ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
Μάθημα: Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Αναφορά στην εργαστηριακή Άσκηση Flex/Bison (2025)

Εργασία των Φοιτητών:	ΑΔΑΜΟΠΟΥΛΟΣ ΘΕΟΔΩΡΟΣ	**1108389**
	ΔΗΜΟΠΟΥΛΟΣ ΗΛΙΑΣ	**1108376**
	ΕΥΘΥΜΙΟΥ ΓΕΩΡΓΙΟΣ	**1108319**

Υπεύθυνοι Καθηγητές: Γαροφαλάκης Ιωάννης
Σιούτας Σπυρίδων
Βονιτσάνος Γεράσιμος

Περιεχόμενα

Εισαγωγή

Η παρούσα αναφορά αφορά την υλοποίηση ενός λεξικού και συντακτικού αναλυτή για τη γλώσσα **myHTML**, στο πλαίσιο της εργαστηριακής άσκησης του μαθήματος «Γλώσσες και Μεταφραστές» για το εαρινό εξάμηνο 2025. Η γλώσσα **myHTML** αποτελεί μια απλουστευμένη εκδοχή της HTML και σχεδιάστηκε ώστε να διευκολύνει την κατανόηση των βασικών εννοιών της ανάλυσης κειμένου και της κατασκευής αναλυτών με τα εργαλεία **Flex** και **Bison**.

Στόχος της άσκησης είναι η δημιουργία ενός συστήματος που αναλύει αρχεία γραμμένα σε myHTML και εντοπίζει συντακτικά λάθη, ενώ παράλληλα διατηρεί πληροφορίες σχετικά με τη δομή του εγγράφου. Η εργασία περιλαμβάνει την κατασκευή του λεξικού αναλυτή (lexer) με το εργαλείο Flex, του συντακτικού αναλυτή (parser) με το εργαλείο Bison, καθώς και την ενσωμάτωσή τους σε ένα λειτουργικό πρόγραμμα.

Εργαλεία και βιβλιοθήκες που χρησιμοποιήθηκαν

Εργαλεία

- **Flex (Fast Lexical Analyzer Generator):** Χρησιμοποιήθηκε για τη δημιουργία λεξικού αναλυτή (scanner). Διαβάζει το αρχείο εισόδου χαρακτήρα προς χαρακτήρα και το χωρίζει σε tokens, σύμφωνα με προκαθορισμένες κανονικές εκφράσεις (regular expressions).
- **Bison:** Χρησιμοποιήθηκε για τη δημιουργία συντακτικού αναλυτή (parser). Παίρνει ως είσοδο τα tokens από τον λεξικό αναλυτή και ελέγχει αν αυτά ακολουθούν τους κανόνες της συντακτικής δομής της γλώσσας μας.
- **GCC (GNU Compiler Collection):** Χρησιμοποιήθηκε για τη μεταγλώττιση των αρχείων που παράγονται από τα εργαλεία Flex και Bison, καθώς και του υπολοίπου βοηθητικού κώδικα C σε εκτελέσιμο αρχείο.
- **Make:** Χρησιμοποιήθηκε για αυτοματοποίηση της διαδικασίας μεταγλώττισής του προγράμματος μας. Διαβάζει ένα αρχείο τύπου Makefile, στο οποίο ορίσαμε τους διάφορους κανόνες για τη μεταγλώττισή και την εκτέλεση του προγράμματος μας.

Βιβλιοθήκες

- **stdio.h:** Χρησιμοποιήθηκε για εκτύπωση κειμένου στο τερματικό, μέσω της συνάρτησης printf/fprintf, για το διάβασμα του αρχείου που περιέχει το πρόγραμμα της γλώσσας myHTML, μέσω της συνάρτησης fopen.
- **stdlib.h:** Χρησιμοποιήθηκε για δυναμική δέσμευση μνήμης στην υλοποίηση της συνδεδεμένης λίστας (Χρησιμοποιείται επίσης και από τη flex/bison για τη δημιουργία ενός input buffer και ενός stack)
- **stdbool.h:** Χρησιμοποιήθηκε για να μπορούμε να έχουμε λογικές μεταβλητές (true/false). Αντί να χρησιμοποιούμε ακέραιες τιμές για να αναπαραστήσουμε το true και το false, αξιοποιούμε τον τύπο bool ώστε ο κώδικας να είναι πιο εκφραστικός και αφαιρετικός.
- **string.h:** Χρησιμοποιήθηκε για να μπορούμε να χειριστούμε πίνακες συμβολοσειρών ως strings. Συγκεκριμένα, χρησιμοποιήσαμε την συνάρτηση strncmp για να συγκρίνουμε μεταξύ τους δυο συμβολοσειρές, την strlen για να πάρουμε το μέγεθος (δηλαδή το πλήθων των χαρακτήρων) μιας συμβολοσειράς, τις strndup, strdup για αντιγραφή συμβολοσειρών.
- **linked_list.h:** Πρόκειται για μία τοπική βιβλιοθήκη που δημιουργήσαμε, και χρησιμοποιήθηκε για να έχουμε υλοποίηση λιστών. Συγκεκριμένα περιέχει τα παρακάτω:

  - **Struct για κόμβο Linked_list (τύπος Node):** Αντιπροσωπεύει ένα στοιχείο της κύριας συνδεδεμένης λίστας. Περιλαμβάνει:

    - Έναν pointer προς τον επόμενο κόμβο (next).
    - Έναν pointer προς μια συμβολοσειρά (id_name), η οποία αντιπροσωπεύει ένα μοναδικό αναγνωριστικό (στην περίπτωσή μας, τα id των tags).

  - **Struct για κόμβο Address_list (τύπος pNode):** Αποτελεί στοιχείο της λίστας διευθύνσεων. Περιλαμβάνει:

    - Έναν pointer προς έναν κόμβο τύπου Node από την Linked_list (δηλαδή μια διεύθυνση).
    - Έναν pointer προς τον προηγούμενο κόμβο της Address_list
    - Έναν pointer προς τον επόμενο κόμβο της Address_list (διπλά συνδεδεμένη λίστα).

  - **Node* newNode / pNode* newPnode:** Δημιουργούν και επιστρέφουν νέους κόμβους για τις αντίστοιχες λίστες. Περιλαμβάνεται δέσμευση μνήμης και (στην περίπτωση του Node) αντιγραφή της συμβολοσειράς.

- **Linked_list* newList / Address_list* newAddressList:** Δεσμεύουν δυναμική μνήμη και αρχικοποιούν τις αντίστοιχες λίστες με μηδενικούς (NULL) δείκτες κεφαλής και ουράς.
- **void emplace_back / void insert_address:** Εισάγουν έναν νέο κόμβο στο τέλος της λίστας και ενημερώνουν τον δείκτη της ουράς, ώστε να δείχνει στον νέο κόμβο.
- **bool find_match / pNode* check_address:** Υλοποιούν γραμμική αναζήτηση μέσα στις λίστες. Λαμβάνουν ως είσοδο μια συμβολοσειρά και την συγκρίνουν με τα δεδομένα κάθε κόμβου. Επιστρέφουν αντίστοιχα:
  - true αν βρεθεί ταύτιση ή false αν όχι
  - Τον pointer στον κόμβο (pNode*) που αντιστοιχεί στην διεύθυνση του ζητούμενου Node, ή NULL αν δεν βρεθεί.
- **void delete_list / void delete_address_list**: Διαγράφουν όλους τους κόμβους των αντίστοιχων λιστών και απελευθερώνουν τη δυναμική μνήμη που καταλαμβάνουν, αποτρέποντας memory leaks.
- **void delete_address_node:** Διαγράφει έναν συγκεκριμένο κόμβο από την Address_list και ανανεώνει σωστά τους δείκτες prev και next των γειτονικών κόμβων, καθώς και τους δείκτες κεφαλής ή/και ουράς εάν είναι απαραίτητο.
- **void print_list / print_adr_list:** Χρησιμοποιούνται για debugging. Εκτυπώνουν τα περιεχόμενα των λιστών.

Επισημαίνεται ότι οι δυο λίστες αποτελούνται από έναν pointer στην κεφαλή (για να την διατρέχουμε με ευκολία) και έναν pointer στην ουρά (για εύκολη προσθήκη στοιχείων)

Ακόμα, η Address List δεν είναι τίποτε άλλο από μία διπλά συνδεδεμένη λίστα (διότι θέλουμε να κάνουμε και διαγραφές συγκεκριμένων κόμβων, κάτι που δε χρειάζεται στην Linked List καθώς δε διαγράφουμε ποτέ κάποιο ID)

Τέλος, και οι δυο λίστες έχουν συναρτήσεις οι οποίες υλοποιούν τις ιδίες διαδικασίες απλώς επειδή αλλάζει ο τύπος δεδομένων που αποθηκεύουν χρειαζόμαστε και διαφορετικές συναρτήσεις.

Ερώτημα 1

Συντακτικός ορισμός γραμματικής σε BNF

<input> ::= <myhtml_file>

<myhtml_file> ::= MYHTML_OPEN <head> <body> MYHTML_CLOSE
| MYHTML_OPEN <body> MYHTML_CLOSE

<head> ::= HEAD_OPEN <head_title_section> < head_meta_section> HEAD_CLOSE

<head_title_section> ::= TITLE_OPEN <text> TITLE_CLOSE

<head_meta_section> ::= ε | <head_meta_section> <meta_tag>

<meta_tag> ::= META_OPEN <meta_tag_attributes> TAG_CLOSE

<meta_tag_attributes> ::= < attr_name> <attr_content> | <attr_charset>

<body> ::= BODY_OPEN < body_tags> BODY_CLOSE

<body_tags> ::= ε | < body_tags> <p_tag>
| <body_tags> <a_tag>
| <body_tags> <img_tag>
| <body_tags> <form_tag>
| <body_tags> <div_tag>

<p_tag> ::= P_OPEN <p_tag_attributes> TAG_CLOSE P_CLOSE
| P_OPEN <p_tag_attributes> TAG_CLOSE <text> P_CLOSE

<p_tag_attributes> ::= <attr_id>
| <attr_id> <attr_style>
| <sttr_style> <attr_id>

<a_tag> ::= <a_tag_section> A_CLOSE
| <a_tag_section> <img_tag> <text> A_CLOSE
| <a_tag_section> <text> <img_tag> A_CLOSE
| <a_tag_section> <text> A_CLOSE

<a_tag_section> ::= A_OPEN <a_tag_attributes> TAG_CLOSE

<a_tag_attributes> ::= <attr_id> <attr_href>
| <attr_href> <attr_id>

<img_tag> ::= IMG_OPEN <img_attributes> TAG_CLOSE

<img_attributes> ::=
| <img_attributes> <attr_src>
| <img_attributes> <attr_alt>
| <img_attributes> <attr_id>
| <img_attributes> <attr_height>
| <img_attributes> <attr_width>

<form_tag> ::= FORM_OPEN <form_attributes> TAG_CLOSE <form_children>
FORM_CLOSE

<form _attributes> ::= <attr_id>
| <attr_id> <attr_style>
| <attr_style> <attr_id>
| <attr_id> <attr_checkboxes>
| <attr_id> <attr_checkboxes> <attr_style>
| <attr_id> <attr_style> <attr_checkboxes>
| <attr_style> <attr_checkboxes> <attr_id>
| <attr_style> <attr_id> <attr_checkboxes>

<form_children> ::= <input_tag>
| <label_tag>
| <form_children> <input_tag>
| <form_children> <label_tag>

<input_tag> ::= INPUT_OPEN <input_attributes> TAG_CLOSE

<input_attributes> ::=
| <input_attributes> <attr_type>
| <input_attributes> <attr_id>
| <input_attributes> <attr_style>
| <input_attributes> <attr_value>

<label_tag> ::= LABEL_OPEN <label_attributes> TAG_CLOSE <text> LABEL_CLOSE

<label_attributes> ::= <attr_id> <attr_for>
| <attr_for> <attr_id>
| <attr_id> <attr_for> <attr_style>
| <attr_id> <attr_style> <attr_for>
| <attr_style> <attr_id> <attr_for>
| <attr_style> <attr_for> <attr_id>

<div_tag> ::= DIV_OPEN <div_attributes> TAG_CLOSE <div_children> DIV_CLOSE

<div_attributes> ::= <attr_id> <attr_style>
| <attr_style> <attr_id>

<div_children> ::=
| <div_children> <p_tag>
| <div_children> <a_tag>
| <div_children> <img_tag>
| <div_children> <form_tag>

<attr_name> ::= ATTR_NAME EQUALS QUOTED_STRING

<attr_content> ::= ATTR_CONTENT EQUALS QUOTED_STRING

<attr_charset> ::= ATTR_CHARSET EQUALS QUOTED_STRING

<attr_id> ::= ATTR_ID EQUALS QUOTED_STRING

<attr_style> ::= ATTR_STYLE EQUALS QUOTED_STRING

<attr_href> ::= ATTR_HREF EQUALS ABSOLUTE_URL
| ATTR_HREF EQUALS QUOTED_STRING
| ATTR_HREF EQUALS FRAGMENTED_URL

<attr_src> ::= ATTR_SRC EQUALS ABSOLUTE_URL
| ATTR_SRC EQUALS QUOTED_STRING

<attr_alt> ::= ATTR_ALT EQUALS QUOTED_STRING

<attr_height> ::=ATTR_HEIGHT EQUALS NUMBER

<attr_width> ::= ATTR_WIDTH EQUALS NUMBER

<attr_type> ::= ATTR_TYPE EQUALS QUOTED_STRING

<attr_for> ::= ATTR_FOR EQUALS QUOTED_STRING

<attr_value> ::= ATTR_VALUE EQUALS QUOTED_STRING

<attr_checkboxes> ::= ATTR_CHECKBOXES EQUALS NUMBER

<text> ::= TEXT
| <text> TEXT

Περιγραφή της υλοποίησης βήμα-βήμα

Αναγνώριση των tags που υπάρχουν στην myHTML

Το πρώτο βήμα για την υλοποίηση του προγράμματος μας ήταν η αναγνώριση των tags που υπάρχουν στην myHTML. Γενικά, παρατηρούμε ότι τα tags έχουν την εξής μορφή:

<tagname attributes>content</tagname> ή

<tagname attributes>

Σημείωση: Το content μπορεί να είναι είτε κείμενο, ή/και άλλα tags ή όταν υπάρχει να είναι και κενό. Το attributes μπορεί επίσης να μην υπάρχει. (Αναλόγως την περιγραφή του κάθε tag στην εκφώνηση).

Κατηγοριοποίηση των tags με βάση το αν έχουν χαρακτηριστικά ή όχι:

Έχουν χαρακτηριστικά	Δεν έχουν χαρακτηριστικά
meta, p, a, img, form, input, label, div	MYHTML, head, title, body,

Κατηγοριοποίηση των tags με βάση αν δέχονται περιεχόμενο ή όχι:

Δέχονται περιεχόμενο	Δεν δέχονται περιεχόμενο
MyHTML, head, title, body, p, a, form, div, label	meta, img, input

Πρέπει τώρα να ορίσουμε τις δεσμευμένες λέξεις που θέλουμε.

Γενικά, ακολουθούμε την εξής διαδικασία:

- Ορίζουμε αρχικά στην bison ένα token με όνομα TAGNAME_START και εάν το tag δέχεται περιεχόμενο (δηλαδή έχει ετικέτα τέλους) ορίζουμε ένα ακόμα token με όνομα TAGNAME_END.
- Έπειτα, πρέπει να γράψουμε την/τις κανονική έκφραση για το TAGNAME_START token. Εάν το tag δέχεται χαρακτηριστικά, για το TAGNAME_START βάζουμε την έκφραση "<tagname" ενώ αν δεν δέχεται, βάζουμε την έκφραση "<tagname>" για απλότητα. Δηλαδή, λαμβάνουμε υπόψη μας στα tags αυτά το γεγονός ότι ακολουθούν τα χαρακτηριστικά του.
- Εφόσον το TAGNAME_END υπάρχει, βάζουμε στην flex την κανονική έκφρασή "</tagname>"

Ορίζουμε επίσης ένα token **TAG_CLOSE** με κανονική έκφρασή το **">"** το οποίο θα χρησιμοποιηθεί από όλα τα tags που διαθέτουν χαρακτηριστικά.

Σημείωση: Το «tagname» το αντικαθιστούμε κάθε φορά με το πραγματικό όνομα του tag. Για παράδειγμα, για το meta tag: tagname = «meta».

Για παράδειγμα, για το MYHTML tag κάνουμε τα παρακάτω:

- Ορίζουμε στην bison το token **MYHTML_OPEN**
- Βάζουμε στην flex τη κανονική έκφραση που πρέπει να αναγνωριστεί για το συγκεκριμένο token, δηλαδή την **"<MYHTML>"**
- Αντίστοιχα φτιάχνουμε και το token **MYHTML_CLOSE** και βάζουμε στην flex την έκφρασή **"</MYHTML>"**

Για το tag meta, κάνουμε τα εξής:

- Ορίζουμε στην bison το token **META_OPEN**
- Βάζουμε στην flex τη κανονική έκφρασή **"<meta"** για το συγκεκριμένο token (Παραλείπουμε προφανώς το τελικό > διότι το meta tag έχει χαρακτηριστικά τα οποία θα λάβουμε υπόψη μας αργότερα)

Ομοίως συνεχίζουμε για όλα τα tags που υπάρχουν στην γλώσσα μας.

Με αυτόν τον τρόπο έχουμε κάνει το πρόγραμμα μας να αναγνωρίζει όλα τα tags που έχει η γλώσσα μας, χωρίς βεβαία χαρακτηριστικά και περιεχόμενο.

Υλοποίηση κανόνων για τα χαρακτηριστικά των tags

Στην προηγούμενη υποενότητα, υλοποιήσαμε τους αρχικούς κανόνες που χρειαζόμαστε ώστε το πρόγραμμα μας να αναγνωρίζει την αρχή και το τέλος ενός tag. Τώρα καλούμαστε να προσθέσουμε τους κανόνες για το περιεχόμενο και τα χαρακτηριστικά που μπορεί να έχουν.

Τα χαρακτηριστικά έχουν τη μορφή **attributename=”content”** ή **attributename=’content’**

Σημείωση: Το content γενικά μπορεί να περιέχει οποιονδήποτε συνδυασμό χαρακτήρων, αριθμών και συμβόλων.

Σημείωση: Εξαίρεση στην μορφή τους αποτελούν τα attributes **height** και **width** τα οποία πρέπει να είναι: **height=integer** και **width=integer**, όπου integer είναι ένας σκέτος ακέραιος αριθμός (χωρίς quotes) και μάλιστα θετικός.

Αρχικά όπως και πριν πρέπει να ορίσουμε δεσμευμένες λέξεις. Για κάθε χαρακτηριστικό (attribute) που έχουμε ακολουθούμε την εξής γενική διαδικασία:

- Ορίζουμε αρχικά στην bison ένα token με όνομα **ATTR_ATTRIBUTENAME**
- Βάζουμε στην flex τη κανονική έκφρασή **"attributename"** για το συγκεκριμένο token

Θα χρειαστούμε επίσης, όπως φαίνεται και από τη μορφή των χαρακτηριστικών μας, ένα ακόμα token με όνομα **EQUALS** και κανονική έκφραση στη flex την **"=".**

Για τις τιμές των attributes ορίζουμε 2 ακόμη tokens με ονόματα **QUOTED_STRING** και **NUMBER** αντίστοιχα.

Για το **NUMBER** στην flex βάζουμε την κανονική έκφρασή: **-?[0-9]+** η οποία αναγνωρίζει ακέραιους αριθμούς (από το μηδέν έως το 9) αρνητικούς και θετικούς. Το ερωτηματικό, στην αρχή της έκφρασης μετρά την παύλα, σημαίνει ότι η παύλα (δηλαδή το σύμβολο για το πλην) μπορεί και να μην υπάρχει, κάτι που συμβαίνει στους θετικούς αριθμούς. Συνεπώς μπορεί να αναγνωρίσει εκφράσεις όπως: -9, 10, 0 κλπ.

Για το **QUOTED_STRING** στην flex βάζουμε τις κανονικές εκφράσεις: **\"[^"]*\"** και **\'[^']*\'** οι οποίες αναγνωρίζουν οτιδήποτε υπάρχει εντός των quotes. Συνεπώς, μπορεί να αναγνωρίσει από εκφράσεις όπως είναι το "string” ή το ‘string2’, ό,τι περιέχεται εντός των εισαγωγικών δηλαδή το string και το string2 αντίστοιχα. Επίσης, επειδή αναγνωρίζει

οτιδήποτε υπάρχει εντός εισαγωγικών, από μία έκφρασή όπως η ‘ " Submit it” ‘ θα αναγνωρίσει το " Submit it”, όπως δηλαδή είναι και το αναμενόμενο.

Το πρόγραμμα μας θέλουμε επίσης να αγνοεί όλους τους χαρακτήρες κενού (spaces, tabs, carriage return) οπότε βάζουμε έναν κανόνα στη flex για αυτό, δηλαδή:

```
[ \t\r]+                    { /* Ignore whitespaces */ }
\n                          { /* Empty */ }
```

Τέλος, δε μένει τίποτε άλλο από το να φτιάξουμε τη γραμματική μας στη bison σύμφωνα πάντα με τη BNF γραμματική που περιγράψαμε σε προηγούμενη ενότητα.

Ενδεικτικά, για το meta tag γράφουμε το εξής στη bison:

```
meta_tag:
    META_OPEN meta_tag_attributes TAG_CLOSE
;

meta_tag_attributes:
    attr_name attr_content
    | attr_charset
;

attr_name:
    ATTR_NAME EQUALS QUOTED_STRING
;

attr_content:
    ATTR_CONTENT EQUALS QUOTED_STRING
;

attr_charset:
    ATTR_CHARSET EQUALS QUOTED_STRING
;
```

Και στη flex:

```
"<meta"                     { return META_OPEN; }
"name"                      { return ATTR_NAME; }
```

```
"content"                    { return ATTR_CONTENT; }
"charset"                    { return ATTR_CHARSET; }

"="                        { return EQUALS; }
\"[^"]*\"                    { return QUOTED_STRING;}
\'[^']*\'                    { return QUOTED_STRING;}

">"                          { return TAG_CLOSE; }
```

Σημείωση: Περισσότερες λεπτομέρειες ως προς τον τελικό κώδικα bison δίνονται σε επόμενη ενότητα

Υλοποίηση κανόνων για το περιεχόμενο των tags

Μέχρι στιγμής, το πρόγραμμα μας αναγνωρίζει την αρχή των tags, το τέλος των tags καθώς και τα χαρακτηριστικά των tags με βάση τους κανόνες που περιγράφονται στην εκφώνηση. Μένει λοιπόν το πρόγραμμα μας να αναγνωρίζει και το περιεχόμενο των tags.

Φτιάχνουμε λοιπόν ένα νέο token με όνομα **TEXT** το οποίο θα αναγνωρίζει οτιδήποτε δεν εμπίπτει σε προηγούμενους (ή και μικρότερους) κανόνες. Ουσιαστικά, λέμε ότι: Οτιδήποτε δεν εμπίπτει σε άλλον κανόνα, αποτελεί κείμενο.

Ο λόγος που υλοποιούμε το token TEXT με αυτόν τον τρόπο είναι διότι μεσα στο περιεχόμενο ενός tag μπορεί να υπάρχουν, εκτός από κείμενο, και άλλα tags όπως για παράδειγμα συμβαίνει με το a tag.

Συνεπώς του δίνουμε την εξής έκφρασή στη flex:

```
.                            { return TEXT; }
```

και στη bison ορίζουμε μια «δομή επανάληψης» (ώστε να μην σταματάει σε ένα μόνο χαρακτήρα):

```
text:
    TEXT
    | text TEXT
;
```

Με αυτόν τον τρόπο τελειώσαμε με το κείμενο στο περιεχόμενο των στοιχείων. Προφανώς επίσης γράφουμε και τους απαραίτητους κανόνες γραμματικής στη bison σύμφωνα πάντα με τη BNF γραμματική μας, ώστε να αναγνωρίζουμε όπως πρέπει το περιεχόμενο των όλων tags, το οποίο δεν είναι απαραίτητο να είναι (μόνο) κείμενο.

Χειρισμός χαρακτήρα αλλαγής γραμμής (newline)

Θέλουμε εάν υπάρχει συντακτικό λάθος στο myHTML αρχείο μας να εμφανίζεται η γραμμή που υπάρχει το πρόβλημα, οπότε χρησιμοποιούμε μια integer μεταβλητή με όνομα lineNumber και αρχική τιμή το 1 και έναν κανόνα στη flex που την αυξάνει όταν αλλάζει η γραμμή:

```
\n                                    { lineNumber++; }
```

Προσθήκη κατάστασης ετικέτας αρχής στον λεκτικό αναλυτή

Μέχρι στιγμής, το πρόγραμμα μας αναγνωρίζει σχεδόν πλήρως τα αρχεία myHTML (Έχουμε προσθέσει κανόνες για όλα τα tags, τα χαρακτηριστικά τους καθώς και για το περιεχόμενο τους).

Το κύριο πρόβλημα μας όμως είναι ότι οι λεκτικοί κανόνες που έχουμε για τις ετικέτες αρχής (δηλαδή τα tokens EQUALS, ATTR_*, QUOTED_STRING, TAG_CLOSE) αναγνωρίζονται ακόμα και σε απλό κείμενο. Κατά συνέπεια, εάν στον τίτλο είχαμε κείμενο που κομμάτι του ή και ολόκληρο τύγχανε να γινόταν match σε κάποιον από αυτό τους κανόνες, λόγω της γραμματικής μας, θα έβγαινε error.

Για παράδειγμα, εάν στον τίτλο μας είχαμε το κείμενο «Principles = Main concepts» ο lexer μας όταν δει το σύμβολο ‘=’ θα επιστρέψει το token EQUALS ενώ εμείς περιμένουμε στον κανόνα text της bison μόνο TEXT tokens.

Το πρόβλημα αυτό λύνεται με την εισαγωγή μιας κατάστασης για τις ετικέτες αρχής (εκεί δηλαδή που βρίσκονται, αν υπάρχουν, τα χαρακτηριστικά του tag). Την κατάσταση αυτή την ονομάζουμε **TAG_START_STATE**.

Σε αυτή την κατάσταση του lexer μπαίνουμε όταν δούμε κάποιο tag που δέχεται χαρακτηριστικά, και μόνο για όσο είμαστε μέσα σε αυτήν, ο lexer αναγνωρίζει τα patterns για τα tokens EQUALS, ATTR_*, QUOTED_STRING, TAG_CLOSE.

Το ενημερωμένο τμήμα της flex είναι:

```
<TAG_START_STATE>"name"                    { return ATTR_NAME; }
<TAG_START_STATE>"content"                 { return ATTR_CONTENT; }
<TAG_START_STATE>"charset"                 { return ATTR_CHARSET; }
<TAG_START_STATE>"id"                      { return ATTR_ID; }
<TAG_START_STATE>"style"                   { return ATTR_STYLE; }
<TAG_START_STATE>"href"                    { return ATTR_HREF; }
<TAG_START_STATE>"src"                     { return ATTR_SRC; }
<TAG_START_STATE>"alt"                     { return ATTR_ALT; }
<TAG_START_STATE>"height"                  { return ATTR_HEIGHT; }
<TAG_START_STATE>"width"                   { return ATTR_WIDTH; }
<TAG_START_STATE>"for"                     { return ATTR_FOR; }
<TAG_START_STATE>"type"                    { return ATTR_TYPE; }
<TAG_START_STATE>"value"                   { return ATTR_VALUE; }

<TAG_START_STATE>"="                       { return EQUALS; }
<TAG_START_STATE>\"[^"]*\"                 { return QUOTED_STRING; }
<TAG_START_STATE>\'[^']*\'                 { return QUOTED_STRING; }

<TAG_START_STATE>-?[0-9]+                  { yylval.num = atoi(yytext);
return NUMBER; }
<TAG_START_STATE>">"                       { yy_pop_state(); return
TAG_CLOSE; }

<*>[ \t\r]+                                { /* Ignore whitespaces */ }
<*>\n                                      { lineNumber++; }
<INITIAL,TAG_START_STATE>.                 { return TEXT; }
```

Στην αρχή του κάθε κανόνα, ορίζουμε σε ποιες καταστάσεις του λεξικού αναλυτή (lexer states) ο κανόνας θα είναι «ενεργός». Εάν βάλουμε το σύμβολο *, τότε ο κανόνας είναι ενεργός σε όλες τις καταστάσεις. Εάν δεν βάλουμε τίποτα, τότε ο κανόνας είναι ενεργός μόνο στην αρχική (INITIAL) κατάσταση.

Για παράδειγμα, ο κανόνας: **<TAG_START_STATE>"value"** **{ return ATTR_VALUE; }** είναι ενεργός μόνο στην κατάσταση TAG_START_STATE.
Αντιθέτως, ο κανόνας: **<*>\n** **{ lineNumber++; }** είναι ενεργός σε όλες τις καταστάσεις.

Προσθήκη υποστήριξης για σχόλια

Το σκεπτικό για τα σχόλια είναι το εξής:

- Εάν δεις **<!--** μπες σε μία κατάσταση σχολίου (COMMENT_STATE)
- Όσο είσαι στην κατάσταση αυτή:
  - Οποιοδήποτε κείμενο δεις, αγνόησε το
  - Εάν δεις **--** βγάλε error
  - Εάν δεις **-->** βγες από την κατάσταση αυτή

Επίσης, λόγω περιορισμού της εκφώνησης, σημειώνεται ότι δεν υποστηρίζονται σχόλια ανάμεσα από χαρακτηριστικά των tags (δηλαδή όσο βρισκόμαστε στην TAG_START_STATE)

Το σχετικό κομμάτι της flex:

```
"<!--"                  { yy_push_state(COMMENT_STATE); }
<COMMENT_STATE>"--"     { return ERROR; }
<COMMENT_STATE>"-->"    { yy_pop_state(); }
<COMMENT_STATE>.        { /*ignore the text in the comment*/ }

<*>[ \t\r]+             { /* Ignore whitespaces */ }
<*>\n                                   { lineNumber++; }
<INITIAL,TAG_START_STATE>.              { return TEXT; }
```

Αρχικός κώδικας λεξικού αναλυτή (Flex)

```
/* Declarations */
%{
    #include "my_html_bison.tab.h"
    #include <string.h>

    int lineNumber = 1;
%}

%option stack

%x TAG_START_STATE COMMENT_STATE

/* Rules */
%%
"<!--"                  { yy_push_state(COMMENT_STATE); }
<COMMENT_STATE>"--"     { return ERROR; }
<COMMENT_STATE>"-->"    { yy_pop_state(); }
<COMMENT_STATE>.        { /*ignore the text in the comment*/ }

"<MYHTML>"   { return MYHTML_OPEN; }
"</MYHTML>"  { return MYHTML_CLOSE; }

"<head>"     { return HEAD_OPEN; }
"</head>"    { return HEAD_CLOSE; }
"<title>"    { return TITLE_OPEN; }
"</title>"   { return TITLE_CLOSE; }
"<meta"      { yy_push_state(TAG_START_STATE); return META_OPEN; }
"<body>"     { return BODY_OPEN; }
"</body>"    { return BODY_CLOSE; }
"<p"         { yy_push_state(TAG_START_STATE); return P_OPEN; }
"</p>"       { return P_CLOSE; }
"<a"         { yy_push_state(TAG_START_STATE); return A_OPEN; }
"</a>"       { return A_CLOSE; }
"<img"       { yy_push_state(TAG_START_STATE); return IMG_OPEN; }
"<form"      { yy_push_state(TAG_START_STATE); return FORM_OPEN; }
"</form>"    { return FORM_CLOSE;}
"<input"     { yy_push_state(TAG_START_STATE); return INPUT_OPEN;}
"<label"     { yy_push_state(TAG_START_STATE); return LABEL_OPEN;}
```

```
"</label>"   { return LABEL_CLOSE; }
"<div"       { yy_push_state(TAG_START_STATE); return DIV_OPEN; }
"</div>"     { return DIV_CLOSE; }

<TAG_START_STATE>"name"                 { return ATTR_NAME; }
<TAG_START_STATE>"content"              { return ATTR_CONTENT; }
<TAG_START_STATE>"charset"              { return ATTR_CHARSET; }
<TAG_START_STATE>"id"                   { return ATTR_ID; }
<TAG_START_STATE>"style"                { return ATTR_STYLE; }
<TAG_START_STATE>"href"                 { return ATTR_HREF; }
<TAG_START_STATE>"src"                  { return ATTR_SRC; }
<TAG_START_STATE>"alt"                  { return ATTR_ALT; }
<TAG_START_STATE>"height"               { return ATTR_HEIGHT; }
<TAG_START_STATE>"width"                { return ATTR_WIDTH; }
<TAG_START_STATE>"for"                  { return ATTR_FOR; }
<TAG_START_STATE>"type"                 { return ATTR_TYPE; }
<TAG_START_STATE>"value"                { return ATTR_VALUE; }

<TAG_START_STATE>"="                    { return EQUALS; }
<TAG_START_STATE>\"[^"]*\"              { return QUOTED_STRING; }
<TAG_START_STATE>\'[^']*\'              { return QUOTED_STRING; }

<TAG_START_STATE>-?[0-9]+     { yylval.num = atoi(yytext); return
NUMBER; }
<TAG_START_STATE>">"          { yy_pop_state(); return TAG_CLOSE;
}

<*>[ \t\r]+                   { /* Ignore whitespaces */ }
<*>\n                         { lineNumber++; }
<INITIAL,TAG_START_STATE>.    { return TEXT; }
%%
```

Αρχικός κώδικας συντακτικού αναλυτή (Bison)

```
/* Declarations */
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <stdbool.h>

    extern int yylex(void);
    extern int yyerror(char* s);
    extern int lineNumber;
    extern FILE *yyin;

    bool parse_success = true;
%}

%code requires {
    struct ImgAttributes {
        int has_id;
        int has_src;
        int has_alt;
        int has_type;
        int has_width;
        int has_height;
    } typedef ImgAttributes;

    struct InputAttributes {
        int has_id;
        int has_type;
        int has_style;
        int has_value;
    } typedef InputAttributes;

    void validateImgAttrs(ImgAttributes attrs);
    void validateInputAttrs(InputAttributes attrs);
}

%union {
    ImgAttributes imgAttrs;
    InputAttributes inputAttrs;
```

```
    int num;
}

%token MYHTML_OPEN MYHTML_CLOSE

%token HEAD_OPEN HEAD_CLOSE BODY_OPEN BODY_CLOSE
%token TITLE_OPEN TITLE_CLOSE META_OPEN P_OPEN P_CLOSE A_OPEN
A_CLOSE
%token IMG_OPEN FORM_OPEN FORM_CLOSE LABEL_OPEN LABEL_CLOSE
INPUT_OPEN DIV_OPEN DIV_CLOSE

%token ATTR_NAME ATTR_CONTENT ATTR_CHARSET ATTR_ID ATTR_STYLE
ATTR_HREF
%token ATTR_SRC ATTR_ALT ATTR_HEIGHT ATTR_WIDTH ATTR_FOR ATTR_TYPE
ATTR_VALUE

%token QUOTED_STRING EQUALS TAG_CLOSE
%token<num> NUMBER
%token TEXT ERROR

%type<imgAttrs> img_attributes
%type<inputAttrs> input_attributes

/* Rules */
%%
input:
    myhtml_file
;
myhtml_file:
    MYHTML_OPEN head body MYHTML_CLOSE
    | MYHTML_OPEN body MYHTML_CLOSE
;
head:
    HEAD_OPEN head_title_section head_meta_section HEAD_CLOSE
;
head_title_section:
    TITLE_OPEN text TITLE_CLOSE
    | TITLE_OPEN TITLE_CLOSE
;
head_meta_section:
```

```
    /* empty */
    | head_meta_section meta_tag
;
meta_tag:
    META_OPEN meta_tag_attributes TAG_CLOSE
;
meta_tag_attributes:
    attr_name attr_content
    | attr_charset
;
body:
    BODY_OPEN body_tags BODY_CLOSE
;
body_tags:
    /* empty */
    | body_tags p_tag
    | body_tags a_tag
    | body_tags img_tag
    | body_tags form_tag
    | body_tags div_tag
;
p_tag:
    P_OPEN p_tag_attributes TAG_CLOSE /* empty */ P_CLOSE
    | P_OPEN p_tag_attributes TAG_CLOSE text P_CLOSE
;
p_tag_attributes:
    attr_id
    | attr_id attr_style
    | attr_style attr_id
;

a_tag:
    a_tag_section /* empty */ A_CLOSE
    | a_tag_section img_tag text A_CLOSE
    | a_tag_section text img_tag A_CLOSE
    | a_tag_section text A_CLOSE
;
a_tag_section:
    A_OPEN a_tag_attributes TAG_CLOSE
;
```

```
a_tag_attributes:
    attr_id attr_href
    | attr_href attr_id
;
img_tag:
    IMG_OPEN img_attributes TAG_CLOSE {
        validateImgAttrs($2);
    }
;
img_attributes:
    {
        $$ = (ImgAttributes){0, 0, 0, 0, 0, 0};
    }
   | img_attributes attr_src {
        $$ = $1;
        $$.has_src++;
   }
   | img_attributes attr_alt {
        $$ = $1;
        $$.has_alt++;
   }
   | img_attributes attr_id {
        $$ = $1;
        $$.has_id++;
   }
   | img_attributes attr_height {
        $$ = $1;
        $$.has_height++;
   }
   | img_attributes attr_width {
        $$ = $1;
        $$.has_width++;
   }
;
form_tag:
    FORM_OPEN form_attributes TAG_CLOSE form_children FORM_CLOSE
;
form_attributes:
    attr_id
    | attr_id attr_style
```

```
;
form_children:
    input_tag
    | label_tag
    | form_children input_tag
    | form_children label_tag
;
input_tag:
    INPUT_OPEN input_attributes TAG_CLOSE {
        validateInputAttrs($2);
    }
;
input_attributes:
    {
        $$ = (InputAttributes){0, 0, 0, 0};
    }
    | input_attributes attr_type {
        $$ = $1;
        $$.has_type++;
   }
    | input_attributes attr_id {
        $$ = $1;
        $$.has_id++;
   }
    | input_attributes attr_style {
        $$ = $1;
        $$.has_style++;
   }
    | input_attributes attr_value {
        $$ = $1;
        $$.has_value++;
   }
;
label_tag:
    LABEL_OPEN label_attributes TAG_CLOSE text LABEL_CLOSE
;
label_attributes:
    attr_id attr_for
    | attr_for attr_id
    | attr_id attr_for attr_style
```

```
    | attr_id attr_style attr_for
    | attr_style attr_id attr_for
    | attr_style attr_for attr_id
;
div_tag:
    DIV_OPEN div_attributes TAG_CLOSE div_children DIV_CLOSE
;
div_attributes:
    attr_id attr_style
    | attr_style attr_id
;
div_children:
    /* empty */
    | div_children p_tag
    | div_children a_tag
    | div_children img_tag
    | div_children form_tag
;
attr_name:
    ATTR_NAME EQUALS QUOTED_STRING
;
attr_content:
    ATTR_CONTENT EQUALS QUOTED_STRING
;
attr_charset:
    ATTR_CHARSET EQUALS QUOTED_STRING
;
attr_id:
    ATTR_ID EQUALS QUOTED_STRING
;
attr_style:
    ATTR_STYLE EQUALS QUOTED_STRING
;
attr_href:
    ATTR_HREF EQUALS QUOTED_STRING
;
attr_src:
    ATTR_SRC EQUALS QUOTED_STRING
;
attr_alt:
```

```
    ATTR_ALT EQUALS QUOTED_STRING
;
attr_height:
    ATTR_HEIGHT EQUALS NUMBER { if($3 <= 0) yyerror("Height must a
positive integer"); }
;
attr_width:
    ATTR_WIDTH EQUALS NUMBER { if($3 <= 0) yyerror("Width must a
positive integer"); }
;
attr_type:
    ATTR_TYPE EQUALS QUOTED_STRING
;
attr_for:
    ATTR_FOR EQUALS QUOTED_STRING
;
attr_value:
    ATTR_VALUE EQUALS QUOTED_STRING
;
text:
    TEXT
    | text TEXT
;
%%

/* C code */
int main(int argc, char** argv) {
    bool inputFromFile = false;

    // Determine if we will be using a file or stdin as input
    if (argc > 1)
        inputFromFile = true;

    // Open the input file, if applicable
    if (inputFromFile) {
        FILE *file = fopen(argv[1], "r");

        if (!file) {
            fprintf(stderr, "Cannot open file %s\n", argv[1]);
            exit(1);
```

```
        }

        yyin = file;
    }

    // Call the bison parser
    yyparse();

    // Show diagnostic message
    if (parse_success) {
        printf("myHTMLParser: Parsing completed successfully and
the file is valid.\n");
    }

    // Close the input file, if applicable
    if (inputFromFile)
        fclose(yyin);

    return 0;
}

void validateImgAttrs(ImgAttributes attrs) {
    if (attrs.has_id != 1 || attrs.has_src != 1 || attrs.has_alt
!= 1) {
        yyerror("img tag requires exactly one each of: id, src,
alt");
    }
    if (attrs.has_width > 1 || attrs.has_height > 1) {
        yyerror("img tag allows at most one each of optional:
width, height");
    }
}

void validateInputAttrs(InputAttributes attrs) {
    if (attrs.has_id != 1 || attrs.has_type != 1) {
        yyerror("input tag requires exactly one each of: id,
type");
    }

    if (attrs.has_value > 1 || attrs.has_style > 1) {
```

```
        yyerror("input tag allows at most one each of optional:
value, style");
    }
}

int yyerror(char* s) {
    printf("\nError: %s in line number %d \n", s, lineNumber);
    parse_success = false;

    exit(1);
}
```

Σημείωση: Επειδή τα χαρακτηριστικά του img tag και του input tag είναι πολλά (σε αριθμό 6 και 4 αντίστοιχα), οι δυνατοί συνδυασμοί είναι πάρα πολλοί και επομένως χρησιμοποιήσαμε κώδικα C για τον έλεγχο τους.

Συγκεκριμένα, φτιάξαμε 2 δομές Struct για τα χαρακτηριστικά του καθενός:

```
    struct ImgAttributes {
        int has_id;
        int has_src;
        int has_alt;
        int has_type;
        int has_width;
        int has_height;
    } typedef ImgAttributes;

    struct InputAttributes {
        int has_id;
        int has_type;
        int has_style;
        int has_value;
    } typedef InputAttributes;
```

Οι 2 συναρτήσεις οι οποίες ελέγχουν την εγκυρότητα των χαρακτηριστικών του κάθε tag:

```
void validateImgAttrs(ImgAttributes attrs) {
    if (attrs.has_id != 1 || attrs.has_src != 1 || attrs.has_alt
!= 1) {
        yyerror("img tag requires exactly one each of: id, src,
alt");
    }
    if (attrs.has_width > 1 || attrs.has_height > 1) {
        yyerror("img tag allows at most one each of optional:
width, height");
    }
}

void validateInputAttrs(InputAttributes attrs) {
    if (attrs.has_id != 1 || attrs.has_type != 1) {
        yyerror("input tag requires exactly one each of: id,
type");
    }

    if (attrs.has_value > 1 || attrs.has_style > 1) {
        yyerror("input tag allows at most one each of optional:
value, style");
    }
}
```

Οι γραμματικοί κανόνες **img_attributes** και **input_attributes** είναι τυπου **ImgAttributes** και **InputAttributes** αντίστοιχα.

Η λογική που ακολουθείται όταν δούμε ένα χαρακτηριστικό είναι να αυξάνουμε μια μεταβλητή που αποθηκεύει το πλήθος των εμφανίσεων του και στο τέλος να ελέγχουμε την εγκυρότητα μέσω των συναρτήσεων validate για κάθε tag.

```
img_attributes:
    {
        $$ = (ImgAttributes){0, 0, 0, 0, 0, 0};
    }
   | img_attributes attr_src {
        $$ = $1;
        $$.has_src++;
   }
   | img_attributes attr_alt {
        $$ = $1;
        $$.has_alt++;
   }
   | img_attributes attr_id {
        $$ = $1;
        $$.has_id++;
   }
   | img_attributes attr_height {
        $$ = $1;
        $$.has_height++;
   }
   | img_attributes attr_width {
        $$ = $1;
        $$.has_width++;
   }
;



input_attributes:
    {
        $$ = (InputAttributes){0, 0, 0, 0};
    }
    | input_attributes attr_type {
        $$ = $1;
        $$.has_type++;
   }
    | input_attributes attr_id {
        $$ = $1;
        $$.has_id++;
   }
```

```
    | input_attributes attr_style {
        $$ = $1;
        $$.has_style++;
   }
    | input_attributes attr_value {
        $$ = $1;
        $$.has_value++;
   }
;
```

Και τα img, input tags από τις οποίες καλούμε την αντίστοιχη συνάρτηση validate

```
img_tag:
    IMG_OPEN img_attributes TAG_CLOSE {
        validateImgAttrs($2);
    }
;

input_tag:
    INPUT_OPEN input_attributes TAG_CLOSE {
        validateInputAttrs($2);
    }
;
```

Στιγμιότυπα εκτέλεσης του αρχικού προγράμματος

```
Running test: tests/test-error1.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <!--this comment is ok-->
</head>

<--
this
is
also
ok-->

<body>
    <p id="id1" style="color:blue">This is my project.</p>
    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >
    <--this comment is -- wrong-->
</body>
</MYHTML>

Error: syntax error in line number 7
```

```
---------------------------------
Running test: tests/test-error2.txt
<MYHTML>

    <body>

    </body>

    <head>
        <!-- if head exists it must be before body-->
    </head>

</MYHTML>

Error: syntax error in line number 7
```

```
---------------------------------
Running test: tests/test-error3.txt
<MYHTML>
    <head>
         <meta name="description" content="Flex Bison Tutorial">
         <title> This is wrong because title must be the first element in the head sect
ion </title>
    </head>

    <body>
        <!--body can be empty-->
    </body>
</MYHTML>

Error: syntax error in line number 3
```

```
----------------------------------
Running test: tests/test-error4.txt
<MYHTML>
    <head>
        <title> Principles of programming </title>
    </head>

    <!--body tag is mandatory-->
</MYHTML>

Error: syntax error in line number 7
```

```
Running test: tests/test-error5.txt
<MYHTML>
    <head>
        <title> Principles of programming and Compilers </title>
        <meta name="description" content="Flex Bison Tutorial">
    </head>

    <body>
         <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="ceid_logo" alt="Example Image1" height=-8 width=10
0.8 >

            <!--height and width must be a positive integer-->
        </a>
    </body>
</MYHTML>

Error: Height must a positive integer in line number 10

Error: syntax error in line number 10
```

```
----------------------------------
Running test: tests/test-error6.txt
<MYHTML>
    <head>
        <title> Principles of programming and Compilers </title>
        <meta name="description" content="Flex Bison Tutorial">
    </head>

    <body>
         <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image1.jpg" id="tree" alt="Example Image1" height=21 width=57 >
            <img src="image2.jpg" id="vector" alt="Example Image2" height=17 width=41 >

            <!--a tag must only have one image-->
        </a>
    </body>
</MYHTML>

Error: syntax error in line number 11
```

```
---------------------------------
Running test: tests/test-error7.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style=" backround_color: red; color: blue; font_size: 76%; font_fami
ly: calibri; ">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >


    <!-- div is missing and id attribute -->
    <div  style="color:white; font_size:69%;">
        <!-- comment -->
        <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="myimg3211" alt="Example Image" height=9 width=9 >
        </a>
    </div>
</body>
</MYHTML>

Error: syntax error in line number 23
```

```
Running test: tests/test-error8.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style=" backround_color: red; color: blue; font_size: 76%; font_fami
ly: calibri; ">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

    <div id = "div1"  style="color:white; font_size:69%;">
        <!-- comment -->
        <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="myimg3211" alt="Example Image" height=9 width=9 >
        </a>
    </div>

    <!--form doesnt have attr style -->
    <form id="form1" value=9>
        <input type="text" id="label1">
        <label for="label1" id="lol">Name:</label>
    </form>
</body>
</MYHTML>

Error: syntax error in line number 30
```

```
Running test: tests/test-error9.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <div id = "div1"  <!--a comment cant be placed here--> style="color:white; font_siz
e:69%;">
        <!-- comment -->
        <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="myimg3211" alt="Example Image" height=9 width=9 >
        </a>
    </div>

</body>
</MYHTML>

Error: syntax error in line number 9
```

```
---------------------------------
Running test: tests/test-errorA.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style=" backround_color: red; color: blue; font_size: 76%; font_fami
ly: calibri; ">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

    <div id = "div1"  style="color:white; font_size:69%;">
        <!-- comment -->
        <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="myimg3211" alt="Example Image" height=9 width=9 >
        </a>
    </div>

    <form id="form1">
        <input type="text" id="label1" value=46.69 >
        <label for="label1" id="lol">Name:</label>
    </form>
</body>
</MYHTML>

Error: syntax error in line number 30
```

```
-----------------------------------
Running test: tests/test-valid1.txt
<MYHTML>
<body>
<!-- comment -->
</body>
</MYHTML>


myHTMLParser: Parsing completed successfully and the file is valid.
```

```
Running test: tests/test-valid2.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
</head>
<body>
    <p id = "par1" style="color:blue">This is my project.</p>
    <a id="hyper" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <a id="hyper" href= "https://arxes.ceid.gr">
        <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=9 >
        Here comes an image. <!-- Comment -->
    </a>

    <!-- Single line comment -->
</body>
</MYHTML>


myHTMLParser: Parsing completed successfully and the file is valid.
```

```
Running test: tests/test-valid3.txt
<MYHTML>
<head>
    <title>Principles <-- Comment --> of Programming Languages & Compilers</title>
</head>
<body>
    <!-- Single line comment -->
</body>
</MYHTML>


myHTMLParser: Parsing completed successfully and the file is valid.
```

Ερώτημα 2

Ερώτημα Α

Από το Μέρος Α, έχουμε ορίσει τη δομή του title tag. Για να περιορίσουμε το πλήθος χαρακτήρων στο περιεχόμενό του, χρησιμοποιούμε μια μεταβλητή title_size στη Flex, η οποία δηλώνεται επίσης ως extern στη Bison, η οποία κρατάει το πλήθος των χαρακτήρων και μια ακόμα κατάσταση, την TITLE_STATE.

Μπαίνοντας στο TITLE_STATE, ο κανόνας [^<]+ αναγνωρίζει συνεχόμενο κείμενο μέχρι να εντοπιστεί το χαρακτήρα <, ο οποίος υποδηλώνει πιθανή έναρξη του closing tag και προσθέτει το size του κειμένου που αποθηκεύσαμε στην yytext στην μεταβλητή title_size (title_size = 0 αρχικά). Αν το < αποτελεί μέρος του κειμένου, καλύπτεται από ξεχωριστό κανόνα (<TITLE_STATE>"<") και ο μετρητής title_size αυξάνεται κατάλληλα. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς μέχρι να αναγνωριστεί το </title>, όπου τότε επιστρέφουμε στην αρχική κατάσταση.

Η τελική τιμή της title_size συγκρίνεται στη Bison με το προκαθορισμένο όριο (60). Αν το μέγεθος ξεπερνά αυτό το όριο, καλείται η yyerror με κατάλληλο διαγνωστικό μήνυμα.

Σημείωση: Δε χρειάζεται να μηδενίσουμε την title_size για επόμενη χρήση, καθώς η myHTML επιτρέπει την ύπαρξη μόνο ενός τίτλου, ο οποίος είναι και υποχρεωτικός (εάν βέβαια υπάρχει το head tag).

Αλλαγές στη Flex:

```
<INITIAL,TITLE_STATE>"<!--"  { yy_push_state(COMMENT_STATE); }


"<title>"      { yy_push_state(TITLE_STATE); return TITLE_OPEN; }


<INITIAL,TITLE_STATE>"</title>"    { if (YY_START == TITLE_STATE)
{ yy_pop_state(); } return TITLE_CLOSE; }
<TITLE_STATE>[^<]+  { title_size += strlen(yytext); return TEXT; }
<TITLE_STATE>"<"    { title_size++; }
```

Ο έλεγχος της μεταβλητής title_size στη Bison:

```
head_title_section:
    TITLE_OPEN text TITLE_CLOSE {
        if (title_size > 60) yyerror("Title size is larger than 60 characters");
    }
    | TITLE_OPEN TITLE_CLOSE
;
```

Ερώτημα Β

Αρχικά, όπως αναφέρθηκε στην ενότητα Βιβλιοθήκες, έχουμε δημιουργήσει τη βιβλιοθήκη Linked_list.h. Δημιουργούμε μια λίστα στο τμήμα δηλώσεων του Bison, η οποία θα χρησιμοποιείται για την αποθήκευση των id των στοιχείων.

Η λίστα αρχικοποιείται στη main() μέσω της συνάρτησης newList, η οποία δεσμεύει τη σχετική μνήμη. Καθώς ο lexer διαβάζει το αρχείο μας , όταν εντοπίσει ένα id, επιστρέφει το token QUOTED_STRING, το οποίο είναι τύπου αλφαριθμητικού (char*) όπως αναλύθηκε σε προηγούμενη ενότητα. Στη συνέχεια, η Bison καλεί τη συνάρτηση find_match.

Αν η συνάρτηση επιστρέψει true, τότε το id υπάρχει ήδη στη λίστα, πράγμα που σημαίνει ότι το id δεν είναι μοναδικό επομένως πρέπει να κληθεί η yyerror και να εμφανιστεί σχετικό μήνυμα λάθους. Αλλιώς το id είναι μοναδικό, οπότε καλούμε την emplace_back, η οποία το προσθέτει στη λίστα.

Ο σχετικός κώδικας της Bison:

```
attr_id:
    ATTR_ID EQUALS QUOTED_STRING {
        (!find_match(id_list, $3)) ? emplace_back(id_list, $3) : yyerror("Duplicate id");
        free($3);
    }
;
```

Ερώτημα C

Ορίζουμε μια νέα συνάρτηση validate_url, η οποία δέχεται ως όρισμα το QUOTED_STRING (τύπου char*). Στη συνέχεια, αγνοούμε τυχόν αρχικά κενά χρησιμοποιώντας κατάλληλη επεξεργασία, και ξεκινάμε την ανάλυση του υπόλοιπου URL. Υπάρχουν τρεις βασικές περιπτώσεις:

- **Εσωτερική αναφορά μέσω id:** Αν ο πρώτος μη-κενός χαρακτήρας είναι #, τότε ο χρήστης αναφέρεται σε id στοιχείου του αρχείου myHTML. Σε αυτή την περίπτωση, απομονώνουμε το id και καλούμε τη συνάρτηση find_match. Αν επιστρέψει false, σημαίνει πως το id δεν έχει δηλωθεί πουθενά, και άρα καλείται η yyerror με κατάλληλο μήνυμα. Αν επιστρέψει true, συνεχίζουμε κανονικά.
- **Absolute URL:** Αν το URL ξεκινά με http:// ή https://, τότε θεωρείται absolute και καλούμε τη συνάρτηση _check_full_relative_url για περαιτέρω έλεγχο εγκυρότητας.
- **Σχετική URL:** Αν δεν ισχύει καμία από τις παραπάνω περιπτώσεις, θεωρούμε ότι πρόκειται για σχετική URL, και πάλι καλείται η _check_full_relative_url.

Η _check_full_relative_url είναι η βοηθητική συνάρτηση που ελέγχει αν υπάρχουν κενά σε σύνδεσμο URL (επιτρέπονται μόνο τα %20 ή + ως υποκατάστατα κενού) και στη συνέχεια απαγορεύει τη χρήση των :// και // μέσα στη διεύθυνση, εμφανίζοντας αντίστοιχα διαγνωστικά μηνύματα αν παραβιαστούν οι κανόνες.

Η βασική συνάρτηση validate_url καλείται μέσα στον κανόνα του href στη Bison, ώστε κάθε φορά που διαβάζεται μια τέτοια τιμή, να πραγματοποιείται αυτόματα ο απαραίτητος έλεγχος εγκυρότητας.

Ερώτημα D

Ακολουθήσαμε την ίδια διαδικασία με το ερώτημα Γ, μόνο που τώρα καλούμε την validate_url στον κανόνα του src στη Bison.

Ερώτημα Ε

Στον συντακτικό κανόνα για το attribute type, πραγματοποιούμε εναν απλό έλεγχο της τιμής του QUOTED_STRING μέσω της strcmp, ώστε να επιβεβαιωθεί ότι αντιστοιχεί σε μία από τις επιτρεπόμενες, σύμφωνα με την εκφώνηση, τιμές: "**text**", "**checkbox**", "**radio**" ή "**submit**".

Αν η τιμή είναι "**submit**", τότε αυξάνεται ο μετρητής **submit_input_count** (ο οποίος έχει αρχικά τιμή 0) και ταυτόχρονα η boolean μεταβλητή **is_input_submit_last** (η οποία έχει αρχική τιμή false) τίθεται σε true. Αντίθετα, όταν αναγνωρίζεται οποιοδήποτε άλλο type, η μεταβλητή **is_input_submit_last** επαναφέρεται σε false.

Στη συνέχεια, κατά την ολοκλήρωση του form tag, εφαρμόζεται έλεγχος εγκυρότητας: Εφόσον επιτρέπεται μόνο ένα input tag τύπου submit, ελέγχουμε ότι ο μετρητής **submit_input_count** είναι ακριβώς ίσος με 1, διαφορετικά εμφανίζεται σφάλμα μέσω yyerror. Επιπλέον, η μεταβλητή **is_input_submit_last** χρησιμοποιείται για να διασφαλιστεί ότι το submit είναι το τελευταίο input στο εσωτερικό του form, όπως ορίζει η εκφώνηση.

Έτσι, διασφαλίζεται ότι κάθε form tag περιέχει ένα και μοναδικό **submit** και ότι αυτό τοποθετείται τελευταίο μέσα στη δομή.

Ο σχετικός κώδικας της Bison στον οποίο γίνονται οι έλεγχοι:

```
form_tag:
    FORM_OPEN form_attributes TAG_CLOSE form_children FORM_CLOSE {
        if (submit_input_count > 1) {
            yyerror("Only one submit input tag is allowed per
form");
        }
        if (submit_input_count == 1 && is_input_submit_last ==
false) {
            yyerror("Submit input tag must be the last input tag
in a form");
        }
        submit_input_count = 0;
    }
;
```

```
attr_type:
    ATTR_TYPE EQUALS QUOTED_STRING {
        if (strcmp($3, "text") != 0 && strcmp($3, "checkbox") != 0
&&
            strcmp($3, "radio") != 0 && strcmp($3, "submit") != 0)
{
            yyerror("Invalid input type value. Allowed: text,
checkbox, radio, submit");
        }
        if (strcmp($3, "submit") == 0) {
            submit_input_count++;
            is_input_submit_last = true;
        } else {
            is_input_submit_last = false;
        }
        free($3);
    }
;
```

Ερώτημα F

Παρατηρούμε ότι τα for attributes των <label> tags πρέπει να συνδέονται αποκλειστικά με ids από <input> tags. Συνεπώς, είναι κρίσιμο να μπορούμε να διαφοροποιούμε τα ids των <input> από εκείνα άλλων στοιχείων και, ιδανικά, να μπορούμε να τα διατρέχουμε αποδοτικά. Τη λύση σε αυτό παρέχει η δεύτερη λίστα που έχει υλοποιηθεί στον header Linked_list.h, η οποία είναι η address_list (βλέπε ενότητα Βιβλιοθήκες).

Τροποποιούμε, λοιπόν, τον κανόνα του id αποκλειστικά για τα input attributes (από input_attribute attr_id σε input_attributes ATTR_ID EQUALS QUOTED_STRING) και όπως πριν, εκτελούμε τον έλεγχο μοναδικότητας και προσθέτουμε το id στην ουρά της id_list.

Επιπλέον, προσθέτουμε τη διεύθυνση του νέου κόμβου της id_list στη address_list (input_addresses), του οποίου η διεύθυνση είναι η νέα διεύθυνση της ουράς της id_list, καθώς αυτός ο κόμβος αφορά id ενός <input> tag.

Όταν εντοπιστεί ένα for attribute, διατρέχουμε τη address_list καλώντας τη συνάρτηση check_address. Αν η συνάρτηση επιστρέψει NULL, σημαίνει ότι το id που δηλώθηκε στο for δεν αντιστοιχεί σε κανένα <input> και συνεπώς καλείται η yyerror. Διαφορετικά, η συνάρτηση επιστρέφει τη διεύθυνση που δείχνει στον σχετικό κόμβο της id_list, τον οποίο

διαγράφουμε από την address_list με τη delete_address_node, ώστε να διασφαλίσουμε ότι κάθε for αντιστοιχεί σε ένα και μόνο input id.

Σημείωση: Ο κόμβος δεν αφαιρείται από την id_list, οπότε αν επιχειρηθεί ξανά η δήλωση του ίδιου id, ο έλεγχος μοναδικότητας θα λειτουργήσει όπως περιγράφηκε στο Ερώτημα Β, διασφαλίζοντας ότι δύο διαφορετικά for δεν μπορούν να αντιστοιχούν στο ίδιο id.

Ο σχετικός κώδικας Bison που υλοποιεί τους ελέγχους είναι ο εξής:

```
attr_for:
    ATTR_FOR EQUALS QUOTED_STRING {
        // Get the address that links our for attribute with the
id of an input
        pNode* temp=check_address(input_addresses,$3);

        // If its null it means that for either has  a duplicate
input id value or none
        if (temp == NULL )
            yyerror("for attribute must be linked uniquely with an
input tag's id");
        else
            delete_address_node(input_addresses,temp);
        free($3);
    }
;


input_attributes:
    {
        $$ = (InputAttributes){0, 0, 0, 0};
    }
    | input_attributes attr_type {
        $$ = $1;
        $$.has_type++;
    }
    // Since this is a special case we do it this way so we can
compare it
    | input_attributes ATTR_ID EQUALS QUOTED_STRING {
        $$ = $1;
        $$.has_id++;
```

```
        (!find_match(id_list,$4))? emplace_back(id_list,$4) :
yyerror("Duplicate id");

        Node* temp=id_list->tail;
        insert_address(input_addresses,temp);
        free($4);
   }
    | input_attributes attr_style {
        $$ = $1;
        $$.has_style++;
   }
    | input_attributes attr_value {
        $$ = $1;
        $$.has_value++;
   }
;
```

Ερώτημα G

Στην Flex ορίζουμε νέοι κανόνες για τις δεσμευμένες λέξεις background_color, color, font_family και font_size, οι οποίες εντοπίζονται μέσα σε style attributes. Το περιεχόμενο των style ακολουθεί τη μορφή "property1:value1; property2:value2; ...", όπου κάθε ιδιότητα (property) αντιστοιχεί σε μια συγκεκριμένη τιμή (value), όπως ορίζεται από την εκφώνηση.

Ιδιαίτερη προσοχή απαιτείται στο font-size, όπου η μονάδα px ή % πρέπει να συνοδεύεται από θετικό ακέραιο αριθμό σε αντίθετη περίπτωση, παράγεται σφάλμα.

Επιπλέον, κάθε style attribute οφείλει να περικλείεται με εισαγωγικά, ακόμα και αν περιλαμβάνει μόνο ένα χαρακτηριστικό αν παραληφθούν, η είσοδος θεωρείται συντακτικά λανθασμένη. Καθώς γίνεται η επεξεργασία των χαρακτηριστικών ενός style, χρησιμοποιείται μια δομή StyleCharacteristics με αριθμητικά μέλη για κάθε επιτρεπόμενη ιδιότητα. Κάθε φορά που αναγνωρίζεται ένα χαρακτηριστικό, γίνεται αύξηση (increment κατα 1) του αντίστοιχου πεδίου.

Μετά την ολοκλήρωση της ανάλυσης των style δεδομένων, καλείται η συνάρτηση validateStyle, η οποία εξετάζει τα μέλη της δομής και εφόσον οποιοδήποτε έχει τιμή μεγαλύτερη από 1, σημαίνει ότι η ίδια ιδιότητα ορίστηκε επαναορίστηκε, κάτι που δεν επιτρέπεται. Σε αυτή την περίπτωση καλείται η yyerror με το κατάλληλο διαγνωστικό μήνυμα.

Στην Flex, εισάγουμε δύο νέα lexer states, τα STYLE_STATE, STYLE_QUOTE. Στο STYLE_QUOTE μπαίνουμε μόλις δούμε ένα χαρακτηριστικό τύπου style. Όσο είμαστε στο STYLE_QUOTE, όταν δούμε ένα quote αρχής (") μπαίνουμε στο STYLE_STATE. Από το STYLE_STATE αναλύουμε τις τιμές του style και έπειτα κάνουμε τους απαραίτητους ελέγχους. Τέλος, από την STYLE_STATE βγαίνουμε όταν συναντήσουμε το τελικό quote (“).

Δηλαδή, το STYLE_QUOTE είναι ένα προσωρινό state για μετάβαση στο STYLE_STATE όταν διαπερνάμε το attribute style ώστε να ξεχωρίζουμε το αρχικό quote (“) με το τελικό.

Ο σχετικός κώδικας σε Bison είναι ο εξής:

```
attr_style:
    ATTR_STYLE EQUALS QUOTE style_characteristics QUOTE{
        validateStyle($4);
    }
;

style_characteristics:
    { $$=(StyleCharacteristics) {0,0,0,0}; }
    |style_characteristics BACKROUND_COLOR COLON text SEMICOLON{
        $$=$1;
        $$.has_backround++;
    }
    |style_characteristics COLOR COLON text SEMICOLON{
        $$=$1;
        $$.has_color++;
    }
    |style_characteristics FONT_FAMILY COLON text SEMICOLON{
        $$=$1;
        $$.has_font_family++;
    }
    |style_characteristics FONT_SIZE COLON NUMBER SEMICOLON{
        if ($4 <= 0) yyerror("Font size must be a possitive
integer");
        $$=$1;
        $$.has_font_size++;
    }
;
```

Ο σχετικός κώδικας της Flex είναι:

```
<TAG_START_STATE>"style"                {
yy_push_state(STYLE_QUOTE); return ATTR_STYLE; }

<STYLE_QUOTE>\"                         {
yy_push_state(STYLE_STATE); return QUOTE;}
<STYLE_STATE>"backround_color"          { return BACKROUND_COLOR;
}
<STYLE_STATE>"color"                    { return COLOR; }
<STYLE_STATE>"font_family"              { return FONT_FAMILY; }
<STYLE_STATE>"font_size"                { return FONT_SIZE; }
<STYLE_STATE>":"                        { return COLON; }
<STYLE_STATE>";"                        { return SEMICOLON; }
<STYLE_STATE>\"                         { yy_pop_state();
yy_pop_state(); return QUOTE; }
<STYLE_STATE>-?[0-9]+"%"                { yylval.num =
atoi(yytext); return NUMBER; }
<STYLE_STATE>-?[0-9]+"px"               { yylval.num =
atoi(yytext); return NUMBER; }
```

Ερώτημα 3

Για την υλοποίηση του ζητούμενου στοιχείου που ονομάσαμε **checkbox-count** χρησιμοποιήσαμε μία integer μεταβλητή με όνομα **checkbox_count_value** (με αρχική τιμή -1 η οποία συμβολίζει ότι δεν υπάρχει καθόλου το attribute αυτό) και έναν μετρητή με όνομα **checkbox_count_value** αντίστοιχα (με αρχική τιμή 0).

Η πρώτη μεταβλητή αποθηκεύει την τιμή του χαρακτηριστικού checkbox-count αν υπάρχει (αλλιώς -1) και ο μετρητης checkbox_count_value μετράει πόσα input tags με type checkbox υπαρχουν σε ένα form tag.

Στο κομμάτι της Flex, δημιουργήσαμε ένα token ATTR_CHECKBOXES με παρόμοιο τρόπο με τα προηγούμενα χαρακτηριστικά και στην Bison έναν άλλο κανόνα attr_checkboxes που περιμένει το χαρακτηριστικό να έχει την κατάλληλη δομή.

Ο έλεγχος για το 1ο ζητούμενο, γίνεται στον attr_checkboxes κανόνα της Bison.

Στο form tag, προσθέσαμε τους επιπλέον απαιτούμενους ελέγχους για το 2ο ζητούμενο δηλαδη για να καθορίζει το **checkbox-count** το ακριβές πλήθος των στοιχείων input τύπου checkbox. Συγκεκριμένα:

- Αν δεν υπάρχει το χαρακτηριστικό checkbox-count στο form tag αλλα ο μετρητής checkbox_count_value έχει θετική τιμή, βγάλε error
- Αν υπάρχει το χαρακτηριστικό checkbox-count στο form tag, έλεγξε αν η τιμή του είναι ίση με εκείνη που έχει πάρει ο μετρητής checkbox_count_value και βγάλε το αντίστοιχο μήνυμα σφάλματος εάν είναι απαραίτητο.

Ο σχετικός κώδικας της Flex:

```
<TAG_START_STATE>"checkbox-count"       { return ATTR_CHECKBOXES; }
```

Ο κώδικας της Bison που υλοποιεί τους ελέγχους είναι ο εξής:

```
form_tag:
    FORM_OPEN form_attributes TAG_CLOSE form_children FORM_CLOSE {
        if (submit_input_count > 1) {
            yyerror("Only one submit input tag is allowed per
form");
        }
        if (submit_input_count == 1 && is_input_submit_last ==
false) {
            yyerror("Submit input tag must be the last input tag
in a form");
        }

        if (checkbox_count_value < 0 && checkbox_input_count > 0)
{
            yyerror("checkbox-count attribute missing, but
checkbox input tag found");
        }

        if (checkbox_count_value > 0) {
            if (checkbox_count_value != checkbox_input_count) {
                if (checkbox_input_count == 0) {
                    yyerror("checkbox-count attribute used, but no
checkbox input tag found");
                } else {
                printf("Expected %d checkbox input tags, got %d",
checkbox_count_value, checkbox_input_count);
                yyerror("Checkbox input tags amount mismatch");
                }
            }
        }

        submit_input_count = 0;
        checkbox_input_count = 0;
        checkbox_count_value = -1;
    }
;
```

```
form_attributes:
    attr_id
    | attr_id attr_style
    | attr_style attr_id
    | attr_id attr_checkboxes
    | attr_id attr_checkboxes attr_style
    | attr_id attr_style attr_checkboxes
    | attr_style attr_checkboxes attr_id
    | attr_style attr_id attr_checkboxes
;



attr_checkboxes:
    ATTR_CHECKBOXES EQUALS NUMBER {
        if($3 <= 0) {
            yyerror("checkboxes-count must be a positive
integer");
        }
        checkbox_count_value = $3;
    }
;
```

Οδηγίες μεταγλώττισής του τελικού προγράμματος

Ο πηγαίος κώδικας του προγράμματος μας περιέχει ένα Makefile, στο οποίο υπάρχουν τα εξής targets:

- Το default target (make) για απλή μεταγλώττιση,
- Ένα run target (make run) το οποίο μεταγλώττιζει το πρόγραμμα και το τρέχει με είσοδο το αρχείο example.txt
- Ένα test target (make test) το οποίο μεταγλωτίζει το πρόγραμμα και το τρέχει με είσοδο κάποια test files τα οποία περιέχονται στον φάκελο tests/

Το εκτελέσιμο αρχείο που παράγεται βρίσκεται στη θέση: `build/my_html_parser.out`

Συνεπώς, για να κάνουμε compile το πρόγραμμα και να το τρέξουμε, αρκεί να εκτελέσουμε από το τερματικό την εντολή:

```
make run
```

Σημείωση: Σε περίπτωση που το εργαλείο make δεν υπάρχει για το λειτουργικό σύστημα για αυτοματοποιημένη μεταγλώττιση, οι εντολές είναι οι εξής:

```
flex -o lex.yy.c my_html_flex.l
bison -d -t -o my_html_bison.tab.c my_html_bison.y
gcc -Ilibs/include -lfl lex.yy.c my_html_bison.tab.c
libs/linked_list.c -o my_html_parser
```

Και για εκτέλεση

```
./my_html_parser
```

Στιγμιότυπα εκτέλεσης του τελικού προγράμματος

```
Running parser with example.txt
./build/my_html_parser.out example.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style=" backround_color: red; color: blue; font_size: 76%; font_family: calibri
; ">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

    <div id="div1" style="color:white; font_size:69%; ">
        <!-- comment -->
        <a id="hyper2" href= "../arxes/flex%20bison/hello_world">
            Here comes an image. <!-- comment -->
            <img src="image.jpg" id="myimg321" alt="Example Image" height=9 width=9 >
        </a>
    </div>

    <div id="div2" style="color:white; font_size:69%; ">
        <!-- comment -->
        <a id="hyper3" href="#hyper2">
            Here comes an image.
            <img src="image.jpg" id="myimg3211" alt="Example Image" height=9 width=9 >
        </a>
    </div>

    <!-- Single line comment -->

    <form id="form1" checkbox-count=1>
        <input type="text" id="label1">
        <label for="label1" id="lol">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="name">last Name:</label>
        <input type="text" id="label3">
        <input id="check1" type="checkbox" value="Allow cookies">
    </form>
</body>
</MYHTML>


myHTMLParser: Parsing completed successfully and the file is valid.
```

```
---------------------------------
Running test: tests/test-error1.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id="id1" style="color:blue;">This is my project.</p>

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

   <--this comment is -- wrong-->
</body>
</MYHTML>

Error: syntax error in line number 13
```

```
Running test: tests/test-error2.txt
<MYHTML>
<head>
    <title> </title>
</head>
<body>
    <form id="form1">

    </form>
</body>
</MYHTML>

Error: syntax error in line number 8
```

```
Running test: tests/test-error3.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id="id1" style="color:blue;">This is my project.</p>

    <img id="kalamata" src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

   <!--this comment is correct-->
</body>
</MYHTML>

Error: img tag requires exactly one each of: id, src, alt in line number 11
```

```
Running test: tests/test-error4.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style="color:blue;">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

    <div id="div1" style="color:blue;">
        <!-- comment -->
        <a id="hyper" href= "https://arxes.ceid.gr">
            Here comes an image. <!-- comment -->
            <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=9 >
        </a>
    </div>

    <!-- Single line comment -->

    <form id="form1">
        <label for ="name" id="label1">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="lastname">last Name:</label>
        <input id="submit" type="submit" value='Submit "it" '>
        <input type="text" id="lastname">
    </form>
</body>
</MYHTML>


Error: Duplicate id in line number 25
```

```
Running test: tests/test-error5.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style="backround_color:red; color:black; font_family:italic; font_size:9px; col
or:yellow; ">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>
</body>
</MYHTML>

Error: style attribute allows at most one each of optional: backround_color, color, font_family, f
ont_size in line number 9
```

```
Running test: tests/test-error6.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style="color:blue">This is my project.</p>
    <a id="hyper" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <form id="form1" checkbox-count=9>
        <label for ="name" id="label1">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="lastname">last Name:</label>
        <input type="text" id="lastname">
        <input id="check1" type="checkbox" value="Allow cookies">
        <input id="submit" type="submit" value='Submit "it" '>
    </form>
</body>
</MYHTML>


Error: for attribute must be linked uniquely with an input tag's id in line number 15
```

```
Running test: tests/test-error8.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id = "par1" style="color:blue;">This is my project.</p>
    <a id="hyper1" href="https://arxes.ceid.gr">
        Here comes an image.
    </a>

    <!--
    A wild
    comment appeared.
    -->

    <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=10 >

    <div id="div1" style="color:blue">
        <!-- comment -->
        <a id="hyper2" href= "https://arxes.ceid.gr">
            Here comes an image.
            <img src="image.jpg" id="myimg123" alt="Example Image" height=9 width=9 >
        </a>
    </div>
</body>
</MYHTML>


Error: Duplicate id in line number 25
```

```
Running test: tests/test-error9.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id="par1" style="">This is my project.</p>

    <img src="image.jpg" id="myimg123" alt="Example Image" height =9 width= 10 >

    <div id="div1" style="color:blue">
        <a id="hyper" href="https://arxes.ceid.gr">

        </a>
    </div>

    <form id="form1" checkbox-count=1>
        <input id="check1" type="checkbox" value="Checkbox check?">
        <label for="name" id="label1">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="lastname">last Name:</label>
        <input type="text" id="lastname">
        <input id="check2" type="checkbox" value="Allow cookies">
        <input id="submit" type="submit" value='Submit it'>
    </form>
</body>
</MYHTML>

Error: for attribute must be linked uniquely with an input tag's id in line number 21
```

```
Running test: tests/test-errorA.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id="par1" style="">This is my project.</p>

    <img src="image.jpg" id="myimg123" alt="Example Image" height =9 width= 10 >

    <div id="div1" style="color:blue">
        <a id="hyper" href="https://arxes.ceid.gr">

        </a>
    </div>

    <form id="form1">
        <input type="text" id="label1">
        <label for="label1" id="lol">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="name">last Name:</label>
        <input type="text" id="label3">
        <input id="check1" type="checkbox" value="Allow cookies">
        <input id="submit" type="submit" value='Submit "it" '>
    </form>
</body>
</MYHTML>

Error: checkbox-count attribute missing, but checkbox input tag found in line number 27
```

```
----------------------------------
Running test: tests/test-errorB.txt
<MYHTML>
<head>
    <title>Principles of Programming Languages & Compilers</title>
    <meta name="description" content="Flex Bison Tutorial">
    <meta charset="UTF-8">
    <meta name="author" content="Input your name here">
</head>
<body>
    <p id="par1" style="">This is my project.</p>

    <img src="image.jpg" id="myimg123" alt="Example Image" height =9 width= 10 >

    <div id="div1" style="color:blue backround_color:black;">
        <a id="hyper" href="https://arxes.ceid.gr">

        </a>
    </div>

    <form id="form1">
        <input type="text" id="label1">
        <label for="label1" id="lol">Name:</label>
        <input type="text" id="name">
        <label id="label2" for="name">last Name:</label>
        <input type="text" id="label3">
        <input id="check1" type="checkbox" value="Allow cookies">
        <input id="submit" type="submit" value='Submit "it" '>
    </form>
</body>
</MYHTML>

Error: syntax error in line number 13
```

Τελικός κώδικας και αρχεία εισόδου του προγράμματος

Ο τελικός κώδικας (που περιλαμβάνει την υλοποίηση όλων των ερωτημάτων 1, 2 και 3) βρίσκεται στο αρχείο zip το οποίο επισυνάπτεται μαζί με την αναφορά. Επίσης, τα αρχεία εισόδου που χρησιμοποιήθηκαν στην προηγούμενη ενότητα περιέχονται στο zip αρχείο στον φάκελο tests.

Σημείωση: Στο αρχείο Makefile υπάρχει ο κανόνας test (make test) ο οποίος τρέχει το τελικό πρόγραμμα με όλα τα αρχεία εισόδου.

Παραδοχές κατά τη δημιουργία του προγράμματος

- Στα ερωτήματα C, D κληθήκαμε να προσθέσουμε κανόνες για τα URLs. Εμείς θεωρήσαμε, για λόγους απλότητας, πως τα αποδεκτά absolute URLs είναι της μορφής **https://** ή **http://**. Στην πράξη όμως, θα μπορούσαμε να έχουμε και άλλα πρωτόκολλα όπως για παράδειγμα **ftp://**.