθήκες στον κώδικα του bison.	36
6. ΤΕΣΤ	49

1. ΕΙΣΑΓΩΓΗ

1.1 Ο ορισμός του μεταγλωττιστή

Μεταγλωττιστής ή μεταφραστής (compiler) ονομάζεται ένα πρόγραμμα υπολογιστή, το οποίο διαβάζει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού (αρχικό πρόγραμμα) και τον μεταφράζει σε ισοδύναμο κώδικα σε μια άλλη γλώσσα προγραμματισμού (τελικό πρόγραμμα). Επιπλέον εμφανίζει διαγνωστικά μηνύματα, συνήθως μηνύματα λάθους, μερικές φορές όμως και μηνύματα προειδοποίησης.

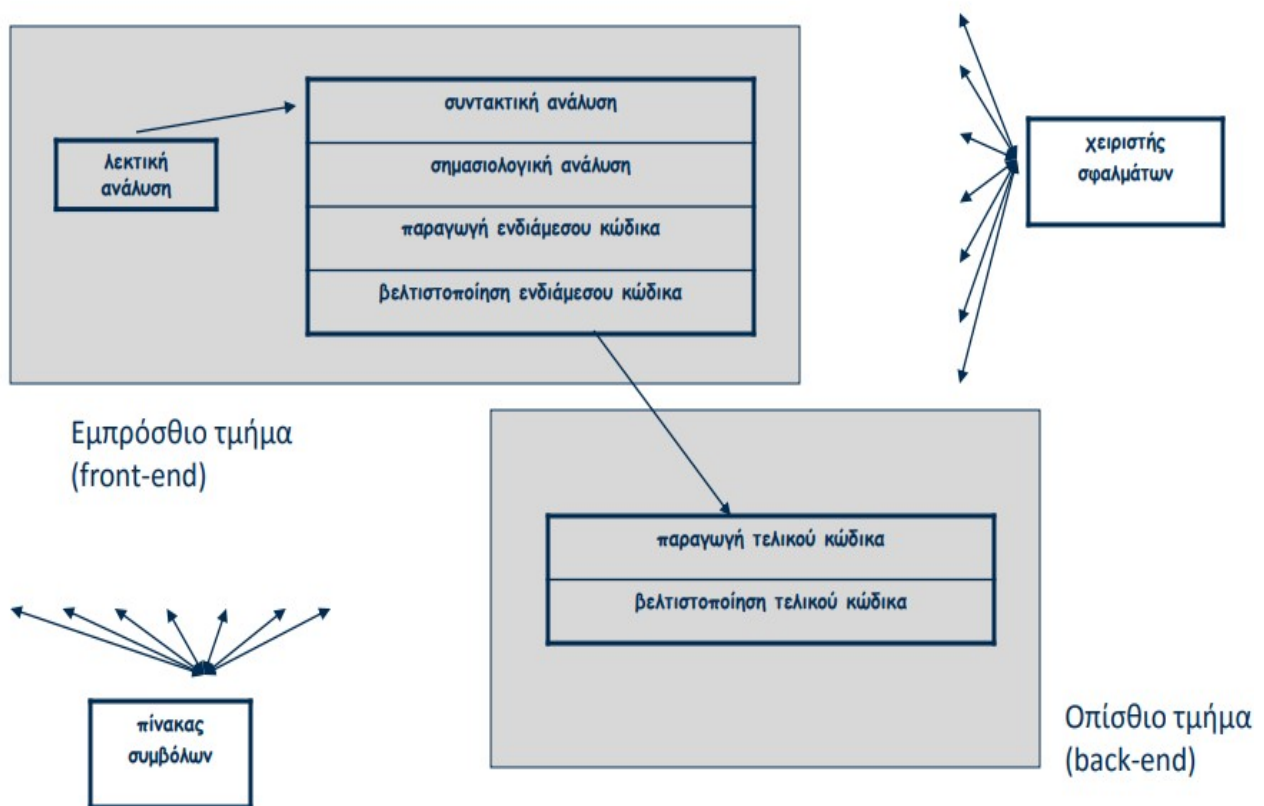
1.2 Οι απαιτήσεις από ένα μεταγλωττιστή

Οι βασικές απαιτήσεις που θα πρέπει να ικανοποιεί ένας μεταγλωττιστής είναι οι εξής:

- ✓ Σωστή λειτουργία
- ✓ Συμμόρφωση με προδιαγραφές αρχικής και τελικής γλώσσας
- ✓ Μετάφραση προγραμμάτων αυθαίρετα μεγάλου μήκους
- ✓ Παραγωγή αποδοτικού κώδικα
- ✓ Μικρός χρόνος εκτέλεσης
- ✓ Μικρές απαιτήσεις μνήμης κατά τη μεταγλώττισης
- ✓ Καλά διαγνωστικά μηνύματα
- ✓ Δυνατότητα συνέχισης ύστερα από εντοπισμό σφαλμάτων
- ✓ Μεταφερσιμότητα

1.3 Οι φάσεις της μεταγλώττισης

Η μεταγλώττιση ενός προγράμματος, γενικά, μπορεί να διαχωριστεί στις εξής φάσεις: λεκτική ανάλυση, συντακτική ανάλυση, σημασιολογική ανάλυση, παραγωγή ενδιάμεσου κώδικα, βελτιστοποίηση ενδιάμεσου κώδικα, παραγωγή τελικού κώδικα και βελτιστοποίηση τελικού κώδικα. Στη συγκεκριμένη άσκηση δεν υλοποιήθηκαν όλες οι φάσεις της μεταγλώττισης. Συγκεκριμένα, τα βήματα που ακολουθήθηκαν ήταν: λεκτική ανάλυση, συντακτική ανάλυση και παραγωγή τελικού κώδικα σε γλώσσα C.



“Συνήθης Οργάνωση Μεταγλωττιστή”

1.4 Χαρακτηριστικά αντικειμενοστρεφών γλωσσών

Τα βασικά χαρακτηριστικά μιας αντικειμενοστραφούς γλώσσας είναι:

- **Αντικείμενα (objects)**
 - ◆ Δεδομένα, Λειτουργίες
 - ◆ Διαφέρουν από τις εγγραφές καθώς οι εγγραφές περιέχουν μόνο δεδομένα ενώ τα αντικείμενα περιέχουν και μεθόδους
 - ◆ Η παράσταση ενός αντικειμένου στη μνήμη δεν διαφέρει από αυτή μιας εγγραφής αφού απαιτείται χώρος μόνο για τα δεδομένα
- **Κελυφοποίηση (encapsulation)**
 - ◆ Αφαιρετική από κοινού αντιμετώπιση της κατάστασης και τις συμπεριφοράς των αντικειμένων
- **Κλάσεις (classes)**
 - ◆ Αντικείμενα με την ίδια συμπεριφορά
 - ◆ Αντικείμενα της ίδιας κλάσης μπορεί να διαφέρουν μόνο κατά την κατάσταση
- **Μέθοδοι (methods)**
 - ◆ Οι διαδικασίες και οι συναρτήσεις που εφαρμόζονται στα δεδομένα ενός αντικειμένου και αφορούν συγκεκριμένη εμφάνιση ενός αντικειμένου
- **Κληρονομικότητα (inheritance)**
 - ◆ Μία κλάση μπορεί να οριστεί σαν επέκταση μιας άλλης κλάσης αποκτώντας τις μεθόδους και ιδιότητες της γονικής τους κλάσης προσθέτοντας και δικές της
 - ◆ Ιεραρχίες κλάσεων για επαναχρησιμοποίηση κώδικα
 - ◆ Στην απλή κληρονομικότητα κάθε κλάση μπορεί να χρησιμοποιήσει τα χαρακτηριστικά μιας άλλης κλάσης, όχι όμως περισσότερων από μιας
 - ◆ Η κλάση που κληρονομεί τα χαρακτηριστικά λέγεται παραγόμενη κλάση (derived class) ή υποκλάση (subclass)
 - ◆ Η κλάση που κληροδοτεί λέγεται υπερκλάση (superclass) ή βασική κλάση (base class)
 - ◆ Η παραγόμενη κλάση μπορεί εκτός από τα χαρακτηριστικά που κληρονομεί να ορίζει δικά της πεδία και μεθόδους

➤ **Στατικό δέσιμο μεθόδων (static binding)**

- ◆ Σε κάθε κλήση μεθόδου πρέπει να είναι γνωστό σε ποια κλάση ανήκει η μέθοδος
- ◆ Αν η αναζήτηση γίνεται από τον μεταγλωττιστή βάσει του τύπου του αντικειμένου πάνω στο οποίο καλείται η μέθοδος τότε λέμε ότι η γλώσσα υποστηρίζει στατικό δέσιμο μεθόδων (static binding)
- ◆ Η μέθοδος αναζητείται πρώτα στην κλάση στην οποία ανήκει το αντικείμενο στο οποίο εφαρμόζεται
- ◆ Αν δε βρεθεί τότε αναζητείται στην υπερκλάση αυτής
- ◆ Αν δε βρεθεί και στην υπερκλάση τότε συνεχίζουμε στην υπερκλάση της υπερκλάσης και η αναζήτηση συνεχίζεται μέχρι τη ρίζα της ιεραρχίας
- ◆ Αν και εκεί δε βρεθεί τότε εμφανίζουμε μήνυμα λάθους

➤ **Δυναμικό δέσιμο μεθόδων (dynamic binding)**

- ◆ Λέγεται και πολυμορφισμός υποτύπων (subtype polymorphism) και καταχρηστικά πολυμορφισμός
- ◆ Αναζήτηση της συνάρτησης που πρέπει να κληθεί σε χρόνο εκτέλεσης
- ◆ Δεν είναι πάντοτε εύκολο να αποφασιστεί σε χρόνο μεταγλώττισης ποια μέθοδος πρέπει να κληθεί (π.χ. απαιτείται αποδεικτοδότηση ενός δείκτη)
- ◆ Απαιτείται διαφορετική παράσταση των αντικειμένων στη μνήμη
- ◆ Ένας πίνακας περιέχει τις διευθύνσεις των δυναμικών μεθόδων, ονομάζεται πίνακας ανταπόκρισης μεθόδων (method dispatch table) ή περιγραφέας κλάσης (class descriptor)

➤ **Υπερφόρτωση μεθόδου (method overloading)**

- ◆ Στην ίδια ή σε διαφορετικές κλάσεις υπάρχουν μέθοδοι με το ίδιο όνομα και διαφορετικά ορίσματα

➤ **Υποσκέλιση μεθόδου (method overriding)**

- ◆ Μία θυγατρική κλάση και η γονική της έχουν μία μέθοδο ομώνυμη και με τα ίδια ορίσματα.

➤ **Πολυμορφισμός (polymorphism)**

- ◆ Ο μεταγλωττιστής αποφασίζει ποια μέθοδο θα καλέσει, βασισμένος στον τύπο του τρέχοντος αντικειμένου

- **Αφηρημένη κλάση (abstract class)**
 - ◆ Μία κλάση που ορίζεται μόνο για να κληρονομηθεί σε θυγατρικές υποκλάσεις και δεν υπάρχουν δικά της στιγμιότυπα (αντικείμενα)
 - ◆ Δεν είναι δυνατόν να κατασκευαστούν αντικείμενα αυτής της κλάσης
- **Κατασκευαστές (constructors)**
 - ◆ Αρχικοποιούν ένα αντικείμενο αμέσως μετά τη δέσμευση μνήμης
 - ◆ Υλοποιούνται σαν μία μέθοδο που δεν επιστρέφει αποτέλεσμα
- **Καταστροφείς (destructors)**
 - ◆ Καλούνται αμέσως πριν την αποδέσμευση της μνήμης
 - ◆ Υλοποιούνται σαν μία μέθοδο που δεν επιστρέφει αποτέλεσμα ούτε έχει παραμέτρους

2. Η ΓΛΩΣΣΑ ΜΑΣ

2.1 Η περιγραφή της γλώσσας μας

Η γλώσσα που χρησιμοποιήθηκε στην άσκηση είναι βασισμένη στο εξής παράδειγμα:

```
class Shape:
{
    int x;
    int y;

    def void __init__(Shape self, int initx, int inity):

        {
            self.moveTo(initx, inity)
        }

    def int get_x(Shape self):
        {
            return self.x
        }
        def int get_y(Shape self):
            {
                return self.y
            }
    def void set_x(Shape self, int newx):
        {
            self.x = newx
        }
    def void set_y(Shape self, int newy):
        {
            self.y = newy
        }

    def void moveTo(Shape self, int newx, int newy):
        {
            self.set_x(newx);
            self.set_y(newy)
        }
    def void shift(Shape self, int deltax, int deltay):
```



```

    {
    self.moveTo(self.get_x() + deltax, self.get_y() + deltay)
    }

def abstract void draw(self)
}

#####

class Rectangle inherits Shape:
{
    int width;
    int height;

    def void __init__(Rectangle self, int initx, int inity, int initwidth, int initheight):
    {
        Shape.__init__(self, initx, inity);
        self.setWidth(initwidth);
        self.setHeight(initheight)
    }

    def int getWidth(Rectangle self):
    {
        return self.width
    }
    def int getHeight(Rectangle self):
    {
        return self.height
    }
    def void setWidth(Rectangle self, int newwidth):
    {
        self.width = newwidth
    }
    def void setHeight(Rectangle self, int newheight):
    {
        self.height = newheight
    }

    def void draw(Rectangle self):
    {
        print 'rectangle: ', self.get_x(), self.get_y(), self.getWidth(), self.getHeight()
    }
}

#####

class Circle inherits Shape:
{
    int radius;

```

```

def void __init__(Circle self, int initx, int inity, int iniradius):
    {
        Shape.__init__(self, initx, inity);
        self.setRadius(iniradius)
    }

def int getRadius(Circle self):
    {
        return self.radius
    }
def void setRadius(Circle self, int newradius):
    {
        self.radius = newradius
    }

def void draw(Circle self):
    {
        print 'circle: ', self.get_x(), self.get_y(), self.getRadius()
    }
}

#####

class Square inherits Rectangle:
{
    def void __init__(Square self, int initx, int inity, int initwidth):
        {
            Shape.__init__(self, initx, inity);
            self.setWidth(initwidth)
        }

    def void draw(Square self):
        {
            print 'square: ', self.get_x(), self.get_y(), self.getWidth()
        }
}

#####

class Couple inherits Shape:
{
    Shape s[2];

    def void __init__(Couple self, Shape s1, Shape s2):
        {
            self.s[0] = s1;
            self.s[1] = s2
        }

    def void draw(Couple self):

```

```

    {
        print 'couple:', self.s[0].get_x(), self.s[0].get_y(),self.s[1].get_x(), self.s[1].get_y()
    }
def void draw(Couple self, Shape obj):
{
    if obj.get_x()==1
        print 'first in couple:', self.s[0].get_x(), self.s[0].get_y();
    else if obj.get_x()==2
        print 'second in couple:', self.s[1].get_x(), self.s[1].get_y();
    else
        print 'couple:', self.s[0].get_x(), self.s[0].get_y(),self.s[1].get_x(), self.s[1].get_y();

}
}

#####

def int __main__():
{
    Rectangle r;
    Circle c;
    Square s;
    Couple S;

    r = Rectangle(1,2,3,4);
    r.draw();
    c = Circle(4,5,1);
    c.draw();
    s = Square(3,3,2);
    s.draw();
    r.moveTo(3,4);

    return 1
}

```

Όπως προκύπτει από το παράδειγμα, η γλώσσα περιέχει αντικειμενοστρέφεια και πιο συγκεκριμένα, αποτελείται από:

- Κλάσεις: Shape, Rectangle, Circle, Square, Couple
- Αντικείμενα: Rectangle r, Circle c, Square s, Couple S
- Κατασκευαστές: def void __init__()
- Κληρονομικότητα: class Rectangle inherits Shape, class Circle inherits Shape, class Square inherits Rectangle, class Couple inherits Shape

Επιπλέον χαρακτηριστικά της γλώσσας, πάντα σύμφωνα με το παράδειγμα, είναι ότι:

- Οι κλάσεις περιέχουν πεδία και μεθόδους. Τα πεδία των κλάσεων είναι αποκλειστικά ακέραιοι αριθμοί ή πίνακες ακεραίων.
- Υποστηρίζονται εντολές εξόδου.
- Υποστηρίζονται κλήσεις μεθόδων.
- Υπάρχει υπερφόρτωση μεθόδου.
- Υπάρχει πολυμορφισμός.
- Υποστηρίζεται αφηρημένη μέθοδος.
- Υπάρχει κληρονομικότητα.
- Υποστηρίζεται στατικό δέσιμο μεθόδων.

Όσον αφορά την κληρονομικότητα, υποστηρίζεται και πολλαπλή κληρονομικότητα, δηλαδή μία υποκλάση μπορεί να κληρονομεί από πολλές κλάσεις. Αυτή η λειτουργία δεν υπάρχει στο παράδειγμα.

Αυτά αλλά και γενικά όλα τα χαρακτηριστικά των αντικειμενοστραφών γλωσσών περιγράφονται αναλυτικά στην ενότητα 1.4.

Στόχος ήταν να πραγματοποιηθεί λεκτική και συντακτική ανάλυση στη συγκεκριμένη αντικειμενοστραφή γλώσσα και να παραχθεί ένα αρχείο σε γλώσσα C, δηλαδή σε γλώσσα διαδικασιακού προγραμματισμού.

2.2 Η γραμματική της γλώσσας μας

Token που χρησιμοποιούνται:

"break"	{ return(BREAK); }
"continue"	{ return(CONTINUE); }
"else"	{ return(ELSE); }
"if"	{ return(IF); }
"int"	{ return(INT); }
"return"	{ return(RETURN); }

"def"	{ return(DEF); }
"class"	{ return(CLASS); }
"abstract"	{ return(ABSTRACT); }
"inherits"	{ return(INHERITS); }
"print"	{ return(PRINT); }
"void"	{ return(VOID); }
{L}({L} {D})*	{ return(IDENTIFIER); }
{D}+	{ return(CONSTANT); }
\'(.)*\'	{ return(STRING_LITERAL); }
"+="	{ return(ADD_ASSIGN); }
"_="	{ return(SUB_ASSIGN); }
"++"	{ return(INC_OP); }
"--"	{ return(DEC_OP); }
"&&"	{ return(AND_OP); }
" "	{ return(OR_OP); }
"<="	{ return(LE_OP); }
">="	{ return(GE_OP); }
"=="	{ return(EQ_OP); }
"!="	{ return(NE_OP); }
","	{ return(';'); }
"{"	{ return('{'); }
"}"	{ return('}'); }
","	{ return(','); }

"."	{ return('.'); }
"="	{ return('='); }
"("	{ return('('); }
")"	{ return(')'); }
"["	{ return('['); }
"]"	{ return(']'); }
".	{ return('.'); }
"_"	{ return('_'); }
"+"	{ return('+'); }
"*"	{ return('*'); }
"/"	{ return('/'); }
"%"	{ return('%'); }
"<"	{ return('<'); }
">"	{ return('>'); }

Κανόνες γραμματικής:

signed_constant

: CONSTANT
| '+' CONSTANT
| '-' CONSTANT
;

primary_expression

: IDENTIFIER
| signed_constant
| STRING_LITERAL

| '(' assignment_expression ')'

;

postfix_expression

: primary_expression

| postfix_expression '[' assignment_expression ']'

| postfix_expression '(' ')'

| postfix_expression '(' argument_expression_list ')'

| postfix_expression '.' postfix_expression

| postfix_expression INC_OP

| postfix_expression DEC_OP

;

argument_expression_list

: assignment_expression

| argument_expression_list ',' assignment_expression

;

multiplicative_expression

: postfix_expression

| multiplicative_expression '*' postfix_expression

| multiplicative_expression '/' postfix_expression

| multiplicative_expression '%' postfix_expression

;

additive_expression

: multiplicative_expression

| additive_expression '+' multiplicative_expression

| additive_expression '-' multiplicative_expression

;

relational_expression

: additive_expression

| relational_expression '<' additive_expression

| relational_expression '>' additive_expression

| relational_expression LE_OP additive_expression

| relational_expression GE_OP additive_expression

;

equality_expression

: relational_expression

| equality_expression EQ_OP relational_expression

| equality_expression NE_OP relational_expression

;

logical_and_expression

: equality_expression

| logical_and_expression AND_OP equality_expression

;

logical_or_expression

: logical_and_expression

| logical_or_expression OR_OP logical_and_expression

;

assignment_expression

: logical_or_expression
| postfix_expression '=' assignment_expression
| postfix_expression ADD_ASSIGN assignment_expression
| postfix_expression SUB_ASSIGN assignment_expression
;

constant_expression

: logical_or_expression
;

declaration

: type_specifier init_declarator ';'
;

init_declarator

: direct_declarator
| direct_declarator '=' initializer
;

type_specifier

: INT
| IDENTIFIER
| VOID
;

direct_declarator

: IDENTIFIER
| direct_declarator '[' constant_expression ']'

| direct_declarator '(' parameter_list ')'
| direct_declarator '(' identifier_list ')'
| direct_declarator '(' ')'
;

parameter_list

: parameter_declaration
| parameter_list ',' parameter_declaration
;

parameter_declaration

: type_specifier direct_declarator
;

identifier_list

: IDENTIFIER
| identifier_list ',' IDENTIFIER
;

initializer

: assignment_expression
| '{' initializer_list '}'
;

initializer_list

: initializer
| initializer_list ',' initializer
;

statement

: assignment_expression
| jump_statement
| print_statement
;

no_semicolon_statement

: selection_statement
| iteration_statement
;

compound_statement

: '{ '}'
| '{' statement_list '}'
| '{' declaration_list '}'
| '{' declaration_list statement_list '}'
;

declaration_list

: declaration
| declaration_list declaration
;

statement_list

: statement
| no_semicolon_statement
| statement_list ';' statement
| statement_list no_semicolon_statement

;

selection_statement

: IF assignment_expression one_or_more_statements

| IF assignment_expression one_or_more_statements elseif_follow

;

elseif_follow

: ELSE IF assignment_expression one_or_more_statements

| ELSE IF assignment_expression one_or_more_statements elseif_follow

| ELSE IF assignment_expression one_or_more_statements else_follow

;

else_follow

: ELSE one_or_more_statements

;

iteration_statement

: WHILE assignment_expression one_or_more_statements

| DO one_or_more_statements WHILE assignment_expression

| FOR assignment_expression ';' assignment_expression ';'

assignment_expression one_or_more_statements

;

jump_statement

: CONTINUE

| BREAK

| RETURN

| RETURN postfix_expression

;

print_elements

: postfix_expression
| print_elements ',' postfix_expression
;

print_statement

: PRINT { isPrint = 1; } print_elements
;

one_or_more_statements

: statement ';'
| compound_statement
;

translation_unit

: class_or_main
| translation_unit class_or_main
;

class_or_main

: function_definition
| CLASS class_definition
;

function_definition

: DEF type_specifier direct_declarator ':' compound_statement
| DEF ABSTRACT type_specifier direct_declarator

;

class_definition

: IDENTIFIER ':' class_compound_statement
| IDENTIFIER INHERITS parents_list ':' class_compound_statement
;

parents_list

: IDENTIFIER
| parents_list ',' IDENTIFIER
;

class_compound_statement

: '{' class_compound_statement_content '}'
;

class_compound_statement_content

:
| declaration_list functions_list
| declaration_list
| functions_list
;

functions_list

: function_definition
| functions_list function_definition
;

3. ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

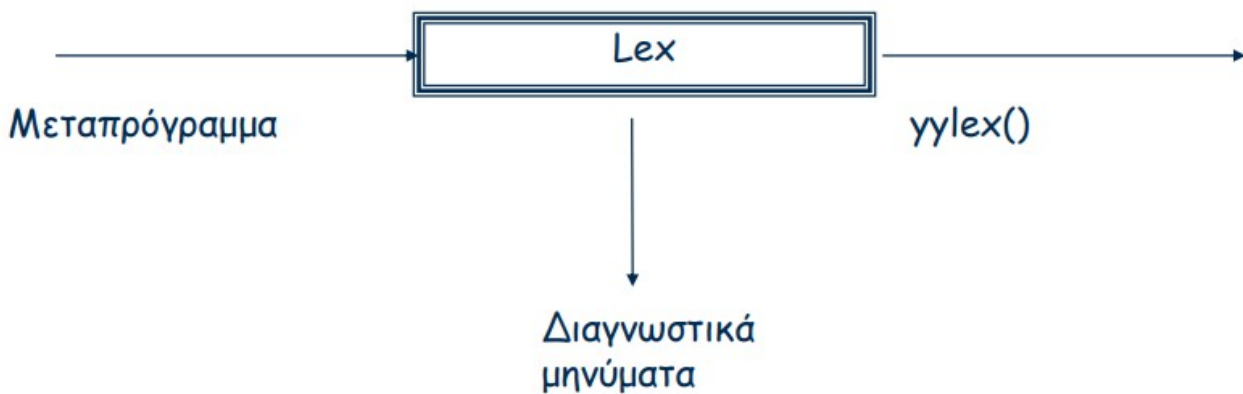
3.1 Ο ορισμός και η λειτουργία του λεκτικού αναλυτή

Ο λεκτικός αναλυτής είναι μία συνάρτηση που έχει ως ρόλο να διαβάσει γράμμα – γράμμα το πηγαίο πρόγραμμα και να επιστρέφει την επόμενη λεκτική μονάδα και έναν κωδικό που τη χαρακτηρίζει. Εσωτερικά λειτουργεί σαν ένα αυτόματο καταστάσεων, το οποίο ξεκινά από μία αρχική κατάσταση και με την είσοδο κάθε χαρακτήρα αλλάζει κατάσταση, έως ότου συναντήσει μία τελική κατάσταση. Το αυτόματο καταστάσεων αναγνωρίζει δεσμευμένες λέξεις, σύμβολα της γλώσσας, αναγνωριστικά και σταθερές, καθώς επίσης και λάθη.



3.2 Το εργαλείο flex

Το μεταεργαλείο flex είναι ένας γεννήτορας λεκτικών αναλυτών. Δέχεται σαν είσοδο ένα μεταπρόγραμμα που περιγράφει τις προς αναγνώριση λεκτικές μονάδες καθώς και τις ενέργειες που πρέπει να γίνουν όταν αυτές αναγνωριστούν. Η έξοδος του είναι ένα πρόγραμμα σε γλώσσα C που περιέχει τη συνάρτηση `yylex()`, η οποία υλοποιεί τον λεκτικό αναλυτή. Η συνάρτηση αυτή αναγνωρίζει την επόμενη λεκτική μονάδα και επιστρέφει έναν κωδικό που αντιστοιχεί σε αυτήν. Επιπλέον, αναγνωρίζει κανονικές εκφράσεις και κάνει κάποια ενέργεια για κάθε κανονική έκφραση που αναγνωρίζει.



Επιπλέον χαρακτηριστικά του εργαλείου flex

- Από τον flex παράγεται το αρχείο `lex.yy.c` το οποίο,
 - ◆ Διαβάζει μία συμβολοσειρά εισόδου
 - ◆ Χωρίζει την είσοδο σε μικρότερες συμβολοσειρές, αναγνωρίζοντας κανονικές εκφράσεις
 - ◆ Αν χρειάζεται μεταφέρει την είσοδο στην έξοδο

➤ Τελεστές “ \ [] ^ ? . - * + | () \$ / { } % < > ”

- ◆ Δεσμευμένοι χαρακτήρες
- ◆ Οποιοσδήποτε άλλος χαρακτήρας θεωρείται κείμενο
- ◆ Οι παραπάνω τελεστές αν θέλουμε να χρησιμοποιηθούν σαν χαρακτήρες πρέπει να προηγείται από αυτούς ο χαρακτήρας «\» , δηλαδή «\\» --> «\»

➤ Ομάδες χαρακτήρων

- ◆ [abc] οποιοσδήποτε από τους χαρακτήρες a b ή c
- ◆ [a-z] οποιοσδήποτε από τους χαρακτήρες 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
- ◆ [-+0-9] προσημασμένος μονοψήφιος αριθμός
- ◆ [^a-zA-Z] οτιδήποτε δεν είναι γράμμα
- ◆ «.» οποιοσδήποτε χαρακτήρας εκτός new line
- ◆ a? καμία ή μία εμφάνιση του a
- ◆ a* καμία ή περισσότερες εμφανίσεις του a
- ◆ a+ μία ή περισσότερες εμφανίσεις του a

➤ Προτεραιότητα τελεστών

- ◆ Από μεγαλύτερη σε μικρότερη
 - * ? +
 - παράθεση
 - |
- ◆ όλοι οι τελεστές συσχετίζονται προς τα αριστερά (left associative)

➤ Σύνταξη Lex

◆ **Μέρος Α:** **Ορισμοί**

- Σχόλια με τη σύμβαση της C
- Μνημονικά ονόματα
- Δηλώσεις αρχικών καταστάσεων
- Κώδικας C
 - Περικλείεται από %{ και %}
 - Συνήθως περιέχει δηλώσεις μακροεντολών, τύπων δεδομένων και μεταβλητών που χρησιμοποιούνται από το λεκτικό αναλυτή.

◆ **Μέρος Β:** **Κανόνες**

- Αποτελείται από κανόνες που περιγράφουν ομάδες λεκτικών μονάδων, ενώ σε κάθε κανόνα αντιστοιχίζονται και κάποιες ενέργειες.
- Διαβάζονται χαρακτήρες από το αρχείο εισόδου έως ότου αναγνωριστεί το μακρύτερο πρόθεμα από μία από τις παραπάνω κανονικές εκφράσεις. Αν το πρόθεμα αυτό περιγράφεται από περισσότερες της μίας κανονικής έκφρασης, τότε επιλέγεται το πρώτο.

◆ **Μέρος Γ:** **Συναρτήσεις**

- Κώδικας C

Σύμβολα περιγραφής στο flex

x	matches the character x
.	(Period) matches any single character except a newline.
\n	matches a newline character
* or "*"	\ is used both as an escape character, so that you can use a reserved character as a literal, and to specify certain control characters, such as newline characters (\n) and tabs (\t). If the \ does not specify a control character, then it escapes the character. For example, * is a literal asterisk, rather than an asterisk meaning 0 or more occurrences of a regular expression. Alternatively you can use quotes (" ") to specify that a reserved character should be interpreted literally as that character.
\$	By itself, \$ is a special symbol meaning end of input (EOF). For example, "\$" { return 0; }. Normally you do not care about EOF unless you need to do some sort of special processing, such as switching to another input file.
r\$	When placed at the end of a regular expression, \$ specifies that the string that matches the regular expression r must be at the end of the current line of input.
[xyz]	a character class that matches any of the characters between the []'s. In this case the character class matches any of x, y, or z
[a-zA-Z]	the '-' denotes a range of ascii characters. This specification matches any lower or upper case letter. Do not make the mistake of writing [a-Z] because there are ascii characters between lowercase 'z' and uppercase 'A' that would be included in the pattern.
[0-9]	any single digit
[\t\n\r\f]	matches any whitespace character. \r and \f stand for "return" and "form feed" and are often present in Windows generated files.
[^A-Z]	A ^ that is the <i>first</i> character inside the character class negates that character class, or alternatively, says any character but the characters in that character class. In this case [^A-Z] says anything except an uppercase letter

<code>^r</code>	When placed at the beginning of a pattern, the <code>^</code> says that the string which matches the regular expression <code>r</code> must start at the beginning of a line of input.
<code>[a-z]{-}</code> <code>[aeiou]</code>	The set difference operator (<code>-</code>) subtracts anything in the second character class from the first character class. In this case the pattern specifies the consonants.
<code>r*</code>	0 or more <code>r</code> 's, where <code>r</code> is any regular expression.
<code>r+</code>	1 or more <code>r</code> 's, where <code>r</code> is any regular expression.
<code>r?</code>	0 or 1 <code>r</code> 's, where <code>r</code> is any regular expression. You may also think of <code>?</code> as saying that the regular expression is optional. For example, <code>-?[0-9]</code> matches a single digit with an optional leading minus sign.
<code>r{2,5}</code>	Matches anywhere from 2 to 5 <code>r</code> 's
<code>r{4,}</code>	Matches 4 or more <code>r</code> 's
<code>r{4}</code>	Matches exactly 4 <code>r</code> 's
<code>rs</code>	the concatenation of the regular expressions <code>r</code> and <code>s</code> . You can also think of the pattern as <code>r</code> followed by <code>s</code> .
<code>rs</code>	the concatenation of the regular expressions <code>r</code> and <code>s</code> . You can also think of the pattern as <code>r</code> followed by <code>s</code> .
<code>r s</code>	either <code>r</code> or <code>s</code> (i.e., the union operation).
<code>[0-9]+</code>	any number
<code>.</code> <code>\n</code>	matches any character.
<code>(brad bvz)*</code>	parentheses are used to group regular expressions and to override precedence. For example, <code>brad bvz*</code> would typically match either "brad" or "bv" followed by 0 or more <code>z</code> 's. To instead match 0 or more occurrences of either "brad" or "bvz", you would use parentheses: <code>(brad bvz)*</code> .
<code>{DIGIT}+".</code> <code>{DIGIT}*</code>	A name that is placed between curly braces (<code>{}</code>) will be replaced by its associated pattern from the definitions section. If <code>DIGIT</code> were defined as <code>[0-9]</code> in the definitions section, then this pattern specifies a number that consists of 1 or more digits, followed by a period, followed by 0 or more digits. Note that the decimal point had to be placed in quotes to prevent it from being interpreted as a pattern that matches any single character.
<code><s>r</code>	A regular expression that is active only when state <code>s</code> is enabled. See Section States for more details.

<code><*>r</code>	A regular expression that is active in any state.
<code><s1,s2,s3>r</code>	A regular expression that is active only when state s1, s2, or s3 is active.

3.3 Η λειτουργία του flex στη γλώσσα μας

Η λεκτική ανάλυση στο πρόγραμμά μας πραγματοποιήθηκε με τη χρήση του εργαλείου flex. Πιο συγκεκριμένα, χρησιμοποιήθηκαν τα εξής:

ΜΕΡΟΣ Α

- **Μνημονικά ονόματα**

- D [0-9] ένα ψηφίο από το 0 έως και το 9
- L [a-zA-Z_] ένα μικρό ή κεφαλαίο γράμμα της αγγλικής αλφαβήτου ή ο χαρακτήρας _

- **Κώδικας C**

```
%{
#include <stdio.h>
#include "y.tab.h"

void count();
%}
```

ΜΕΡΟΣ Β

- **Κανόνες και σύμβολα περιγραφής**

➤ $\{L\}(\{L\}|\{D\})^*$

ένα μικρό ή κεφαλαίο γράμμα της αγγλικής αλφαβήτου ή ο χαρακτήρας _ και μετά οσαδήποτε γράμματα ή ψηφία από το 0 έως και το 9 (id)

➤ $\{D\}^+$

ένα ή περισσότερα ψηφία από το 0 έως και το 9 (number)

➤ $\backslash'(.)*'$

χαρακτήρας ' με οτιδήποτε άλλο στη συνέχεια και τέλος χαρακτήρας ' (string)

● Λεκτικές μονάδες – χαρακτήρες και token

"break"	BREAK
"continue"	CONTINUE
"else"	ELSE
"if"	IF
"int"	INT
"return"	RETURN
"def"	DEF
"class"	CLASS
"abstract"	ABSTRACT
"inherits"	INHERITS
"print"	PRINT
"void"	VOID
"+="	ADD_ASSIGN
"_="	SUB_ASSIGN
"*="	MUL_ASSIGN
"/="	DIV_ASSIGN
"%="	MOD_ASSIGN
"++"	INC_OP
"--"	DEC_OP
"&&"	AND_OP
" "	OR_OP
"<="	LE_OP
">="	GE_OP
"=="	EQ_OP
"!="	NE_OP

"."	'.'
","	','
"{"	'{'
"}"	'}'
" "	' '
","	','
"."	'.'
"="	'='
"("	'('
")"	')'
"["	'['
"]"	']'
"."	'.'
"_"	'_'
"+"	'+'
"*"	'*'
"/"	'/'
"%"	'%'
"<"	'<'
">"	'>'
[\t\v\n\f]	
.	

ΜΕΡΟΣ Γ

● Κώδικας C

```
int column = 0;

void count()
{
    int i;

    for (i = 0; yytext[i] != '\0'; i++)
        if (yytext[i] == '\n')
            column = 0;
        else if (yytext[i] == '\t')
            column += 8 - (column % 8);
        else
            column++;

    ECHO;
}
```

Η συνάρτηση `count()` χρησιμοποιείται στη διόρθωση σφαλμάτων του προγράμματος (debugging). Εμφανίζει στο τερματικό τις λεκτικές μονάδες που διαβάζει με κατάλληλα κενά/ tab/ αλλαγή γραμμής, κτλ.

4. ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

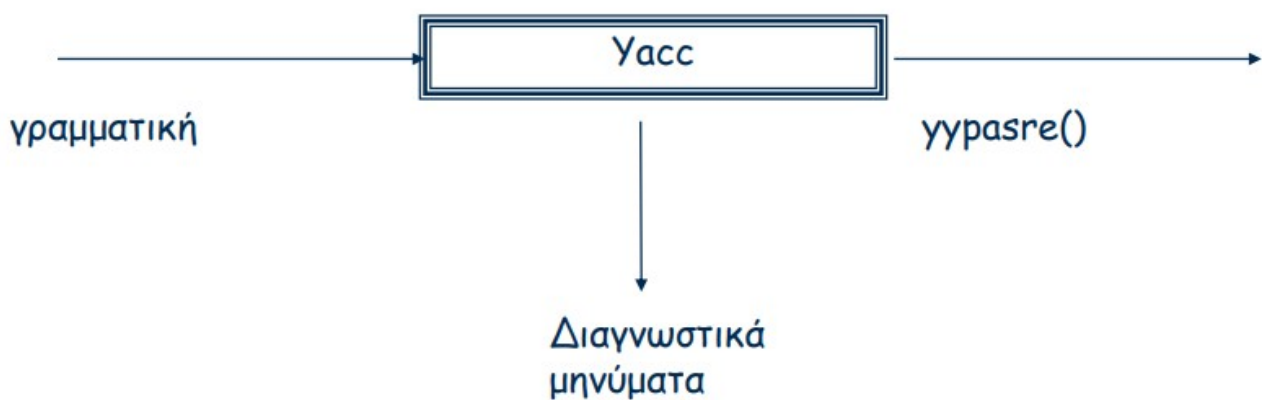
4.1 Ο ορισμός και η λειτουργία του συντακτικού αναλυτή

Με τον όρο συντακτικό αναλυτή αναφερόμαστε στο κομμάτι κώδικα που εκτελεί τη συντακτική ανάλυση. Είναι αυτός που χρησιμοποιεί το λεκτικό αναλυτή και σε αυτό γίνεται ο έλεγχος για να διαπιστωθεί εάν το πηγαίο πρόγραμμα ανήκει ή όχι στη γλώσσα.



4.2 Το εργαλείο bison

Το μεταεργαλείο bison είναι ένας γεννήτορας συντακτικών αναλυτών. Δέχεται σαν είσοδο ένα μεταπρόγραμμα και περιέχει τη σύνταξη μίας γλώσσας (κωδικοποιημένη με σε μία γραμματική). Η έξοδος είναι ένα πρόγραμμα σε γλώσσα C που περιέχει τη συνάρτηση `yyparse`, η οποία υλοποιεί το συντακτικό αναλυτή. Η συνάρτηση αναγνωρίζει ένα συντακτικά ορθό πρόγραμμα ή επιστρέφει μήνυμα λάθους.



➤ Σύνταξη bison

- | | |
|------------|-------------|
| ◆ Μέρος Α: | Ορισμοί |
| ◆ Μέρος Β: | Κανόνες |
| ◆ Μέρος Γ: | Συναρτήσεις |

Επιπλέον πληροφορίες για το bison

➤ Επίλυση συγκρούσεων

- ◆ Οι αμφισημίες στις γραμματικές μπορούν να επιλυθούν
 - Είτε κατασκευάζοντας μία μη διαφορούμενη γραμματική
 - Ή ευκολότερα χρησιμοποιώντας τις δυνατότητες που παρέχει ο bison για το σκοπό αυτό

➤ Αν οι κανόνες προτεραιότητας και συσχέτισης δεν επιλύουν τις συγκρούσεις, τότε οι συγκρούσεις επιλύονται με βάση τους ακόλουθους κανόνες

- ◆ Η σύγκρουση ελλάτωσης - ελλάτωσης επιλύεται επιλέγοντας τον κανόνα που έχει δηλωθεί πρώτος στη γραμματική.
- ◆ Η σύγκρουση ολίσθησης - ελλάτωσης επιλύεται επιλέγοντας ολίσθηση.

➤ Προτεραιότητα και συσχέτισης

- ◆ Μεγαλύτερη προτεραιότητα έχουν οι τελεστές διαίρεσης και πολλαπλασιασμού και μικρότερη αυτοί της πρόσθεσης και αφαίρεσης
- ◆ Οι τελεστές '+' '-' και '*' '/' έχουν αριστερή συσχέτιση, δηλαδή $\alpha + \beta + \gamma = (\alpha + \beta) + \gamma$
- ◆ Ο τελεστής '=' έχει δεξιά συσχέτιση, δηλαδή $\alpha = \beta = \gamma$ σημαίνει $\alpha = (\beta = \gamma)$

4.3 Η λειτουργία του bison στη γλώσσα μας

Η συντακτική ανάλυση στη γλώσσα μας πραγματοποιήθηκε με τη χρήση του εργαλείου bison. Σ' αυτόν υπάρχει η γραμματική της γλώσσας μας, η οποία περιγράφεται στην ενότητα 2.2.

Σ' αυτόν ορίζουμε επίσης τους κανόνες για το flex:

```
%token IDENTIFIER CONSTANT STRING_LITERAL VOID
%token INC_OP DEC_OP LE_OP GE_OP EQ_OP NE_OP AND_OP OR_OP
%token MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
SUB_ASSIGN
%token INT
%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR CONTINUE
BREAK RETURN
%token DEF CLASS ABSTRACT INHERITS PRINT
```

5. ΠΑΡΑΓΩΓΗ ΚΩΔΙΚΑ ΣΕ ΓΛΩΣΣΑ C

5.1 Προσθήκες στον κώδικα του flex

- Στον κανόνα για τα id ({L}({L}|{D})*) αποθηκεύεται επιπλέον το id που βρίσκεται στο yytext, στο yyval.
- Στον κανόνα για τους αριθμούς ({D}+) αποθηκεύεται επίσης ο αριθμός που βρίσκεται στο yytext, στο yyval.

- Στον κανόνα για τα string (`\'(.)*\'`) αφαιρούνται οι χαρακτήρες ‘ και ’ που βρίσκονται στην αρχή και στο τέλος του string και το υπόλοιπο αποθηκεύεται στο `yynval`.

5.2 Προσθήκες στον κώδικα του bison

Στο εργαλείο bison, δηλαδή στη γραμματική της γλώσσας, προστέθηκαν δομές, συναρτήσεις και κώδικας για τη μετατροπή σε γλώσσα C, όπως παρουσιάζεται παρακάτω:

● Δομές

- **struct Class** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τις κλάσεις, δηλαδή:
 - **char name[40]** όνομα κλάσης
 - **Field fields[10]** πεδία
 - **int fieldsSize** πλήθος πεδίων
 - **Class *parents** γονείς
 - **int countParents** πλήθος γονέων
- **struct Field** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τα πεδία, δηλαδή:
 - **char type[20]** τύπος
 - **char name[40]** όνομα
 - **int isArray** μέγεθος πίνακα, αν είναι
 - **int *int_value** τιμή/τιμές τύπου int
 - **Class *object_value** τιμή/τιμές object

- struct Method** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τις μεθόδους, δηλαδή:
 - **int isAbstract** flag για αφηρημένη μέθοδο
 - **Parameter parameters[20]** παράμετροι
 - **char return_type[30]** τύπος επιστροφής
 - **char methodTitle[500]** επεξεργασμένο όνομα
 - **char methodBody[2000]** παράμετροι με τύπους
- struct Parameter** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τις παραμέτρους, δηλαδή:
 - **char type[20]** τύπος
 - **char name[40]** όνομα
- struct Object** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τα αντικείμενα, δηλαδή:
 - **Class *object_class** τύπος
 - **char name[40]** όνομα

Εκτός από τα παραπάνω, υλοποιήσαμε και δύο συνδεδεμένες λίστες. Η μία αφορά την αποθήκευση των κλάσεων του προγράμματος και η άλλη την αποθήκευση των αντικειμένων. Για την υλοποίηση και τη διαχείριση των λιστών υπάρχουν:

- struct class_node** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τον κόμβο της κλάσης, δηλαδή:

- **Class value** κλάση
- **struct class_node_s *next** δείκτης στον επόμενο κόμβο
- **struct object_node** χρησιμοποιείται για αποθήκευση κάποιων πληροφοριών που αφορούν τον κόμβο του αντικειμένου, δηλαδή:
 - **Object value** κλάση
 - **struct class_node_s *next** δείκτης στον επόμενο κόμβο
- **void class_addNode()** προσθέτει ένα κόμβο στη λίστα με τις κλάσεις
- **void object_addNode()** προσθέτει ένα κόμβο στη λίστα με τα αντικείμενα
- **class_node *class_head** πρώτος κόμβος λίστας κλάσεων
- **class_node *class_tail** τελευταίος κόμβος λίστας κλάσεων
- **object_node *object_head** πρώτος κόμβος λίστας αντικειμένων
- **object_node *object_tail** τελευταίος κόμβος λίστας αντικειμένων
- Προσωρινές μεταβλητές
 - **Class *newClass** περιέχει την κλάση που βρισκόμαστε τώρα
 - **Field *newField** περιέχει το πεδίο που αρχικοποιούμε τώρα
 - **Method *newMethod** περιέχει τη μέθοδο που βρισκόμαστε τώρα
 - **Parameter *newParameter** περιέχει την παράμετρο που αρχικοποιούμε τώρα

● Συναρτήσεις

- **int findSelfField(char *str)** παίρνει ως όρισμα ένα string και ελέγχει αν ανήκει στα πεδία της κλάσης – σε περίπτωση πίνακα, κρατάει το όνομά του και ψάχνει στα πεδία με βάση αυτό – αν δεν το βρει επιστρέφει μήνυμα λάθους
- **int findClass(char *name)** παίρνει ως όρισμα το όνομα μιας κλάσης και ψάχνει στη λίστα με τις κλάσεις – αν το βρει επιστρέφει τη θέση της στη λίστα
- **char *typeOfCallerVariable(char *variable)**
παίρνει ως όρισμα μια μεταβλητή, αν είναι πίνακας κρατάει μόνο το όνομα και ψάχνει στις παραμέτρους της μεθόδου που βρισκόμαστε τώρα και στα ονόματα των κλάσεων να την εντοπίσει – αν την εντοπίσει επιστρέφει τον τύπο της, αλλιώς μήνυμα λάθους
- **int checkParentClass(char * classType)**
παίρνει ως όρισμα μια κλάση και ελέγχει αν είναι γονέας της κλάσης που βρισκόμαστε τώρα
- **void checkIfParentExists(char *name)**
παίρνει ένα string και ελέγχει αν είναι όνομα κλάσης – αν δεν είναι εμφανίζει μήνυμα λάθους
- **void createParent(char *name)**

παίρνει ως όρισμα το όνομα μιας κλάσης – ελέγχει αν ανήκει στη λίστα με τις κλάσεις και τη βάζει ως γονιό στην κλάση που βρισκόμαστε τώρα

- **int findParent(char *name)** παίρνει ως όρισμα το όνομα μίας κλάσης και ελέγχει αν είναι γονιός της κλάσης που βρισκόμαστε – αν είναι επιστρέφει τη θέση της στη λίστα με τους γονείς, αλλιώς εμφανίζει μήνυμα λάθους
- **void registerGrandparents(char *parentName)**

παίρνει ως όρισμα το όνομα ενός γονέα και αφού τον εντοπίσει στη λίστα με τις κλάσεις, αν έχει γονείς, βάζει τους γονείς του στην λίστα με τους γονείς της κλάσης που βρισκόμαστε
- **char *findObject(char *name)** παίρνει ως όρισμα το όνομα ενός αντικειμένου και το ψάχνει στη λίστα με τα αντικείμενα – αν το βρει αποθηκεύει σε ένα buffer που βρίσκεται στη λίστα, αλλιώς επιστρέφει μήνυμα λάθους
- **int searchObject(char *name)** παίρνει ως όρισμα το όνομα ενός αντικειμένου και ελέγχει αν υπάρχει στη λίστα με τα αντικείμενα

- **char *searchObjectClass(char *name)**
παίρνει ως όρισμα το όνομα ενός αντικειμένου και ελέγχει αν υπάρχει στη λίστα με τα αντικείμενα
- **void makeObjectClassCopy(char *name)**
παίρνει ως όρισμα ένα όνομα και δημιουργεί ένα αντικείμενο με αυτό το όνομα και όλα τα χαρακτηριστικά της κλάσης του τύπου του αντικειμένου
- **char *searchArgType(char *arg)**
παίρνει ως όρισμα το όνομα μίας παραμέτρου και ψάχνει στις παραμέτρους της μεθόδου που βρισκόμαστε για να εντοπίσει τον τύπο της
- **char *printParamTypes()**
επιστρέφει τους τύπους των ορισμάτων που παίρνει η μέθοδος
- **void printClassesToFile()**
αποθηκεύει στο αρχείο τα περιεχόμενα των δομών που έχουμε, που αφορούν τις κλάσεις
- **void printObjectsToFile()**
αποθηκεύει στο αρχείο τα περιεχόμενα των δομών που έχουμε, που αφορούν τα αντικείμενα
- **void *createTheMethods()**
ψάχνει στο methodBody, δηλαδή στο περιεχόμενο της μεθόδου - αν εντοπίσει κλίση συνάρτησης καλεί τη

searchForMethod για να δει αν υπάρχει στις μεθόδους της κλάσης - αν δεν υπάρχει καλείται η searchParentsForMethod, για να δει αν υπάρχει σε κάποια γονική κλάση και τότε αλλάζει το όνομα της, ώστε να καλείται η κατάλληλη μέθοδος - διαφορετικά επιστρέφει μήνυμα λάθους

- **int searchForMethod(char *result)**

παίρνει ως όρισμα το όνομα μιας μεθόδου και ψάχνει αν υπάρχει στις μεθόδους της κλάσης που βρισκόμαστε

- **int searchParentsForMethod(char **result)**

παίρνει σαν όρισμα το επεξεργασμένο όνομα μιας μεθόδου – κάνει κατάλληλη επεξεργασία και κρατάει το όνομα της μεθόδου και την κλάση που βρισκόμαστε – ψάχνει αν η μέθοδος υπάρχει στους γονείς της κλάσης που βρισκόμαστε

- Κώδικας σε κανόνες της γραμματικής

- Όλοι οι κανόνες επιστρέφουν τύπο string

- **signed_constant**

Επιστρέφει αυτό που του έρχεται

- **primary_expression**

Αν είναι string, προσθέτει στην αρχή και στο τέλος του τον

χαρακτήρα “ και το επιστρέφει, αλλιώς επιστρέφει αυτό που του έρχεται.

- **postfix_expression**

Αν είναι απλή έκφραση επιστρέφει αυτό που του έρχεται. Αν είναι πίνακας επιστρέφει όνομα πίνακα '[' περιεχόμενο ']'. Αν είναι μέθοδος χωρίς παραμέτρους:

Αν βρισκόμαστε στη main, επιστρέφει το όνομα της μεθόδου και θέτει το constructor_parameters σε κενό αφού δεν έχει παραμέτρους η μέθοδος.

Αν δεν βρισκόμαστε στη main, αν η κλήση έγινε με self, τότε επιστρέφει όνομα κλάσης \$ όνομα μεθόδου \$ όνομα κλάσης (τύπος self) και ως παράμετρο το self. Αν η κλήση δεν έγινε με self, τότε στο keepParent υπάρχει η πληροφορία, αν η κλήση έγινε με γονιό ή όχι. Αν έγινε με γονιό επιστρέφει όνομα μεθόδου \$ self → object_class → parents[θέση γονιού στη λίστα με τους γονείς], αλλιώς επιστρέφει όνομα μεθόδου και ως παράμετρος το self.

Αν είναι μέθοδος με παραμέτρους:

Αν βρισκόμαστε στη main, επιστρέφει το όνομα της μεθόδου και προσθέτει στο constructor_parameters τις παραμέτρους της μεθόδου.

Αν δεν βρισκόμαστε στη main, αν η κλήση έγινε με self, τότε επιστρέφει όνομα κλάσης \$ όνομα μεθόδου \$ τύπους παραμέτρων χωρισμένα με \$. Αν η κλήση δεν έγινε με self, τότε στο keepParent υπάρχει η πληροφορία, αν η κλήση έγινε με γονιό. Αν έγινε με γονιό επιστρέφει όνομα μεθόδου \$ self → object_class → parents[θέση γονιού στη λίστα με τους γονείς], αλλιώς επιστρέφει όνομα μεθόδου και ως παράμετρος το self.


Αν είναι κλήση μεθόδου:

Ελέγχει αν κάλεσε γονιός ή έγινε κλήση με self. Όσον αφορά τη μέθοδο που καλείται:

Αν βρισκόμαστε στη main, αν η μέθοδος είναι χωρίς παραμέτρους επιστρέφει όνομα κλάσης αντικειμένου που καλεί \$ όνομα μεθόδου και ως παράμετρο το όνομα του αντικειμένου. Αν είναι μέθοδος με παραμέτρους επιστρέφει όνομα κλάσης αντικειμένου που καλεί \$ όνομα μεθόδου \$ τύπους παραμέτρων χωρισμένα με \$ και τις παραμέτρους της μεθόδου.

Αν δεν βρισκόμαστε στη main, αν καλείται μέθοδος με self την επιστρέφουμε κατάλληλα, αν καλείται πίνακας με self επιστρέφουμε self → object_class → fields[θέση πίνακα στα πεδία] → int_value[στοιχείο του πίνακα].

Γενικά ακολουθούμε την εξής λογική:

<pre> class Shape var posX, posY: real; procedure move (dx,dy:real) begin posX:=posX+dx; posY:=posY+dy; end end var s: Shape; class Shape var posX, posY: real; procedure constructor (dx,dy:real) begin posX:=dx; posY:=dy; end procedure destructor (l) begin ... end end </pre>		<pre> procedure Shape@move (var self: Shape; dx,dy:real) begin self.posX:=self.posX+dx; self.posY:=self.posY+dy; end procedure Shape@constructor (var self:Shape;x,y:real); begin self.posX:=x; self.posY:=y; end procedure Shape@destructor(var self:Shape) begin ... end; </pre>
---	---	--

```

var  s:Shape(10,20)
      c:Circle(30,30,10)
      l:Line(10,20,30,30)

s.move (5,5)  --->    Shape@move(s,5,5)
c.move(5,5)   --->    Shape@move(c,5,5)
l.move(5,5)   --->    Line@move(l,5,5)
c.scale(2)    --->    Circle@scale(c,2)

```

- **argument_expression_list**

Επιστρέφει τη λίστα με τα ορίσματα και κρατάει στο argumentTypes τους τύπους τους.

- **multiplicative_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **additive_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **relational_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **equality_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **logical_and_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **logical_or_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους.

- **assignment_expression**

Επιστρέφει αυτό που του έρχεται ή επιστρέφει αυτά που του έρχονται με τις κατάλληλες πράξεις μεταξύ τους. Αν έχουμε περίπτωση = τότε αν έχουμε στη main αρχικοποίηση πεδίου, επιστρέφει κλήση constructor της κατάλληλης κλάσης, αλλιώς απλά επιστρέφει αυτά που έρχονται με κλήση μεταξύ τους.

- **constant_expression**

Επιστρέφει αυτό που του έρχεται.

- **declaration**

Δεν γίνεται κάτι εδώ.

- **init_declarator**

Αν βρισκόμαστε στη main δημιουργούμε καινούριο αντικείμενο ή integer ή πίνακα με integer ή πίνακα με αντικείμενα.

- **type_specifier**

Προσθέτει τον κατάλληλο τύπο στη μέθοδο, πεδίο, παράμετρο, ανάλογα την περίπτωση που έχουμε.

- **direct_declarator**

Αν είναι μέθοδος main ή παράμετρος ή μέθοδος, ενεργοποιεί τα κατάλληλα flags. Αν είναι πεδίο του προσθέτει το όνομα.

- **parameter_list**

Δεν γίνεται κάτι εδώ.

- **parameter_declaration**

Προσθέτει όνομα σε παράμετρο και την προσθέτει στη λίστα παραμέτρων της μεθόδου που βρισκόμαστε.

- **identifier_list**

Δεν γίνεται κάτι εδώ.

- **initializer**
Επιστρέφει αυτό που του έρχεται.
- **initializer_list**
Χρησιμοποιείται για αρχικοποιήσεις πεδίων ή πίνακα.
- **statement**
Επιστρέφει αυτό που του έρχεται.
- **no_semicolon_statement**
Επιστρέφει αυτό που του έρχεται.
- **compound_statement**
Επιστρέφει αυτό που του έρχεται.
- **declaration_list**
Δεν γίνεται κάτι εδώ.
- **statement_list**
Επιστρέφει αυτό που του έρχεται.
- **selection_statement**
Επιστρέφει αυτό που του έρχεται με κατάλληλη σύνταξη if.
- **elseif_follow**
Επιστρέφει αυτό που του έρχεται με κατάλληλη σύνταξη else if.
- **else_follow**
Επιστρέφει αυτό που του έρχεται με κατάλληλη σύνταξη else.
- **jump_statement**
Επιστρέφει τις κατάλληλες εντολές.
- **print_elements**
Αποθηκεύει με κατάλληλο τρόπο στο printableForms ό,τι βρίσκεται μετά το print.

- **print_statement**
Δημιουργεί το κείμενο με τους τύπους μπροστά για το print και το επιστρέφει.
- **one_or_more_statements**
Επιστρέφει αυτό που του έρχεται.
- **translation_unit**
Δεν γίνεται κάτι εδώ.
- **class_or_main**
Δημιουργεί νέα κλάση.
- **function_definition**
Αν βρισκόμαστε στη main τυπώνει στο αρχείο κάποια πρωτότυπα, έπειτα τη συνάρτηση main με κάποιες κλήσεις συναρτήσεων, τυπώνει τις επεξεργασμένες μεθόδους, τις κλάσεις και τα αντικείμενα. Αν δεν βρισκόμαστε στη main δημιουργεί τα πρωτότυπα και τα περνάει στο αρχείο. Αν η μέθοδος είναι abstract ενεργοποιείται το κατάλληλο flag.
- **class_definition**
Δίνει όνομα στην κλάση που βρισκόμαστε.
- **parents_list**
Για κάθε γονιό ψάχνει αν υπάρχει, αλλιώς τον δημιουργεί και τον βάζει στη λίστα με τους γονείς της κλάσης που βρισκόμαστε.
- **class_compound_statement**
Ενεργοποιείται το flag που αφορά τις παραμέτρους πριν αρχίσουν να διαβάζονται παράμετροι και απενεργοποιείται όταν διαβαστεί και η τελευταία παράμετρος.
- **functions_list**
Δεν γίνεται κάτι εδώ.

- Στη main του κώδικα του bison

Αρχικοποιούνται οι προσωρινές μεταβλητές που αναφέρθηκαν πιο πάνω. Επίσης, δημιουργείται ένα αρχείο .c με ίδιο όνομα με το πηγαίο πρόγραμμα. Σ' αυτό περνιούνται αρχικά κάποιες βιβλιοθήκες της γλώσσας C, οι δομές που δημιουργήθηκαν και όλες οι πληροφορίες που υπήρχαν στο αρχικό πρόγραμμα, αλλά σε μορφή που να ικανοποιείται η γλώσσα C.

6. TEST

TEST 1

```
class ExampleClass:
{
    int number;

    def void __init__ (ExampleClass self):
    {
        self.number = 2
    }

    def void yell(ExampleClass self):
    {
        print 'Hello Greece!'
    }
}

def int __main__():
{
```

```

ExampleClass ex;

print '#Test1: Basic Hello World Test\n';
ex = ExampleClass();
ex.yell()
}

```

TEST 2

```

class ExampleFieldClass:
{
    int exFieldNumber;

    def void __init__(ExampleFieldClass self):
    {
        exFieldNumber = 2
    }

    def void setExFieldNumber(int num):
    {
        self.exFieldNumber=num
    }

    def int getExFieldNumber(ExampleFieldClass self):
    {
        return self.exFieldNumber
    }
}

class ExampleClass:
{
    int number = 1;
    int number_array[2] = {10, 11};
    ExampleFieldClass object;
    ExampleFieldClass object_array[2];
}

```

```

def void createExampleFieldClass(ExampleClass self):
{
    self.object = ExampleFieldClass();
    self.object_array[0] = ExampleFieldClass();
    self.object_array[1] = ExampleFieldClass();

    self.object.setExFieldNumber(2);
    self.object_array[0].setExFieldNumber(20);
    self.object_array[1].setExFieldNumber(21)
}

def void toString(ExampleClass self):
{
    print 'number: ', number, ', number_array[0]: ', number_array[0], ', number_array[1]: ',
number_array[1],
        ', object: ', object.getExFieldNumber(), ', object_array[0]:',
object_array[0].getExFieldNumber(),
        ', object_array[1]:', object_array[1].getExFieldNumber()
    }
}

def int __main__():
{
    ExampleClass ex;

    print '#Test2: Interacting with Fields\n';
    ex = ExampleClass();
    ex.toString()
}

```

TEST 3

```
class ExampleClass:
{
    int values[2];
    int result;

    def void __init__(ExampleClass self):
    {
        result = 2
    }

    def void createVals(ExampleClass self):
    {
        self.values[0] = 10;
        self.values[1] = 5;
        self.result = 0
    }

    def void doSomeMath(ExampleClass self):
    {
        print '##\nTesting math expressions\n';
        self.result = self.values[0] || self.values[1];
        print self.values[0], ' || ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] && self.values[1];
        print self.values[0], ' && ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] == self.values[1];
        print self.values[0], ' == ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] != self.values[1];
        print self.values[0], ' != ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] < self.values[1];
        print self.values[0], ' < ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] > self.values[1];
        print self.values[0], ' > ', self.values[1], ' = ', self.result, '\n';

        self.result = self.values[0] + self.values[1];
```

```

print self.values[0], ' <= ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] + self.values[1];
print self.values[0], ' >= ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] + self.values[1];
print self.values[0], ' + ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] - self.values[1];
print self.values[0], ' - ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] * self.values[1];
print self.values[0], ' * ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] / self.values[1];
print self.values[0], ' / ', self.values[1], ' = ', self.result, '\n';

self.result = self.values[0] % self.values[1];
print self.values[0], ' % ', self.values[1], ' = ', self.result, '\n'
}

def void iterate(ExampleClass self):
{
    print '##\nTesting ++ and --\n';
    print 'value now = ', self.result;
    self.result ++;
    print 'value ++ = ', self.result;
    self.result --;
    print 'value -- = ', self.result
}
}

def int __main__():
{
    ExampleClass ex;
    print '#Test3: Supported expressions\n';
    ex = ExampleClass();
    ex.doSomeMath();
    ex.iterate()
}

```

TEST 4

```
class ExampleClass:
{
    int bool_vals[2];
    int value;

    def void __init__(ExampleClass self):
    {
        self.value = 2
    }

    def void createVals(ExampleClass self):
    {
        self.bool_vals[0] = 0;
        self.bool_vals[1] = 1;
        self.value = 5
    }

    def void useIf_ElseIf_Else(ExampleClass self):
    {
        if self.bool_vals[0]==0
            print 'Right Answer: Value is 0\n';
        else if bool_vals[0]==1
            print 'Wrong Answer: Value is not 1\n';
        else
            print 'Wrong Answer: Value should be 0\n';

        if self.bool_vals[1]==0
            print 'Wrong Answer: Value is not 0\n';
        else if bool_vals[1]==1
            print 'Right Answer: Value is 1\n';
        else
            print 'Wrong Answer: Value should be 1\n';

        if self.bool_vals[1]==0
            print 'Wrong Answer: Value is not 0\n';
        else if bool_vals[1]==1
            print 'Wrong Answer: Value is not 1\n';
        else
            print 'Right Answer: Value is neither 0 nor 1\n';
```

```

}

def void useIf_ElseIf_Else_2statements(ExampleClass self):
{
    if self.bool_vals[0]==0 {
        print 'if-1\n';
        print 'if-2\n'
    }
    else if bool_vals[0]==1 {
        print 'else if-1\n';
        print 'else if-2\n'
    }
    else {
        print 'else-1\n';
        print 'else-2\n'
    }
}

def int __main__():
{
    return '#Test4: If, If-Else, Else Statements\n'
}

```