ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Project Flex-Bison 2023

Ομάδα:

Μαρκάκης Κωνσταντίνος

AM: 1084541

E-mail: up1084541@upnet.gr

Εργασία: Ανάπτυξη κώδικα & Συγγραφή αναφοράς

Μελιτσόπουλος Φώτιος

AM: 1084600

E-mail: up1084600@upnet.gr

Εργασία: Testing & Συγγραφή αναφοράς

Τσοπάνογλου Χρυσή

AM: 1084596

E-mail: up1084596@upnet.gr

Εργασία: Testing & Συγγραφή αναφοράς

Περιεχόμενα

1. Γραμματική BNF

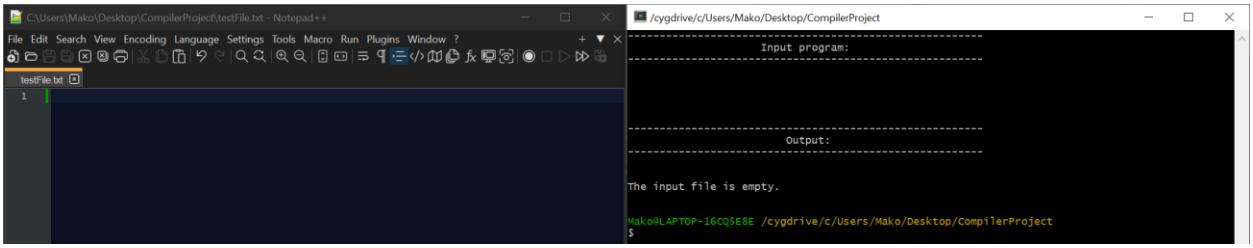
```
<root> ::=
     <LinearLayout> |
     <RelativeLayout>

<single_tag> ::=
     <LinearLayout> |
     <RelativeLayout> |
     <TextView> |
     <ImageView> |
     <Button> |
     <RadioGroup> |
     <ProgressBar>

<content> ::=
     <content> <single_tag> |
     <single_tag> |
     ε

<LinearLayout> ::=
     "<LinearLayout " <linear_layout_attributes> ">"
     <single_tag> <content> "</LinearLayout>"

<RelativeLayout> ::=
     "<RelativeLayout " <relative_layout_attributes> ">"
     <content> "</RelativeLayout>"

<TextView> ::=
     "<TextView " <text_view_attributes> "/>"

<ImageView> ::=
     "<ImageView " <image_view_attributes> "/>"

<Button> ::=
     "<Button " <button_attributes> "/>"


<RadioGroup> ::=
     "<RadioGroup " <radio_group_attributes> ">" <radio_content>
     "</RadioGroup>"

<radio_content> ::=
     <radio_content> <RadioButton> |
     <RadioButton> |
     ε
```

```
<RadioButton> ::=
     "<RadioButton " <radio_button_attributes> "/>"

<ProgressBar> ::=
     "<ProgressBar " <progress_bar_attributes> "/>"

<linear_layout_attributes> ::=
     <layout_width> <layout_height> <id> <orientation>

<relative_layout_attributes> ::=
     <layout_width> <layout_height> <id>

<text_view_attributes> ::=
     <layout_width> <layout_height> <id> <text> <text_color>

<image_view_attributes> ::=
     <layout_width> <layout_height> <id> <src> <padding>

<button_attributes> ::=
     <layout_width> <layout_height> <id> <text> <padding>

<radio_group_attributes> ::=
     <layout_width> <layout_height> <id> <checked_button>
     <button_quantity>

<radio_button_attributes> ::=
     <layout_width> <layout_height> <id> <text>

<progress_bar_attributes> ::=
     <layout_width> <layout_height> <id> <max> <progress>

<layout_width> ::=
     "android:layout_width =\"" <integer> "\"" |
     "android:layout_width =\"match_parent\"" |
     "android:layout_width=\"wrap_content\""

<layout_height> ::=
     "android:layout_height=\"" <integer> "\""

<id> ::=
     "android:id=\"" <string> "\"" |
      ε

<text> ::=
     "android:text=\"" <string> "\""
```

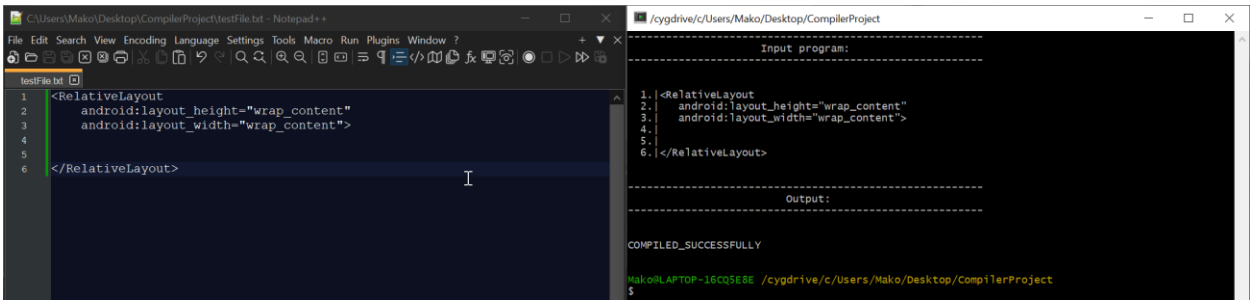
```
<text_color> ::=
     "android:textColor=\"" <string> "\"" |
     ε

<src> ::=
     "android:src=\"" <string> "\""

<padding> ::=
     "android:padding=\"" <integer> "\"" |
     ε

<checked_button> ::=
     "android:checkedButton=\"" <string> "\"" |
     ε

<max> ::=
     "android:max=\"" <integer> "\"" |
     ε

<progress> ::=
     "android:progress=\"" <integer> "\"" |
     ε

<orientation> ::=
     "android:orientation=\"" <string> "\"" |
     ε

<button_quantity> ::=
     "android:button_quantity=\" <integer> \""

<integer> ::=
     <integer> <digit> |
     <digit>

<digit> ::=
     [0-9]

<string> ::=
     <string> <character> |
     <character>

<character> ::=
     [0-9A-Za-z_]
```

2. Λίστα δοκιμαστικών αρχείων(.txt) και αποτελέσματα(screenshots)

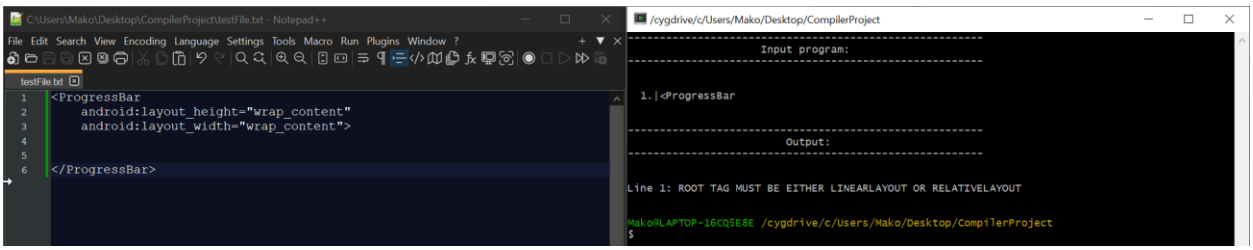
Test_0 (Κενό αρχείο εισόδου - Successful)



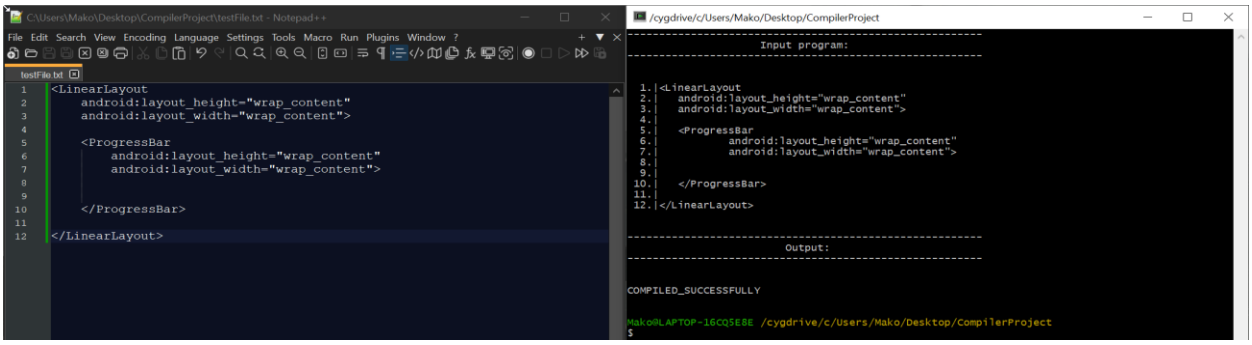
Test_1 (Έλεγχος Root Tag - Successful)



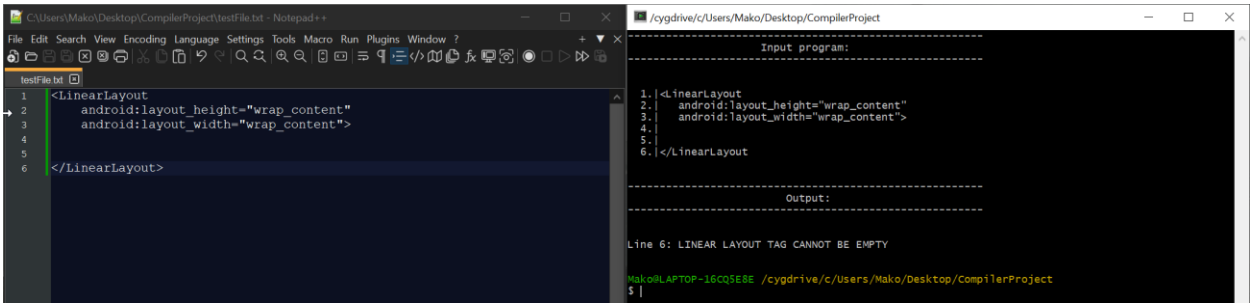
Test_2 (Έλεγχος Root Tag - Error)



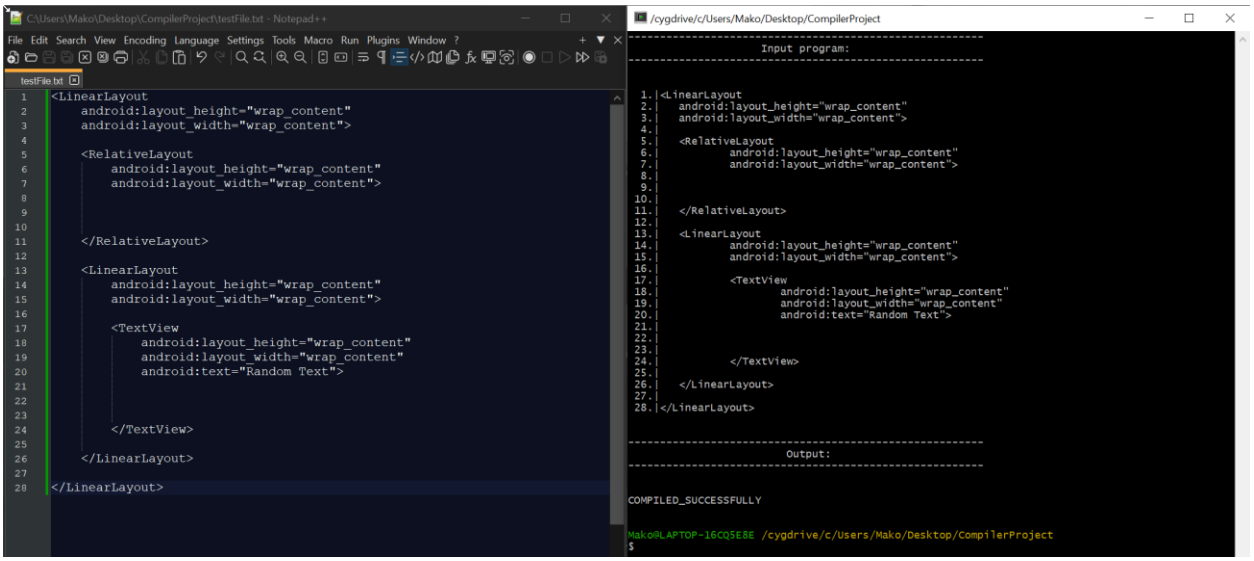
Test_3 (LinearLayout με περιεχόμενο - Successful)



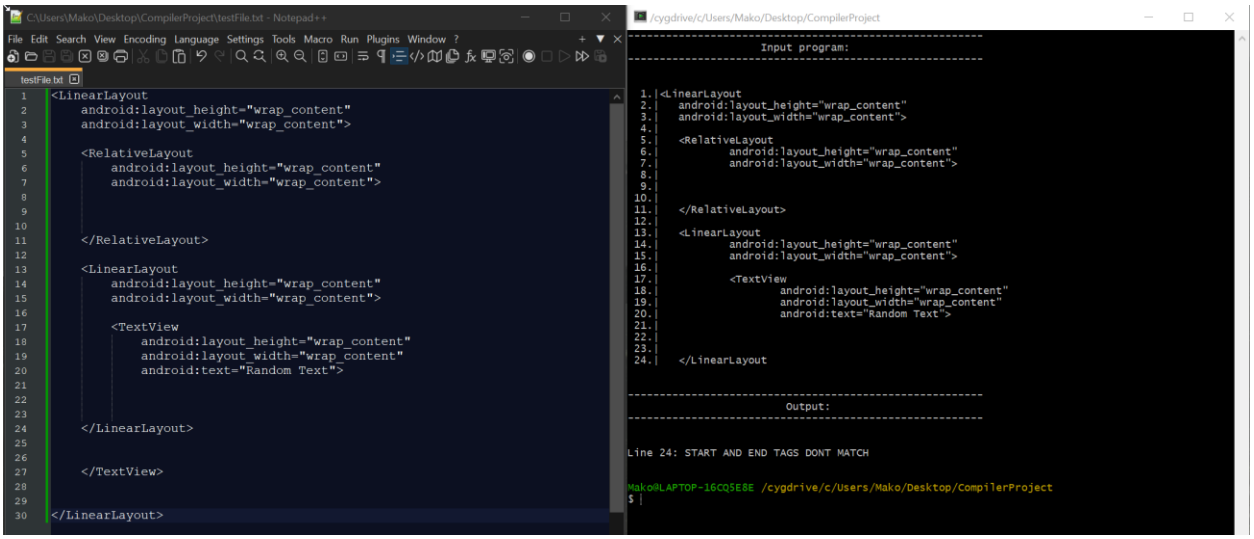
Test_4 (LinearLayout χωρίς περιεχόμενο - Error)



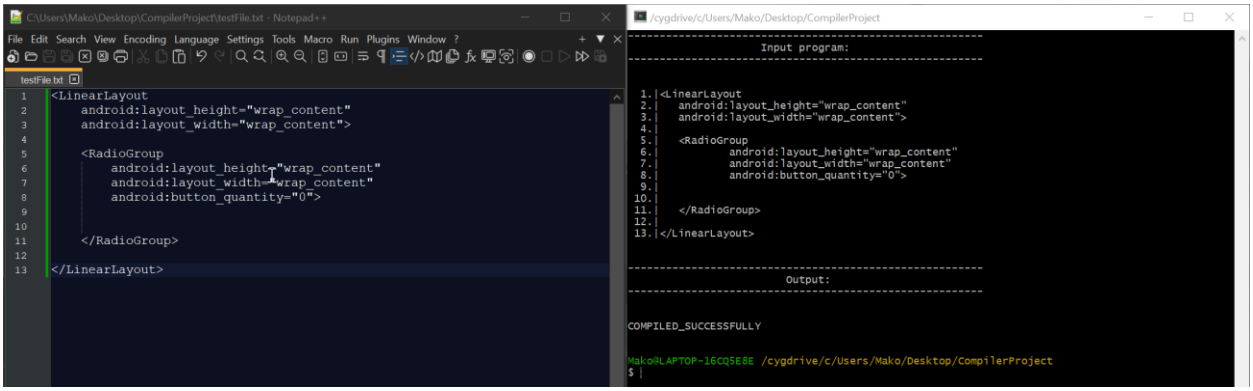
Test_5 (Εμφώλευση - Successful)



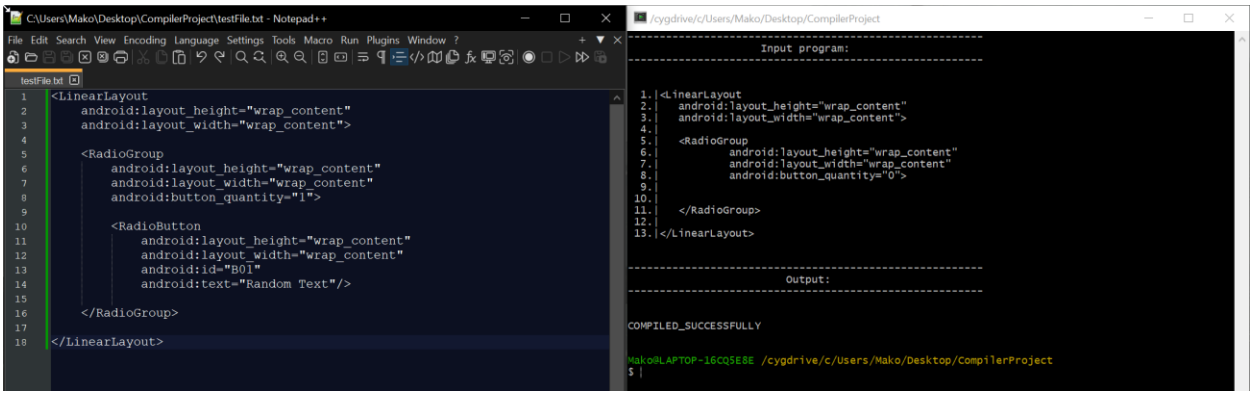
Test_6 (Εμφώλευση - Error)



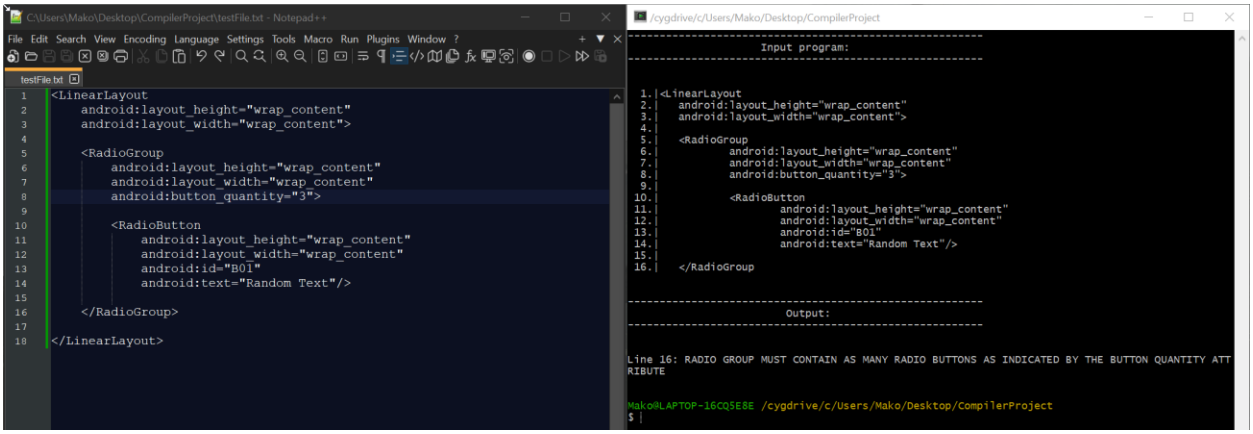
Test_7 (Άδειο RadioGroup - Successful)



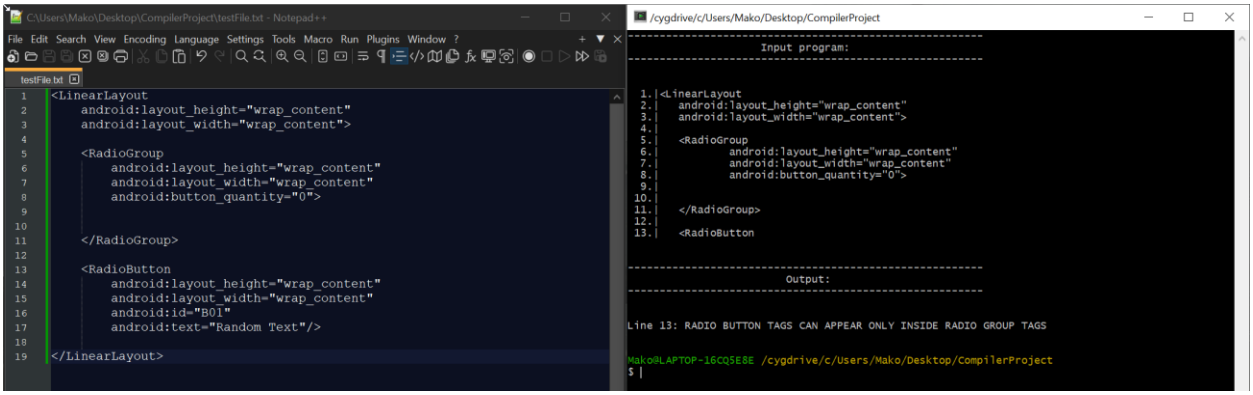
Test_8 (RadioGroup με RadioButton - Successful)



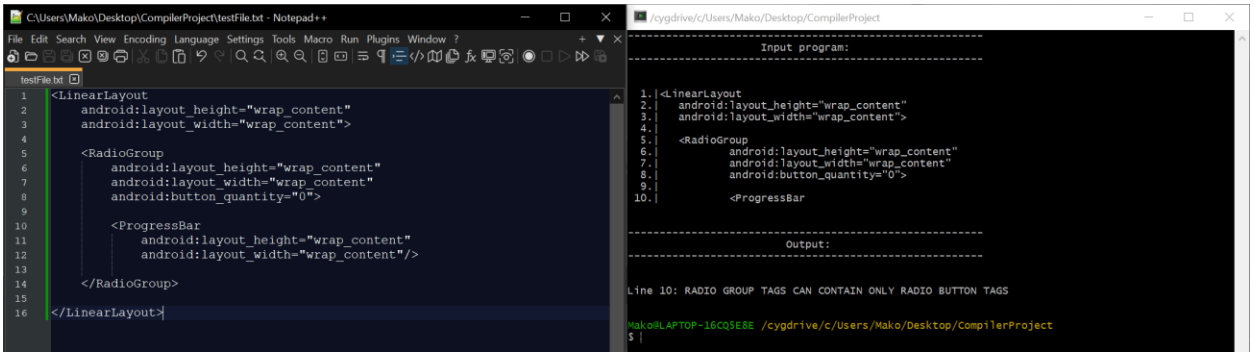
Test_9 (RadioGroup με λάθος πλήθος RadioButton - Error)



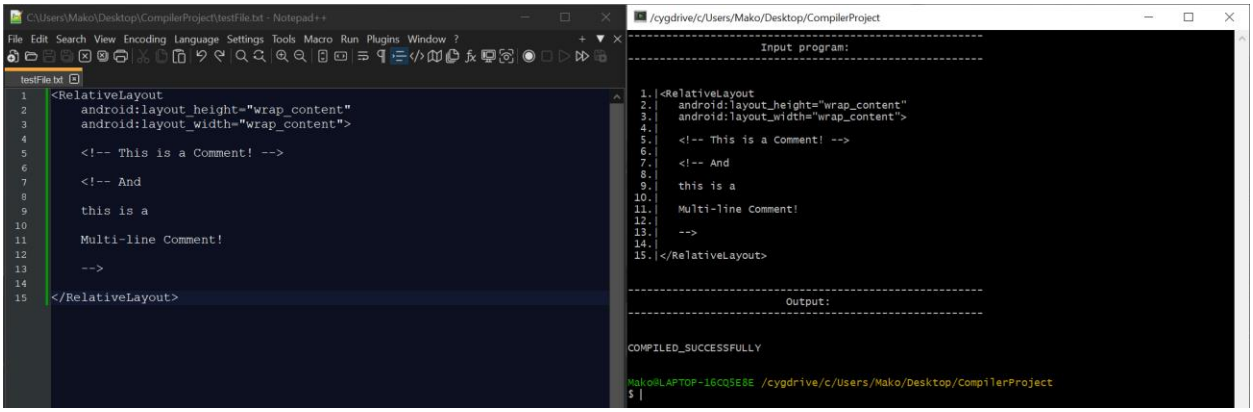
Test_10 (RadioButton έξω από RadioGroup - Error)



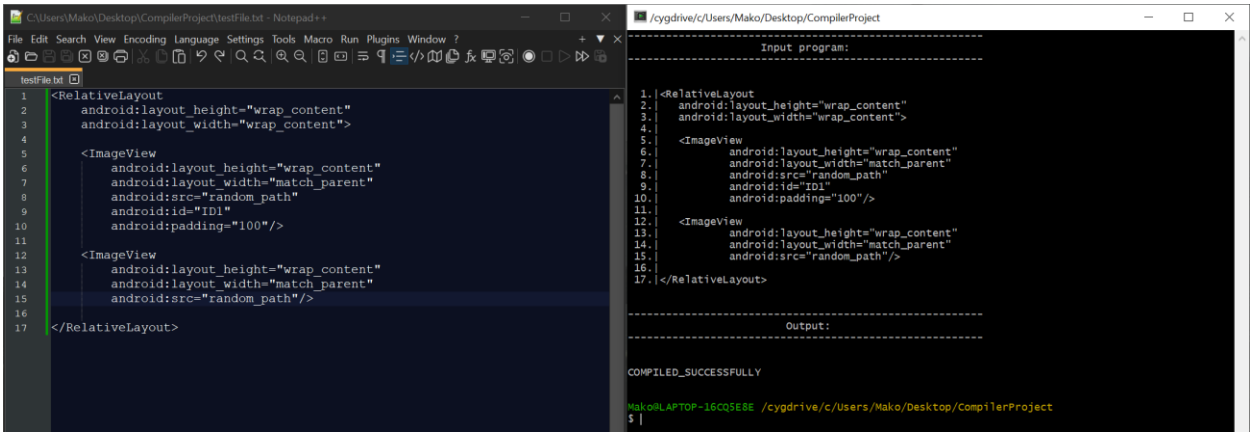
Test_11 (Διαφορετικό Tag εντός RadioGroup - Error)



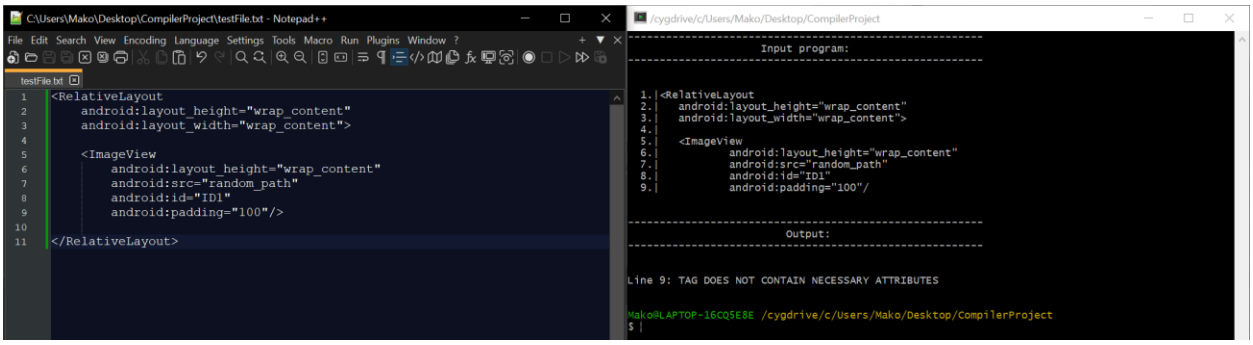
Test_12 (Απλό και Multiline Comment)



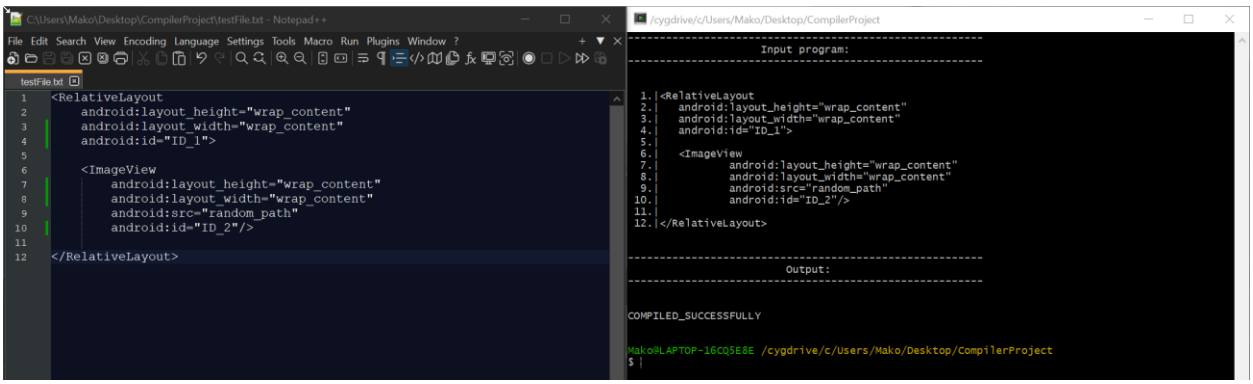
Test_13 (Tags που περιέχουν όλα τα υποχρεωτικά Attributes - Successful)



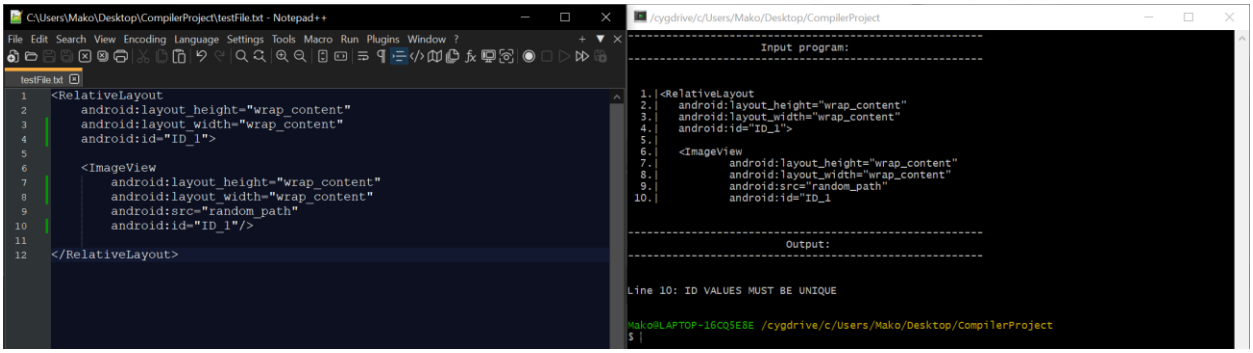
Test_14 (Tag που του λύπει κάποιο υποχρεωτικό Attribute - Error)



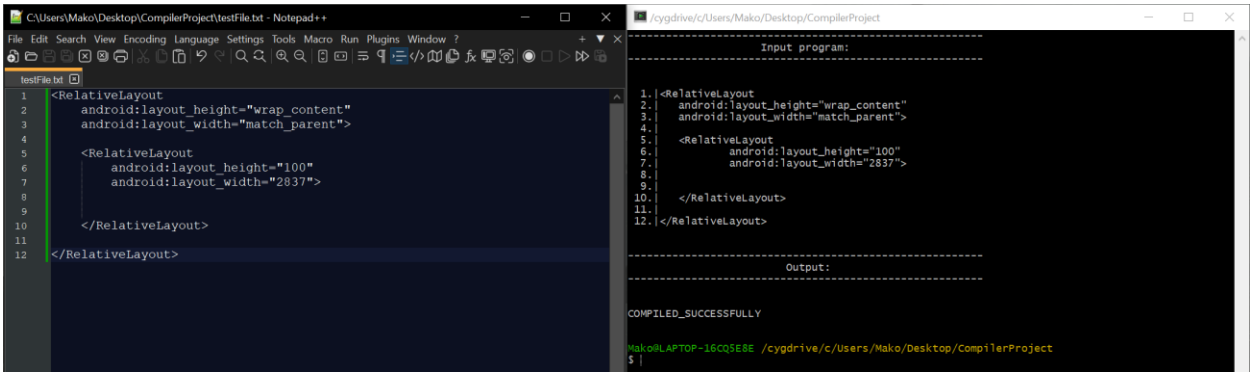
Test_15 (Elements με διαφορετικά ID - Successful)



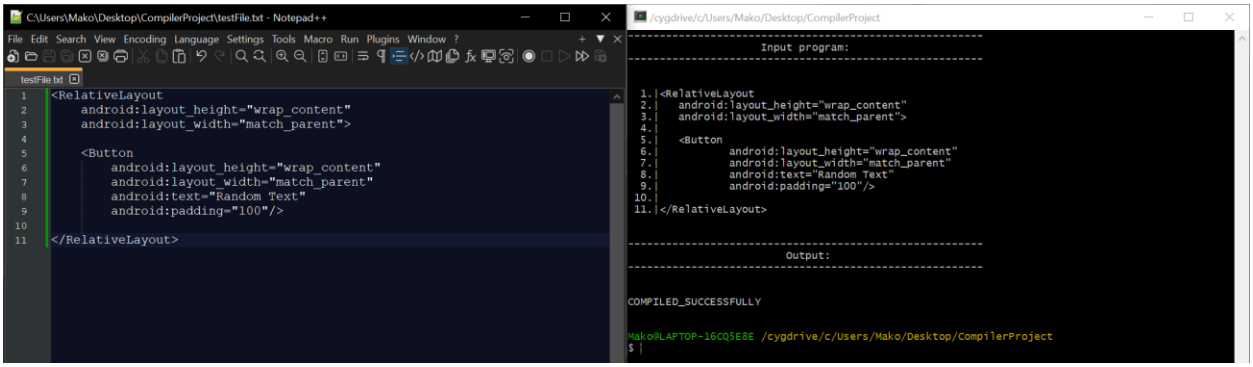
Test_16 (Elements με ίδια ID - Error)



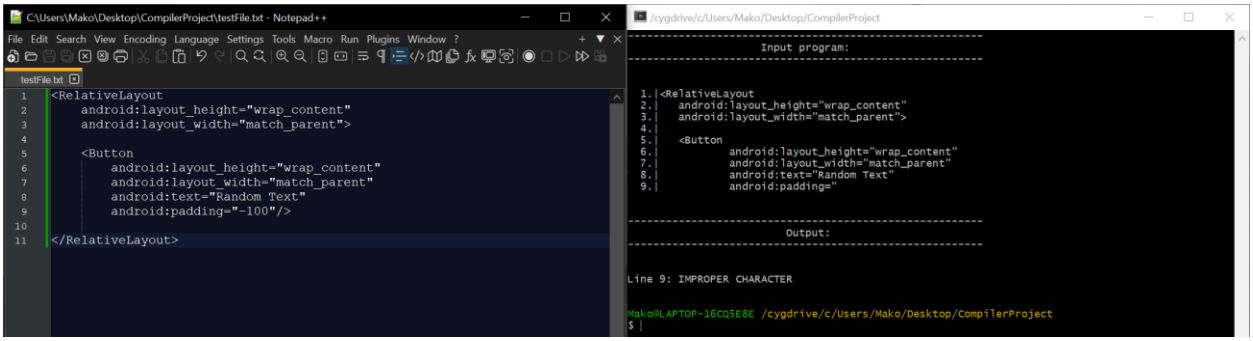
Test_17 (Width & Height με όλα τα διαθέσιμα Values - Successful)



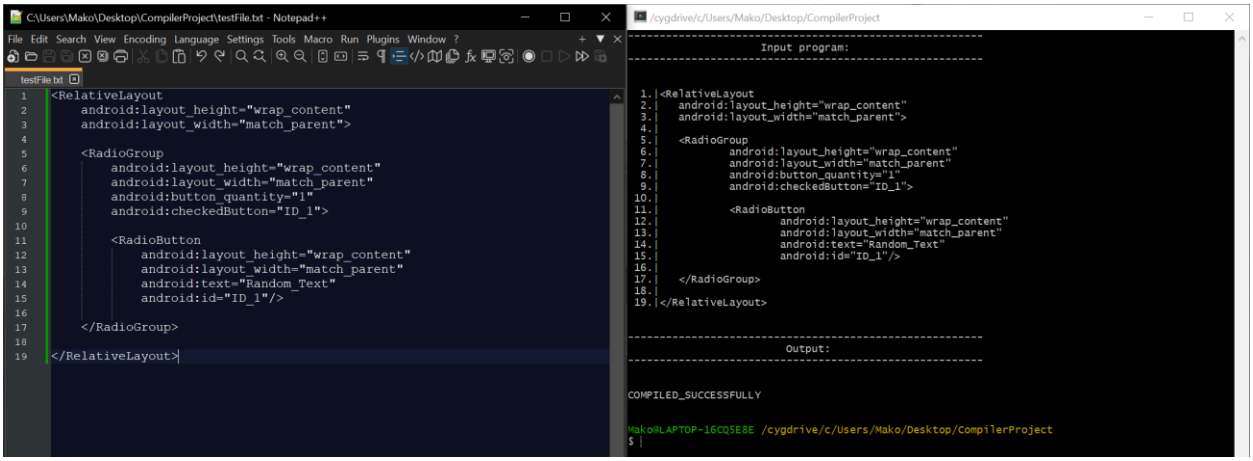
Test_18 (Padding με θετικό ακέραιο - Successful)



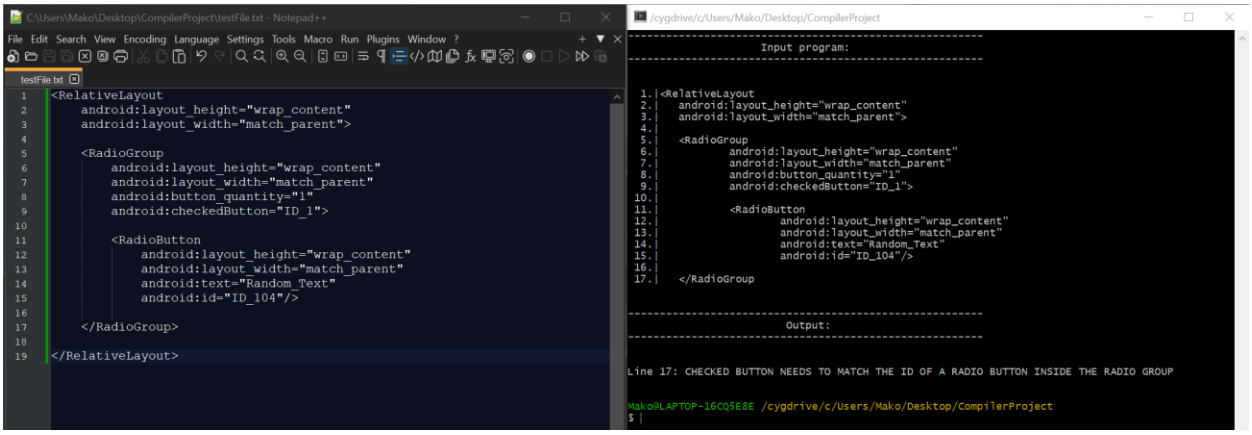
Test_19 (Padding με αρνητικό ακέραιο - Error)



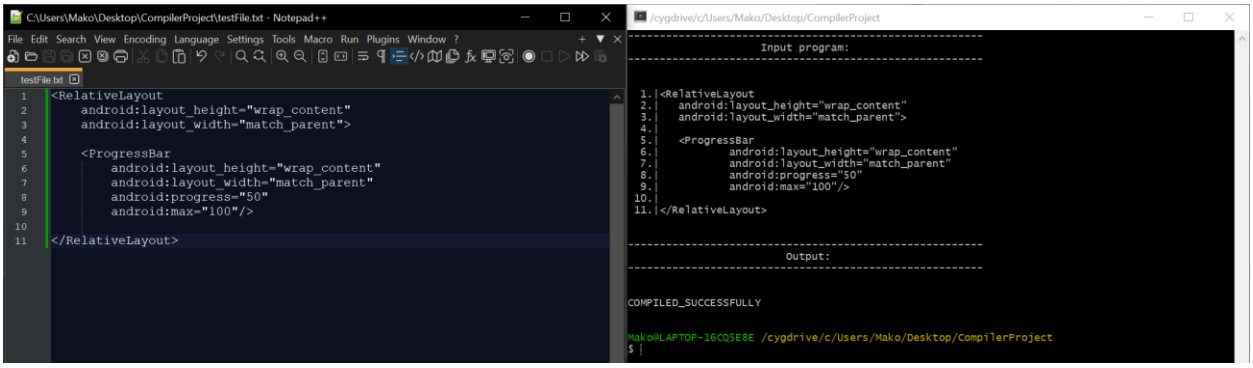
Test_20 (CheckedButton αντιστοιχεί σε ID εντός του RadioGroup - Successful)



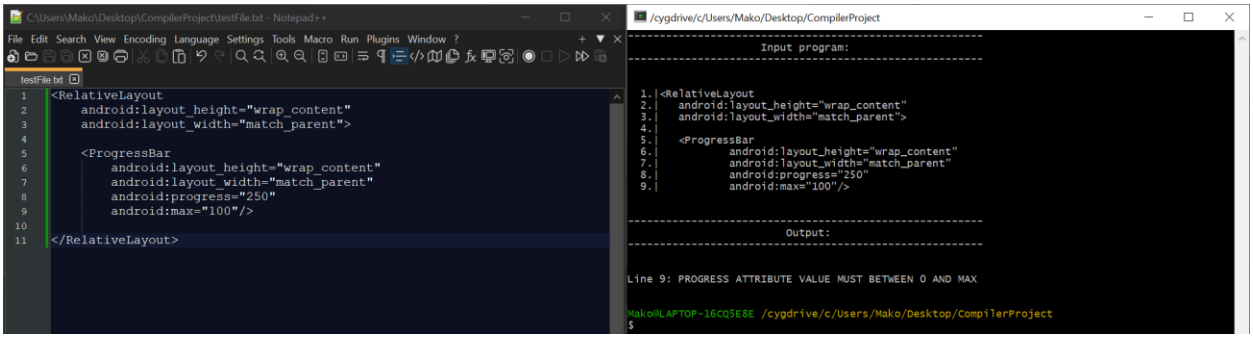
Test_21 (CheckedButton δεν αντιστοιχεί σε ID εντός του RadioGroup - Error)



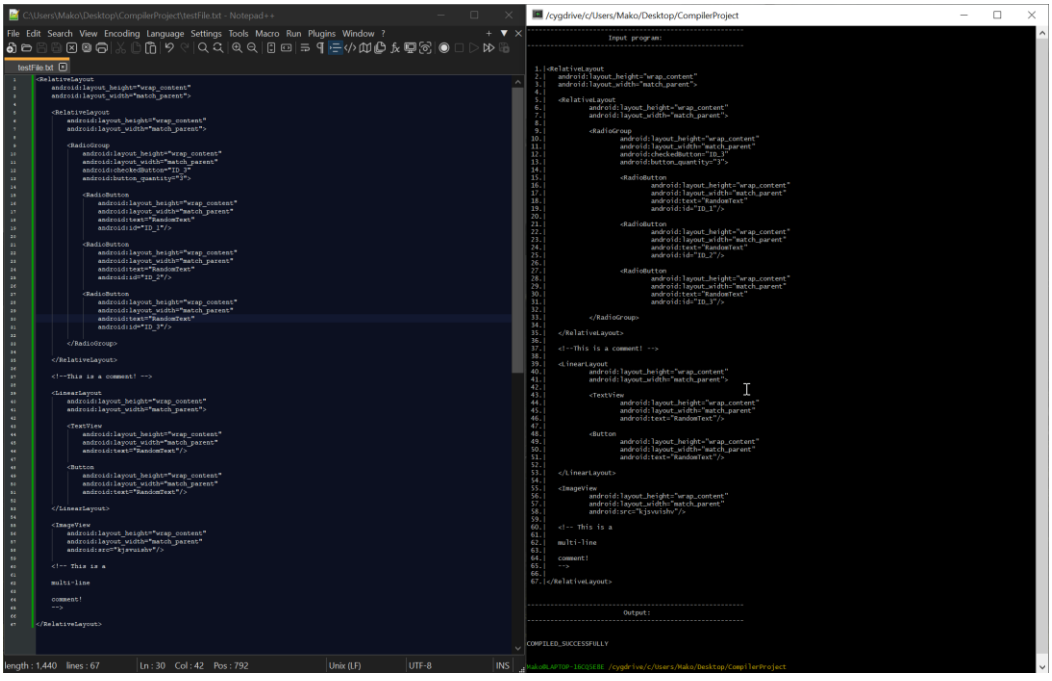
Test_22 (Τιμή Progress μεταξύ 0 και Max - Successful)



Test_23 (Τιμή Progress εκτός εύρους 0 και Max - Error)



Test_24 (Ένα πιό πλήρες παράδειγμα - Successful)



3. Ανάλυση και σχολιασμός πηγαίου κώδικα (scanner.l) & (parser.y)

Καθ' όλη τη διαδικασία ανάπτυξης και υλοποίησης των ζητημάτων αυτού του Project, δόθηκε μεγάλη έμφαση στην έξυπνη διαχείριση των ερωτημάτων μέσω κώδικα C και αποφυγή ορισμού μεγάλης γραμματικής με πολλούς μεμονωμένους κανόνες. Έτσι, θα γίνει γρήγορα αντιληπτό, ότι τόσο η γραμματικοί κανόνες για το Bison, αλλά και οι κανονικές εκφράσεις για την λεκτική ανάλυση του Flex, είναι αρκετά αφηρημένα και ορισμένα γενικά. Αντιθέτως, έχουν υλοποιηθεί πολλές και αρκετά εκτενείς συναρτήσεις στο επίπεδο της C, οι οποίες εκτελούν όλους τους απαραίτητους ελέγχουν και διεργασίες για τον λεκτικό και συντακτικό έλεγχο την ζητούμενης γλώσσας.

*Σημείωση: Έχοντας περάσει πλέον από κάθε απαραίτητο στάδιο για την υλοποίηση αυτού του Project έχει γίνει αντιληπτό, πως ορισμένα από τα ερωτήματα ενδεχομένως να υλοποιούνταν πιο αποδοτικά και με λιγότερο κώδικα απ' ότι τελικά χρειάστηκε.

3.1 Παραδοχές και σχεδιαστικές επιλογές

- Όποτε προκύπτει κάποιο σφάλμα κατά τον έλεγχο του αρχείου εισόδου πρέπει η εκτέλεση να σταματά. Για τη διαχείριση των όποιων σφαλμάτων προτιμήσαμε την δημιουργία δικής μας συνάρτησης, παρά της **yyError()** του Bison. Γι' αυτό το λόγο, προκειμένου να εξομοιωθεί κατά μια έννοια ο τερματισμός της εκτέλεσης, χρησιμοποιείται η σημαία **copyInputFlag**, η οποία ενεργοποιείται όποτε προκύψει κάποιο σφάλμα. Επειδή την ελέγχουν όλες οι διεργασίες που κάνουν κάποιου είδους εκτύπωση, στην πράξη μοιάζει σαν η εκτέλεση να σταματά. Παρ' ολ' αυτά το αναφέρουμε εδώ ώστε να μην προκύψει κάποια παρεξήγηση στην πορεία.
- Σε πολλές διεργασίες που πρέπει να γίνουν, είναι αναγκαία η χρήση της μεταβλητής **yytext**, του Flex, η οποία περιέχει το κείμενο που έχει αναγνωριστεί πιο πρόσφατα από το scanner. Γι' αυτός το λόγο, προκειμένου να υπάρχει πρόσβαση στις αναγνωρισμένες ακολουθίες έπρεπε να σπάσουμε τους γραμματικούς κανόνες του Bison σε μέρη, στα σημεία που έπρεπε να διαβαστεί κάτι από το **yytext**.
- Η κύρια ιδέα πίσω από τον έλεγχο σωστής εμφώλευσης των διαφόρων elements, είναι η προσθήκη των ονομάτων των start tags, σε ένα stack και κάθε φορά που συναντιέται ένα closing tag, να ελέγχεται αν είναι ίδιο με την κορυφή του **stack**. Έτσι, ανεξαρτήτως του πως έχουν δομηθεί τα elements εντός του αρχείου εισόδου, όσο τηρείται σωστή εμφώλευση, η κορυφή του stack πάντα θα αντιστοιχεί στο element εντός του οποίου βρίσκεται ο έλεγχος και είναι το πρώτο που οφείλει να κλείσει. Προφανώς τα self-closing tags δεν λαμβάνονται υπόψη στην εμφώλευση.

- Αφού το attribute id δεν είναι υποχρεωτικό, δημιουργείται πρόβλημα αν χρησιμοποιηθούν **RadioButton** δίχως αυτό. Γι' αυτό κάνουμε την παραδοχή πως τα **RadioButton** θα παίρνουν πάντα id. Το πρόβλημα αυτό θα μπορούσε να λυθεί με την χρήση default id με αυξανόμενη αρίθμηση για όσα **RadioButton** δεν είχαν, αλλά δεν υπήρξε αρκετός χρόνος γι' αυτό.
- Αφού δεν υπάρχει ποτέ η ανάγκη για χρήση αρνητικών ακέραιων τιμών, η κανονική έκφραση για το **INTEGER**, αναγνωρίζει γενικά ακολουθίες ψηφίων δίχως πρόσημο. Αν το scanner συναντήσει ποτέ αρνητικό πρόσημο, προκαλείται σφάλμα, αλλά όχι επειδή υπάρχει κάποιος σχετικός έλεγχος, αλλά επειδή το συγκεκριμένο σύμβολο δεν ανήκει σε αυτά που αναγνωρίζει η κανονική έκφραση.
- **Σημαντικό!** Τα δοκιμαστικά αρχεία txt έχουν ρυθμιστεί επίτηδες με Format (LF) του Unix αντί για (CR LF) των Windows, επειδή προέκυπταν σφάλματα από την διαφορετική διαχείριση που κάνουν με τους ειδικούς χαρακτήρες τέλους γραμμής.

3.2 Λεκτική ανάλυση (Flex) – Regular Expressions & Κώδικας C

3.2.1 Scanner – Δηλώσεις Flex

Για το κομμάτι των δηλώσεων, αρχικά

- Προσθέτουμε το **%option noyywrap**, το οποίο όταν βρεθεί EOF στο τελικό αρχείο εισόδου, θεωρεί πως δεν υπάρχουν περαιτέρω αρχεία εισόδου και αποτρέπει την κλήση της συνάρτησης **yywrap()** του Flex.
- Ορίζουμε την κατάσταση **%s COMMENT**, την οποία αξιοποιούμε για την διαχείριση των σχολίων.
- Κάνουμε **#include** τις βιβλιοθήκες **<stdio.h>** και **<string.h>** που χρησιμοποιούνται αργότερα στις συναρτήσεις της C, κι επίσης το αρχείο **parser.tab.h** που παράγεται κατά το compilation του πηγαίου κώδικα του parser από το Bison.
- Εισάγουμε την εξωτερική συνάρτηση **myErrorPrint()**, που έχει ορισθεί εντός του parser και χρειάζεται να χρησιμοποιηθεί κι εντός του scanner.
- Ορίζουμε μεταβλητές και συναρτήσεις οι οποίες χρησιμοποιούνται παρακάτω και θα αναλυθούν περισσότερο όποτε προκύπτουν εντός του κώδικα.

3.2.2 Scanner – Κανονικές Εκφράσεις

Καταρχάς λόγω της χρήσης flex states που χρειάστηκε να κάνουμε για τη διαχείριση των σχολίων, δημιουργήθηκε πρόβλημα στις δηλώσεις των κανόνων συντακτικής ανάλυσης. Εν ολίγης, αφότου ορίσθηκε το block κανόνων που αναγνωρίζονται εντός του **COMMENT** state, όλα τα υπόλοιπα χρειάστηκε επίσης να εισαχθούν σε ένα block υπό το **INITIAL** state, αλλιώς προέκυπταν errors εντός το flex κατά το compilation.

Έτσι λοιπόν, κατά το **INITIAL** state το scanner, αναγνωρίζει:

- Το σύνολο ειδικών συμβόλων **(<,>,/,:,!,=,”)**, για τα οποία επιστρέφει στον parser tokens που ως ονομασία έχω ορίσει το ίδιο το σύμβολο εντός **(‘ ‘)**.
- Τα ονόματα όλων των **Tags**, για τα οποία ο parser δέχεται το ίδιο όνομα token **ELEMENT_NAME**.
- Τα ονόματα όλων των **Attributes**, για τα οποία ο parser δέχεται το ίδιο όνομα token **ATTRIBUTE_NAME**.
- Την δεσμευμένη λέξη **android** που χρησιμοποιείται παράλληλα με τα Attributes και επιστρέφει στον parser το όνομα token **ATTRIBUTE_START**.
- Την κανονική έκφραση **[0-9]+** για την αναγνώριση ακέραιων αριθμών στο δεκαδικό σύστημα με τουλάχιστον ένα ψηφίο, και επιστρέφεται το token **INTEGER**.
- Την κανονική έκφραση **[0-9A-Za-z_]+** για την αναγνώριση αλφαριθμητικών με τουλάχιστον ένα χαρακτήρα και μπορούν να περιέχουν επίσης κάτω παύλα και κενό. Επιστρέφεται το token **STRING.**

- Τον ειδικό χαρακτήρα νέας γραμμής **\n** για τον οποίο αυξάνει κατά 1 την μεταβλητή **currentLine**, η οποία ορίζεται εντός του scanner και κρατά το πλήθος των γραμμών εντός του αρχείου εισόδου και έπειτα καλεί τη συνάρτηση **printLineNumbers()**, η οποία χρησιμοποιείται για την εκτύπωση της αρίθμησης των γραμμών στην έξοδο.
- Το κενό και το Tab.
- Και τέλος, κάθε άλλο δυνατό χαρακτήρα, με τη χρήση της κανονική έκφρασης που περιλαμβάνει μόνο τελεία.

Επίσης, πάλι κατά το **INITIAL** state, αλλά σε διαφορετικό σημείο αναγνωρίζεται η ακολουθία χαρακτήρων **"<!--"**, που σηματοδοτεί την αρχή ενός σχολίου. Σε αυτή την περίπτωση καλείται η συνάρτηση του Flex **BEGIN(COMMENT)**, ώστε να μεταβούμε στην κατάσταση **COMMENT**.

*Σημείωση: Σε κάθε κανονική έκφραση που αναφέρθηκε ως τώρα, καλείται επίσης η συνάρτηση **printProgram()**, η οποία είναι υπεύθυνη για την εκτύπωση του αρχείου εισόδου στην έξοδο.

Έπειτα, κατά το **COMMENT** state το scanner, αναγνωρίζει:

- Κάθε πιθανό χαρακτήρα ξανά με τη χρήση της τελείας ως κανονική έκφραση. Σε αυτή τη περίπτωση καλείται η συνάρτηση **checkCharacters()**, η οποία φροντίζει τον έλεγχο σχετικά με τον περιορισμό ύπαρξης 2 συνεχόμενων εμφανίσεων παύλας εντός του σχολίου. Παρόλο που δεν είναι άμεσα φανερό σε αυτό το σημείο, το περιεχόμενο του σχολίου εκτυπώνεται κι αυτό εντός την συνάρτησης αργότερα.
- Τον ειδικό χαρακτήρα νέας γραμμής, για τον οποίο η διαχείριση είναι ακριβώς ίδια με τον αντίστοιχο κανόνα στο **INITIAL** state.
- Την ακολουθία χαρακτήρων **"-->"**, που σηματοδοτεί το τέλος του σχολίου. Σε αυτή την περίπτωση καλείται η συνάρτηση του Flex **BEGIN(INITIAL)**, ώστε να μεταβούμε στην κατάσταση **INITIAL**. Επίσης εκτυπώνεται στην έξοδο και μηδενίζει τη μεταβλητή **hyphensInARow**, η οποία χρησιμοποιείται από την συνάρτηση **checkCharacters()**.

3.2.3 Scanner – Συναρτήσεις C

void checkCharacters()

Δουλειά αυτής της συνάρτησης είναι ο έλεγχος του περιεχομένου ενός σχολίου ώστε να εντοπισθεί τυχόν ύπαρξη δύο **'-'** στη σειρά.

Πρώτα ελέγχει τη μεταβλητή **yytext** του Flex, η οποία περιέχει το πιο πρόσφατο κείμενο που έχει αναγνωριστεί από κάποιον λεκτικό κανόνα, στην προκειμένη κάποιον μεμονωμένο χαρακτήρα εντός του σχολίου. Αν ο χαρακτήρας είναι παύλα, αυξάνει κατά 1 την μεταβλητή **hyphensInARow**, η οποία αποτελεί μετρητή για τις παύλες, και εκτυπώνει το κείμενο στην έξοδο. Αν είναι οποιοσδήποτε άλλος χαρακτήρας, επαναφέρει τον μετρητή στο 0, διότι ο τελευταίος χαρακτήρας που αναγνωρίσθηκε δεν ήταν παύλα και εκτυπώνει το κέιμενο.

Τέλος, γίνεται ένας έλεγχος στο πλήθος των **'-'** μέσω της **hyphensInARow**. Αν έχουν μετρηθεί τουλάχιστον 2 παύλες στη σειρά καλείται η συνάρτηση **myErrorPrint()** με κατάλληλη είσοδο για να διαχειριστεί την εκτύπωση του μηνύματος σφάλματος.

void printProgram()

Η κύρια δουλειά αυτής της συνάρτησης είναι απλά η εκτύπωση όσων διαβάζει από το αρχείο εισόδου το scanner και πρέπει να εμφανιστούν στην έξοδο. Αυτό το κάνει με τη χρήση της μεταβλητής **yytext** του Flex, που περιέχει το κείμενο που αναγνωρίστηκε τελευταίο από κάποιον κανόνα. Πάντα, πριν γίνει η εκτύπωση, ελέγχεται το flag **copyInputFlag**, για τον λόγο που εξηγείται στις παραδοχές.

Επίσης η συνάρτηση αυτή εκτελεί κι έναν δευτερεύοντα στόχο. Φροντίζει να εκτυπώνεται στην αρχή της εξόδου η αρίθμηση για την πρώτη γραμμή του αρχείου. Ο λόγος είναι πως υπήρχαν δυσκολίες στην υλοποίηση αυτής της λειτουργίας μέσω της **printLineNumbers()**, η οποία κανονικά εκτυπώνει την αρίθμηση, οπότε η μοναδική λύση που βρέθηκε ήταν η εκτύπωση της πρώτης γραμμής από εδώ.

void printLineNumbers()

Σκοπός αυτής της συνάρτησης είναι μόνο η εκτύπωση της αρίθμησης των γραμμών στην έξοδο. Αφού γίνει έλεγχος της σημαίας σφάλματος, εκτυπώνει την αρίθμηση που χρειάζεται με την κατάλληλη μορφοποίηση. Η μορφοποίηση αλλάζει ελάχιστα με βάση τον αριθμό των ψηφίων της εκάστοτε αρίθμησης προκειμένου να εμφανίζονται όλες οι γραμμές ομοιόμορφα. Τον αριθμό στον οποίο βρίσκεται κάθε στιγμή το πρόγραμμα, τον διαβάζουμε από την μεταβλητή **currentLine**.

3.3 Συντακτική ανάλυση (Bison)- BNF Grammar & Κώδικας C

3.3.1 Parser – Δηλώσεις Bison

Για το κομμάτι των δηλώσεων, αρχικά:

- Κάνουμε **#include** τις βιβλιοθήκες **<stdio.h>**, **<stdlib.h>** και **<string.h>** που χρησιμοποιούνται αργότερα στις συναρτήσεις της C.
- Εισάγουμε την εξωτερική συνάρτηση **yylex()** και τις μεταβλητές **yytext** και **yyleng** του Flex, καθώς και τις μεταβλητές **copyInputFlag** και **currentLine**, που ορίσαμε εμείς για το scanner, αλλά χρειάζεται να χρησιμοποιήσουμε και στον parser.
- Ορίζουμε μεταβλητές και συναρτήσεις οι οποίες χρησιμοποιούνται παρακάτω και θα αναλυθούν περισσότερο όποτε προκύπτουν εντός του κώδικα.
- Αναγνωρίζουμε τα token που στέλνει το scanner στον parser.

3.3.2 Parser – Κανόνες Γραμματικής

Όπως έγινε ήδη αναφορά στην εισαγωγή του μέρους 3, η γραμματικοί κανόνες που έχουν οριστεί είναι αρκετά γενικοί εφόσον η κύρια διαχείριση γίνεται από την C. Πιο συγκεκριμένα έχουμε:

- **program**:
  Πρόκειται για τη ρίζα του δέντρου συντακτικής ανάλυσης και είναι η αρχή του προγράμματος που εισάγεται σαν είσοδος.
  Γίνεται να είναι η κενή λέξη, στην οποία περίπτωση το αρχείο εισόδου είναι εντελώς κενό και εκτυπώνεται το αντίστοιχο μήνυμα εξόδου. Αλλιώς, μπορεί να είναι το token **element_start**, το οποίο αποτελεί το πρώτο μέρος ενός tag και αν δεν προκύψει πουθενά σφάλμα στο αρχείο εισόδου και ο κανόνας αυτός είναι που εκτυπώνει το τελικό μήνυμα επιτυχίας.

- **element_start**:
  Όπως αναφέρθηκε παραπάνω, αυτός ο κανόνας αντιπροσωπεύει την αρχής ενός element. Αφού αναγνωριστεί το όνομα του tag, καλείται η συνάρτηση **pushStackI()** για να το προσθέσει στο stack ελέγχου της εμφώλευσης, όπως εξηγείται και στο μέρος των σχεδιαστικών επιλογών. Έπειτα ακολουθεί το token **element_middle**.

- **element_middle**:
  Δουλειά αυτού του κανόνα είναι να διαχειριστεί το μεσαίο τμήμα κάθε element, που είναι τα attribues. Έτσι, εντός αυτού του κανόνα μπορεί να αναγνωριστεί το token **attributes**, που εξηγείται παρακάτω. Αφού ο έλεγχος επιστρέψει στο σημείο αυτό καλείται η συνάρτηση **necessaryAttributes()**, που εκτελεί έλεγχο για την ύπαρξη όλων των αναγκαίων attributes για το συγκεκριμένο element. Αμέσως μετά η **free(attributeList)**, που αποδεσμεύει τη μνήμη της μεταβλητής **attributeList** και ταυτόχρονα μηδενίζεται επίσης η μεταβλητή **attributeCount**. Και οι δύο αυτές μεταβλητές χρησιμοποιούνται κατά τον έλεγχο των attributes. Τέλος,

αναγνωρίζεται το token **element_end**.

- **element_end**:
  Αυτός ο κανόνας σπάει σε τρία κύρια μέρη.
  Πρώτα έχουμε την περίπτωση το element να έχει content. Σε αυτή την περίπτωση αφού αναγνωριστεί το token content και διαβαστεί το **ELEMENT_NAME** στο closing tag, καλείται η συνάρτηση **compTags()**, που είναι υπεύθυνη για τον έλεγχο της
  Στη δεύτερη περίπτωση, το element δεν έχει περιεχόμενο, οπότε ακολουθεί απλά ο ίδιος έλεγχος της εμφώλευσης.
  Και τέλος, η τελευταία περίπτωση είναι το element να είναι self-closing, οπότε ούτως η άλλως δεν μπορεί να περιέχει content. Σε αυτή την περίπτωση απλά καλείται η **popStack()**, για να αφαιρέσει το όνομα του συγκεκριμένου tag από το stack, αφού τα self-closing elements, δεν επηρεάζουν την εμφώλευση.

- **content**:
  Το content ενός element όπως είναι γνωστό από τις προδιαγραφές της γλώσσας, μπορεί να αποτελείται μόνο από άλλα elements. Έτσι, ο συγκεκριμένος κανόνας οδηγεί σε μια ακολουθία από τουλάχιστον ένα element σαν content του γονιού element στο οποία βρισκόμαστε.

- **attributes**:
  Αντίστοιχα με τον προηγούμενο κανόνα, αυτός ο κανόνας οδηγεί σε μια ακολουθία από τουλάχιστον ένα attribute.

- **attribute**:
  Αυτός ο κανόνας αντιπροσωπεύει ένα μεμονωμένο attribute εντός του tag κάποιου element. Έτσι, αρχικά αναγνωρίζει το token **ATTRIBUTE_START**, που είναι η ακολουθία **“android”** και τον ειδικό χαρακτήρα **“:”**. Έπειτα, διαβάζει το όνομα του attribute και καλεί τις συναρτήσεις **checkAttribute()** και **storeAttribute()**. Η πρώτη ελέγχει αν το συγκεκριμένο attribute μπορεί να χρησιμοποιηθεί εντός του τωρινού element και η δεύτερη αποθηκεύει το όνομα του attribute σε μια λίστα που περιέχει όλα τα attributes που χρησιμοποιήθηκαν ως τώρα για το τωρινό element. Τέλος, αναγνωρίζεται το token **attribute_value**.

- **attribute_value**:
  Αυτός ο κανόνας είναι απλά η τιμή του έχει ανατεθεί σε κάποιο συγκεκριμένο attribute και μπορεί να είναι απλά είτε θετικός ακέραιος αριθμός, είτε αλφαριθμητικό. Και στις δύο περιπτώσεις καλείται η συνάρτηση **attributeValueCheck()**, αλλά με διαφορετική τιμή εισόδου. Για τα αλφαριθμητικά καλείται επίσης η **idCheck()**.

3.3.3 Parser – Συναρτήσεις C

void pushStack()

Αυτή η συνάρτηση χρησιμοποιείται για να προστεθούν ονόματα από elements στην μεταβλητή stack, η οποία χρησιμοποιείται για τον έλεγχο της εμφώλευσης των στοιχείων.

Κατά την κλήση της, ορίζεται αρχικά μια μεταβλητή temp, δεσμεύεται γι' αυτή μνήμη και αντιγράφεται σε αυτή το περιεχόμενο της **yytext**. Αμέσως μετά γίνεται έλεγχος του περιεχομένου αυτού και αν είναι το **"LinearLayout"**, θέτουμε τη σημαία **linearContentFlag** στο 1, αλλιώς τη θέτουμε στο 0.

Η χρήση της σημαίας αυτής εξηγείται αναλυτικά στην συνάρτηση **popStack()**.

Έπειτα, διαχωρίζουμε μεταξύ των περιπτώσεων το **stackElements** να είναι άδειο ή όχι. Ο λόγος είναι, ότι αν είναι άδειο, οπότε καταχωρούμε το πρώτο στοιχείο, πρέπει να ελέγξουμε αν είναι κάποιο από τα δύο επιτρεπτά tags. Αν όχι, καλείται η **myErrorprint()**. Επίσης, υπάρχει και ένας δεύτερος λόγος, που είναι η αρχική δέσμευση μνήμης για την μεταβλητή stack.

Να αναφερθεί εδώ, ότι η μεταβλητή **stackElements** κρατάει το πλήθος των στοιχείων εντός του stack και χρησιμοποιείται για τον έλεγχο αυτού.

Αν από την άλλη, το stack έχει ήδη στοιχεία μέσα, ακολουθεί η εξής διαδικασία. Εκτελούμε δύο ελέγχους που σχετίζονται με το tag **RadioGroup**. Ο πρώτος ελέγχει, ότι ένα στοιχείο **RadioButton** θα εμφωλεύεται μόνο εντός **RadioGroup** και ο δεύτερος, ότι κανένα στοιχείο πέρα από **RadioButton** δεν θα εμφωλεύεται εντός **RadioGroup**.

Αφού περάσουν αυτοί οι έλεγχοι, προχωράμε στην καταχώρηση εντός του stack. Επειδή δουλεύουμε με C, στην οποία δεν έχουμε πρόσβαση σε dynamic arrays, πρέπει να κάνουμε βήμα-βήμα την αλλαγή του μεγέθους του **stack**. Αφού πλέον δημιουργηθεί ένα νέο stack με μια έξτρα θέση, εισάγουμε το νέο στοιχείο, αποδεσμεύουμε την μνήμη του παλιού stack και αυξάνουμε την **stackElements** κατά 1.

void popStack()

Αυτή η συνάρτηση αποτελεί το δεύτερο μέρος του ελέγχου εμφώλευσης στοιχείων και δουλειά της είναι το αντίθετο της **pushStack()**, δηλαδή να αφαιρεί στοιχεία από την μεταβλητή stack.

Εδώ πρέπει να εξηγήσουμε την χρήση της σημαίας **linearContentFlag**, που πρωτοεμφανίστηκε στην εξήγηση της **pushStack()**. Όπως υποδεικνύεται στις προδιαγραφές της γλώσσας το element **LinearLayout** δεν γίνεται να είναι κενό. Έτσι εντός της **pushStack()**, όταν συναντάται ένα **LinearLayout**, η σημαία παίρνει την τιμή 1. Εδώ εκτελούμε πλέον τον έλεγχο για το αν το element είναι η όχι κενό.

Αν η σημαία έχει την τιμή 1 όταν καλείται αυτή εδώ η συνάρτηση σημαίνει πως από τη στιγμή που συναντήθηκε το **LinearLayout**, μέχρι να φράσουμε στο τέλος του, δεν συναντήθηκε κανένα άλλο element, αλλιώς η σημαία θα είχε ξανά αλλάξει σε 0. Έτσι αν η σημαία είναι 1 και η κορυφή του stack είναι **LinearLayout**, προκαλείται σφάλμα.

Έπειτα αν η κορυφή του **stack** είναι **RadioGroup**, χρειάζεται να ελεγθούν κάποια ζητούμενα που μπορούμε να τα ελέγξουμε μόνο στο τελείωμα του element. Πρώτα καλείται η **RadioGroupCheck()**, που εκτελεί διάφορους ελέγχους πάνω στο element σε σχέση με τα attributes του. Έπειτα αποδεσμεύουμε μνήμη και επαναφέρουμε σε αρχικές τιμές, μεταβλητές που χρησιμοποιούνται σε αυτούς ακριβώς τους ελέγχους.

Παρομοίως. Αν η κορυφή του stack είναι **ProgressBar**, χρειάζεται να ελεγχθούν αντίστοιχα ζητούμενα. Καλείται η **progressBarCheck()**, που εκτελεί αυτούς τους ελέγχους και στη συνέχεια επαναφέρονται σε αρχικές τιμές ορισμένες μεταβλητές.

Τέλος, με αντίστροφη διαδικασία από αυτή που γίνεται στην **pushStack()**, η μεταβλητή stack αντικαθίσταται από μια με μια θέση λιγότερη, αποδεσμεύεται μνήμη από σχετικές μεταβλητές και μειώνεται κατά 1 η **stackElements**.

void compTags()

Αυτή η συνάρτηση αποτελεί το τελευταίο κομμάτι του ελέγχου της εμφώλευσης στοιχείων και είναι αρκετά σύντομη. Απλά εκτελεί μια σύγκριση μεταξύ της κορυφής του stack και του closing-tag που έχει πιο πρόσφατα αναγνωρισθεί και περιέχεται στην **yytext**.

Αν τα δύο ταυτίζονται τότε έχει γίνει σωστή εμφώλευση και καλείται η **popStack()**, αλλιώς προκύπτει σφάλμα.

void checkAttribute()

Αυτή η συνάρτηση ελέγχει αν το attribute που έχει μόλις διαβαστεί, μπορεί όντως να χρησιμοποιηθεί από το στοιχείο στο οποίο έχει τοποθετηθεί.

Αρχικά, καλείται η **assignTagNumber()**, που επιστρέφει πίσω έναν ακέραιο που αντιστοιχεί στο tag που βρίσκεται στην κορυφή του stack, δηλαδή στο tag εντός του οποίου βρισκόμαστε. Αυτό το κάναμε μόνο επειδή δεν γίνεται να χρησιμοποιήσουμε αλφαριθμητικά εντός ελέγχου switch, παρά μόνο ακέραιους.

Έπειτα, εντός ενός ελέγχου switch, ελέγχουμε αν το attribute που μόλις έχει διαβαστεί είναι ένα από τα επιτρεπόμενα για το στοιχείο στο οποίο βρίσκεται. Από αυτόν τον έλεγχο εξαιρούνται τα **layout_width**, **layout_height** και **id**, αφού μπορούν να χρησιμοποιηθούν σε κάθε στοιχείο. Αν το attribute δεν ανήκει στο element, τότε προκαλείται σφάλμα και καλείται η **myErrorPrint()**.

void storeAttribute()

Αυτή η συνάρτηση ακολουθεί την **checkAttribute()** και αναλαμβάνει να αποθηκεύσει το attribute που μόλις έχει ελεγχθεί στην μεταβλητή **attributeList**, που περιέχει όλα τα attributes εντός του τωρινού στοιχείου.

Η εσωτερική της λειτουργία είναι ολόιδια με την διαδικασία αποθήκευσης tag στην μεταβλητή stack, οπότε δεν θα αναλυθεί ξανά.

Στο τέλος αυξάνεται κατά 1 η **attributeCount**, που αντιπροσωπεύει το πλήθος attribute εντός της **attributeList**.

void necessaryAttributes()

Η συγκεκριμένη συνάρτηση φαίνεται αρκετά μεγάλη, όμως δεν είναι ιδιαίτερα περίπλοκη. Στην πραγματικότητα αποτελείται από πολλούς μικρούς μεμονωμένους ελέγχους που έχουν τοποθετηθεί εντός της ίδιας συνάρτησης και αφορούν την ύπαρξη όλων των απαραίτητων attributes για κάθε στοιχείο.

Καταρχάς ορίζονται τρεις σημαίες **flag1**, **flag2** και **flag3**, με τις οποίες χωρίζω τα στοιχεία σε ομάδες βάσει των απαραίτητων τους attributes και καλείται η **assignTagNumber()** για να χρησιμοποιούμε ακέραιους στους ελέγχους αντί για αλφαριθμητικά.

Κάθε element έχει τουλάχιστον 2 υποχρεωτικά attributes, που είναι τα **layout_width** και **layout_height**. Γι' αυτό, ο πρώτος έλεγχος είναι απλά αν το πλήθος των attributes είναι μικρότερο από 2, στην οποία περίπτωση απορρίπτουμε κατευθείαν.

Έπειτα ακολουθούν δύο αρκετά όμοιοι μεταξύ τους έλεγχοι. Και στις δύο περιπτώσεις διατρέχεται η λίστα των attributes που έχουν χρησιμοποιηθεί στο τωρινό στοιχείο και ψάχνουμε μια για το **layout_width** και μια για το **layout_height**. Αν βρεθούν, τότε οι αντίστοιχες σημαίες **flag1** και **flag2** παίρνουν τη τιμή 1.

Μετά ακολουθούν τρεις μεμονωμένοι έλεγχοι που αφορούν τρεις ξεχωριστές ομάδες tags.

- Αρχικά έχουμε μόνο του το **ImageView**, το οποίο χρειάζεται υποχρεωτικά το attribute src.
- Δεύτερη ομάδα είναι το **RadioGroup** μόνο του, που χρειάζεται το button_quantity.
- Και τέλος, είναι μαζί τα **TextView**, Button και **RadioButton**, που πρέπει να έχουν το text.

Ανάλογα με το tag με το οποίο ασχολούμαστε εισερχόμαστε στον αντίστοιχο έλεγχο και διατρέχοντας ξανά τη λίστα των attributes, ψάχνουμε τα ζητούμενα. Αν οι προδιαγραφές πληρούνται, τότε την τιμή παίρνει επίσης το **flag3**.

Οι τρεις σημαίες ελέγχονται ταυτόχρονα αν έχουν πάρει τιμή 1 και αν ναι, τότε έχουν εντοπισθεί επιτυχώς όλα να υποχρεωτικά attributes για το tag, αλλιώς προκύπτει σφάλμα.

int assignTagNumber()

Μοναδικός σκοπός αυτής της συνάρτησης είναι να ελέγχει ποιο tag βρίσκεται στην κορυφή του stack και να επιστρέφει πίσω, τον ακέραιο που αντιστοιχεί σε αυτό για να χρησιμοποιηθεί από τις **checkAttribute()** και **necessaryAttributes()**.

void attributeValueCheck(int type)

Μέρος του ελέγχου γύρω από τα attributes είναι και η τιμή που ανατίθεται στο καθένα. Αυτή είναι και η δουλειά αυτής της συνάρτησης, η οποία χωρίζεται σε δύο μεγάλα μέρη ανάλογα τον τύπο της τιμής. Η αναγνώριση αυτού γίνεται με τις τιμές 0 και 1, όπου 0 αντιστοιχεί σε ακέραιο και 1 σε αλφαριθμητικό.

Για τους ακέραιους:

Πρώτα γίνεται ένας έλεγχος αν έχει ανατεθεί σε οποιοδήποτε από τα διαθέσιμα attributes που δέχονται ακέραια τιμή και αν όχι, τότε προκύπτει σφάλμα.

Έπειτα, αν το τωρινό attribute του οποίου την τιμή ελέγχουμε είναι το **progress**, **max** ή **button_quantity**, αποθηκεύουμε την τιμή που τους έχει ανατεθεί αντιστοίχως στις **currentProgress**, **currentMax** και **currentButtonQuantity** για μετέπειτα χρήση από τους αντίστοιχους ελέγχους.

Για τα αλφαριθμητικά:

Πρώτα ελέγχουμε αν το attribute που ελέγχουμε είναι το **layout_width** ή **layout_height**, στην οποία περίπτωση το αλφαριθμητικό πρέπει να είναι αυστηρά **“wrap_content”** ή **“match_parent”**, αλλιώς προκύπτει σφάλμα.

Αν δεν έχουμε κάποιο από αυτά τα δύο attributes, τότε κάνουμε έναν περαιτέρω έλεγχο για το αν το τωρινό attribute είναι οποιοδήποτε από αυτά που δέχονται αλφαριθμητικό σαν τιμή και αν όχι, τότε προκαλείται σφάλμα.

Τέλος αν έχουμε το attribute **checkedButton**, αποθηκεύουμε την τιμή που του ανατίθεται στην μεταβλητή **currentCheckedButton** για μετέπειτα χρήση στον αντίστοιχο έλεγχο.

void idCheck()

Αυτή η συνάρτηση διαχειρίζεται όλους τους ελέγχους που αφορούν το attribute **id**. Καταρχάς γίνεται ένας μικρός έλεγχος αν το τωρινό attribute με αλφαριθμητική τιμή, για το οποίο κλήθηκε αυτή η συνάρτηση είναι όντως το **id**, αλλιώς αυτή σταματά.

Η υπόλοιπη συνάρτηση χωρίζεται σε δύο μεγάλα μέρη. Το πρώτο αφορά τα **id** γενικά, ανεξαρτήτως στοιχείου και το δεύτερο είναι αποκλειστικά για τα **id** από **RadioButton** στοιχεία.

Για το πρώτο μέρος:

Θέλουμε να προστεθεί το **id** που μόλις διαβάστηκε στη λίστα όλων των **id** που έχουν διαβαστεί ως τώρα, την **idList**. Η γενική διαδικασία είναι παρόμοια με την προσθήκη tags στο stack και attributes στο **attributeList**, οπότε δεν θα αναλυθεί ξανά. Υπάρχει μια σημαντική διαφορά όμως. Προτού το **id** εισαχθεί πρέπει να γίνει έλεγχος για το αν το **id** είναι μοναδικό ανάμεσα σε όλα τα άλλα που μπορεί να έχουν διαβασθεί. Αν οποιαδήποτε δύο **id** ταυτίζονται, τότε προκύπτει σφάλμα. Στο τέλος αυξάνεται κατά 1 η μεταβλητή **idCount** που αντιπροσωπεύει το πλήθος των **id**.

Για το δεύτερο μέρος:

Αν το στοιχείου του οποίο το **id** διαχειριζόμαστε είναι **RadioButton**, τότε θέλουμε να προστεθεί το **id** περαιτέρω και στη λίστα όλων των **id** από **RadioButton** στοιχεία εντός του συγκεκριμένου **RadioGroup** που βρίσκεται ο έλεγχος της γλώσσας, την **buttonIds**. Η γενική διαχείριση είναι ξανά ίδια με του προηγούμενου μέρους οπότε δεν αναλύεται περαιτέρω.

void radioGroupCheck()

Αυτή η συνάρτηση αφορά ελέγχους αποκλειστικά στα **RadioGroup** στοιχεία. Πρώτα, ελέγχουμε αν το πλήθος **RadioButton** εντός του **RadioGroup** ταυτίζεται με την τιμή του attribute **button_quantity**, του οποίου η τιμή έχει αποθηκευτεί στην **currentButtonQuantity**. Αν όχι, τότε προκύπτει σφάλμα.

Έπειτα, πρέπει να γίνει έλεγχος για το αν υπάρχει **RadioButton** με **id** που ταυτίζεται με την τιμή του attribute **checkedButton**. Γι' αυτός το σκοπό χρησιμοποιείται η μεταβλητή **buttonIds**. Αν δεν βρεθεί κανένα id που να ικανοποιεί το attribute, τότε προκύπτει σφάλμα.

void progressBarCheck()

Αυτή η συνάρτηση είναι αρκετά απλή και σύντομη, αφού εκτελεί μόνο έναν σύντομο έλεγχο. Πρόκειται για τον έλεγχο της τιμής του attribute **progress** του στοιχείου **ProgressBar**, η οποία πρέπει να κυμαίνεται μεταξύ του 0 και της τιμής **max**. Αν δεν τηρείται αυτό, τότε προκύπτει σφάλμα.

void myErrorPrint(int messageCode)

Πρόκειται για μια πολύ απλή συνάρτηση, παρά το μέγεθος της. Με βάση έναν ακέραιο αριθμό που δίνεται σαν είσοδος σε αυτή, κι αν δεν έχει ήδη προκύψει κάποιο άλλο σφάλμα με βάση τη σημαία **copyInputFlag**, εκτυπώνει το κατάλληλο μήνυμα σφάλματος που αντιστοιχεί στον ακέραιο που δόθηκε.

3.4 Γνωστά προβλήματα και Bugs

- Αν στο τύπο δεδομένων **STRING** που αντιστοιχεί σε κάποιο attribute χρησιμοποιηθούν μόνο αριθμοί και τίποτα άλλο προκύπτει πρόβλημα, ενώ κανονικά θα έπρεπε να αναγνωρίζεται ως αλφαριθμητικό. Προφανώς δεν είναι εντελώς σωστά ορισμένες οι κανονικές εκφράσεις που αντιστοιχούν στα **INTEGER** και **STRING**.
- Ενώ έχει γίνει προσπάθεια για να εμφανίζεται ένα μόνο μήνυμα σφάλματος τη φορά ακόμη κι αν συμβαίνουν πολλαπλά ταυτόχρονα, έχει τύχει μεμονωμένη περίπτωση όπου εμφανίστηκαν δύο μηνύματα μαζί. Δυστυχώς δεν υπήρξε αρκετός χρόνος για debugging αυτού του ζητήματος.

4. Πηγαίος κώδικας λεκτικού αναλυτή – (scanner.l)

```
%option noyywrap
%s COMMENT
%{
#include "parser.tab.h"
#include <stdio.h>
#include <string.h>

extern void myErrorPrint(int messageCode);

int copyInputFlag = 0;
int currentLine = 1;
int firstLinePrint = 0;
int hyphensInARow = 0;

void checkCharacters();
void printProgram();
void printLineNumbers();
%}

%%

<INITIAL>"<!--" {BEGIN(COMMENT); printProgram();}

<COMMENT>{
    "-->" {BEGIN(INITIAL); hyphensInARow = 0; printProgram();}
    . {checkCharacters();}
    "\n" {printProgram(); currentLine++; printLineNumbers();}
}

<INITIAL>{
    "\n" {printProgram(); currentLine++; printLineNumbers();}
    " " {printProgram();}
    "\t" {printProgram();}

    "<" {printProgram(); return '<';}
    ">" {printProgram(); return '>';}
    "/" {printProgram(); return '/';}
    ":" {printProgram(); return ':';}
    "!" {printProgram(); return '!';}
    "=" {printProgram(); return '=';}
    \" {printProgram(); return '"';}

    "LinearLayout" {printProgram(); return ELEMENT_NAME;}
    "RelativeLayout" {printProgram(); return ELEMENT_NAME;}
    "TextView" {printProgram(); return ELEMENT_NAME;}
    "ImageView" {printProgram(); return ELEMENT_NAME;}
    "Button" {printProgram(); return ELEMENT_NAME;}
    "RadioButton" {printProgram(); return ELEMENT_NAME;}
    "RadioGroup" {printProgram(); return ELEMENT_NAME;}
    "ProgressBar" {printProgram(); return ELEMENT_NAME;}

    "android" {printProgram(); return ATTRIBUTE_START;}
```

```
    "layout_width" {printProgram(); return ATTRIBUTE_NAME;}
    "layout_height" {printProgram(); return ATTRIBUTE_NAME;}
    "orientation" {printProgram(); return ATTRIBUTE_NAME;}
    "id" {printProgram(); return ATTRIBUTE_NAME;}
    "text" {printProgram(); return ATTRIBUTE_NAME;}
    "textColor" {printProgram(); return ATTRIBUTE_NAME;}
    "src" {printProgram(); return ATTRIBUTE_NAME;}
    "padding" {printProgram(); return ATTRIBUTE_NAME;}
    "checkedButton" {printProgram(); return ATTRIBUTE_NAME;}
    "max" {printProgram(); return ATTRIBUTE_NAME;}
    "progress" {printProgram(); return ATTRIBUTE_NAME;}
    "button_quantity" {printProgram(); return ATTRIBUTE_NAME;}

    [0-9]+ {printProgram(); return INTEGER;}
    [0-9A-Za-z_ ]+ {printProgram(); return STRING;}
    . {myErrorPrint(15);}
}
%%

void checkCharacters()
{
    if(yytext[0] == '-')
    {
        hyphensInARow++;
        printProgram();
    }
    else
    {
        hyphensInARow = 0;
        printProgram();
    }

    if(hyphensInARow >= 2)
        myErrorPrint(16);
}

void printProgram()
{
    if(firstLinePrint == 0)
    {
        printf("  1.|");
        firstLinePrint = 1;
    }

    if(copyInputFlag == 0)
        printf(yytext);
}

void printLineNumbers()
{
    if(copyInputFlag == 0)
    {
        if(currentLine < 10)
            printf("  %d.|", currentLine);
        else if(currentLine >=10 && currentLine < 100)
            printf(" %d.|", currentLine);
        else
```

```
            printf("%d.|", currentLine);
    }
}
```

5. Πηγαίος κώδικας συντακτικού αναλυτή – (parser.y)

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

extern int yylex();
extern char * yytext;
extern int yyleng;
extern int copyInputFlag;
extern int currentLine;

char * inputFormat = "------------------------------------------------------
-\n                    Input program:\n-----------------------------------
-------------------\n\n\n";
char * outputFormat = "\n\n\n-------------------------------------------------
--------\n                          Output:\n---------------------------------
-----------------------\n\n\n";

void yyerror(char const *);

void myErrorPrint(int messageCode);

void pushStack();
void popStack();
void compTags();

void checkAttribute();
void storeAttribute();
void necessaryAttributes();
int assignTagNumber();
void attributeValueCheck(int type);
void idCheck();
void radioGroupCheck();
void progressBarCheck();

int linearContentFlag = 0;

char ** stack;
int stackElements = 0;

char ** attributeList;
int attributeCount = 0;

char ** idList;
int idCount = 0;

char * currentCheckedButton;
char ** buttonIds;
int buttonCount = 0;
int currentButtonQuantity = 0;

int currentProgress = 0;
int currentMax = 100;
```

```
%}

%token '<' '>' '/' ':' '!' '=' '"'
%token ELEMENT_NAME ATTRIBUTE_START ATTRIBUTE_NAME
%token INTEGER STRING

%%

program:
    %empty {printf("%sThe input file is empty.\n\n", outputFormat);}
    | element_start {if(copyInputFlag == 0)
printf("%sCOMPILED_SUCCESSFULLY\n\n", outputFormat);}
    ;

element_start:
    '<' ELEMENT_NAME {pushStack();} element_middle
    ;

element_middle:
    attributes {necessaryAttributes(); free(attributeList); attributeCount =
0;} element_end
    ;

element_end:
    '>' content '<' '/' ELEMENT_NAME {compTags();} '>'
    | '>' '<' '/' ELEMENT_NAME {compTags();} '>'
    | '/' '>' {popStack();}
    ;

content:
    content element_start
    | element_start
    ;

attributes:
    attributes attribute
    | attribute
    ;

attribute:
    ATTRIBUTE_START ':' ATTRIBUTE_NAME {checkAttribute(); storeAttribute();}
attribute_value
    ;

attribute_value:
    '=' '"' INTEGER {attributeValueCheck(0);} '"'
    | '=' '"' STRING {attributeValueCheck(1); idCheck();} '"'
    ;

%%
void attributeValueCheck(int type)
{
    if(type == 0)
    {
        if(strcmp(attributeList[attributeCount - 1], "layout_height") != 0
            && strcmp(attributeList[attributeCount - 1], "layout_width") != 0
            && strcmp(attributeList[attributeCount - 1], "padding") != 0
```

```
            && strcmp(attributeList[attributeCount - 1], "progress") != 0
            && strcmp(attributeList[attributeCount - 1], "max") != 0
            && strcmp(attributeList[attributeCount - 1], "button_quantity")
!= 0)
        {
            myErrorPrint(5);
        }

        if(strcmp(attributeList[attributeCount - 1], "progress") == 0)
            currentProgress = atoi(yytext);

        if(strcmp(attributeList[attributeCount - 1], "max") == 0)
            currentMax = atoi(yytext);

        if(strcmp(attributeList[attributeCount - 1], "button_quantity") == 0)
            currentButtonQuantity = atoi(yytext);
    }
    else
    {
        if(strcmp(attributeList[attributeCount - 1],"layout_width") == 0 ||
strcmp(attributeList[attributeCount - 1],"layout_height") == 0)
        {
            if(strcmp(yytext,"wrap_content") != 0 &&
strcmp(yytext,"match_parent") != 0)
            {
                myErrorPrint(5);
            }
            return;
        }
        else if(strcmp(attributeList[attributeCount - 1], "padding") == 0
            || strcmp(attributeList[attributeCount - 1], "progress") == 0
            || strcmp(attributeList[attributeCount - 1], "max") == 0
            || strcmp(attributeList[attributeCount - 1], "button_quantity")
== 0)
        {
            myErrorPrint(5);
        }

        if(strcmp(attributeList[attributeCount - 1], "checkedButton") == 0)
        {
            char * temp = (char *) malloc(sizeof(yytext));
            strcpy(temp, yytext);
            currentCheckedButton = temp;
        }
    }
}

void idCheck()
{
    if(strcmp(attributeList[attributeCount - 1], "id") != 0)
        return;

    char * temp = (char *) malloc(sizeof(yytext));
    strcpy(temp, yytext);

    if(idCount == 0)
    {
```

```
        idList = (char **) malloc(sizeof(char *));
        idList[0] = temp;
    }
    else
    {
        for(int i = 0; i < idCount; i++)
        {
            if(strcmp(idList[i], temp) == 0)
            {
                myErrorPrint(6);
                return;
            }
        }

        char ** newList = (char **) malloc(sizeof(char *) * (idCount + 1));

        for(int i = 0; i < idCount; i++)
            newList[i] = idList[i];

        newList[idCount] = temp;

        free(idList);
        idList = newList;
    }
    idCount++;

    if(strcmp(stack[stackElements - 1], "RadioButton") == 0)
    {
        if(buttonCount == 0)
        {
            buttonIds = (char **) malloc(sizeof(char *));
            buttonIds[0] = temp;
        }
        else
        {
            for(int i = 0; i < buttonCount; i++)
            {
                if(strcmp(buttonIds[i], temp) == 0)
                {
                    myErrorPrint(6);
                    return;
                }
            }

            char ** newList = (char **) malloc(sizeof(char *) * (buttonCount
+ 1));

            for(int i = 0; i < buttonCount; i++)
                newList[i] = buttonIds[i];

            newList[buttonCount] = temp;

            free(buttonIds);
            buttonIds = newList;
        }
        buttonCount++;
    }
```

```
}

void checkAttribute()
{
    int tagNumber = assignTagNumber();

    if(strcmp(yytext, "layout_width") != 0 && strcmp(yytext, "layout_height")
!= 0 && strcmp(yytext, "id") != 0)
    {
        switch(tagNumber)
        {
            case 0:
                myErrorPrint(2);
                break;
            case 1:
                if(strcmp(yytext, "orientation") != 0)
                    myErrorPrint(2);
                break;
            case 2:
                if(strcmp(yytext, "text") != 0 && strcmp(yytext, "textColor")
!= 0)
                    myErrorPrint(2);
                break;
            case 3:
                if(strcmp(yytext, "src") != 0 && strcmp(yytext, "padding") !=
0)
                    myErrorPrint(2);
                break;
            case 4:
                if(strcmp(yytext, "text") != 0 && strcmp(yytext, "padding")
!= 0)
                    myErrorPrint(2);
                break;
            case 5:
                if(strcmp(yytext, "checkedButton") != 0 && strcmp(yytext,
"button_quantity") != 0)
                    myErrorPrint(2);
                break;
            case 6:
                if(strcmp(yytext, "text") != 0)
                    myErrorPrint(2);
                break;
            case 7:
                if(strcmp(yytext, "max") != 0 && strcmp(yytext, "progress")
!= 0)
                    myErrorPrint(2);
                break;
        }

    }
}

void storeAttribute()
{
    char * temp = (char *) malloc(sizeof(yytext));
    strcpy(temp, yytext);
```

```
    if(attributeCount == 0)
    {
        attributeList = (char **) malloc(sizeof(char *));
        attributeList[0] = temp;
    }
    else
    {
        for(int i = 0; i < attributeCount; i++)
        {
            if(strcmp(attributeList[i],temp) == 0)
            {
                myErrorPrint(3);
                return;
            }
        }

        char ** newList = (char **) malloc(sizeof(char *) * (attributeCount +
1));

        for(int i = 0; i < attributeCount; i++)
            newList[i] = attributeList[i];

        newList[attributeCount] = temp;

        free(attributeList);
        attributeList = newList;
    }
    attributeCount++;
}

void necessaryAttributes()
{
    int flag1 = 0;
    int flag2 = 0;
    int flag3 = 0;

    int tagNumber = assignTagNumber();

    if(attributeCount < 2)
    {
        myErrorPrint(4);
        return;
    }

    for(int i = 0; i < attributeCount; i++)
    {
        if(strcmp(attributeList[i], "layout_width") == 0)
        {
            flag1 = 1;
            break;
        }
    }

    for(int i = 0; i < attributeCount; i++)
    {
        if(strcmp(attributeList[i], "layout_height") == 0)
        {
```

```
            flag2 = 1;
            break;
        }
    }

    if(tagNumber == 3)
    {
        for(int i = 0; i < attributeCount; i++)
        {
            if(strcmp(attributeList[i], "src") == 0)
            {
                flag3 = 1;
                break;
            }
        }

    }
    else if(tagNumber == 5)
    {
        for(int i = 0; i < attributeCount; i++)
        {
            if(strcmp(attributeList[i], "button_quantity") == 0)
            {
                flag3 = 1;
                break;
            }
        }
    }
    else if(tagNumber == 2 || tagNumber == 4 || tagNumber == 6)
    {
        for(int i = 0; i < attributeCount; i++)
        {
            if(strcmp(attributeList[i], "text") == 0)
            {
                flag3 = 1;
                break;
            }
        }
    }
    else flag3 = 1;

    if(flag1 == 0 || flag2 == 0 || flag3 == 0)
        myErrorPrint(4);
}

int assignTagNumber()
{
    if(strcmp(stack[stackElements - 1], "RelativeLayout") == 0)
        return 0;
    else if(strcmp(stack[stackElements - 1], "LinearLayout") == 0)
        return 1;
    else if(strcmp(stack[stackElements - 1], "TextView") == 0)
        return 2;
    else if(strcmp(stack[stackElements - 1], "ImageView") == 0)
        return 3;
    else if(strcmp(stack[stackElements - 1], "Button") == 0)
        return 4;
```

```
    else if(strcmp(stack[stackElements - 1], "RadioGroup") == 0)
        return 5;
    else if(strcmp(stack[stackElements - 1], "RadioButton") == 0)
        return 6;
    else if(strcmp(stack[stackElements - 1], "ProgressBar") == 0)
        return 7;
}

void pushStack()
{
    char * temp = (char *) malloc(sizeof(yytext));
    strcpy(temp, yytext);

    if(strcmp(temp, "LinearLayout") == 0)
        linearContentFlag = 1;
    else
        linearContentFlag = 0;

    if(stackElements == 0)
    {
        if(strcmp("LinearLayout", temp) != 0 && strcmp("RelativeLayout",
temp) != 0)
            myErrorPrint(7);

        stack = (char **) malloc(sizeof(char *));
        stack[0] = temp;
    }
    else
    {
        if(strcmp("RadioButton",temp) == 0 &&
strcmp("RadioGroup",stack[stackElements - 1]) != 0)
            myErrorPrint(9);

        if(strcmp("RadioButton",temp) != 0 &&
strcmp("RadioGroup",stack[stackElements - 1]) == 0)
            myErrorPrint(10);

        char ** newStack = (char **) malloc(sizeof(char *) * (stackElements +
1));

        for(int i = 0; i < stackElements; i++)
            newStack[i] = stack[i];

        newStack[stackElements] = temp;

        free(stack);
        stack = newStack;
    }
    stackElements++;
}

void popStack(){
    if(strcmp("LinearLayout", stack[stackElements - 1]) == 0 &&
linearContentFlag == 1)
        myErrorPrint(8);

    if(strcmp(stack[stackElements - 1], "RadioGroup") == 0)
```

```
    {
        radioGroupCheck();
        free(buttonIds);
        free(currentCheckedButton);
        buttonCount = 0;
        currentButtonQuantity = 0;
    }

    if(strcmp(stack[stackElements - 1], "ProgressBar") == 0)
    {
        progressBarCheck();
        currentProgress = 0;
        currentMax = 100;
    }

    char ** newStack = (char **) malloc(sizeof(char *) * (stackElements -
1));

    for(int i = 0; i < stackElements - 1; i++)
        newStack[i] = stack[i];

    free(stack);
    stack = newStack;
    stackElements--;
}

void radioGroupCheck()
{
    if(currentButtonQuantity != buttonCount)
        myErrorPrint(14);

    if(currentCheckedButton)
    {
        if(buttonCount == 0)
        {
            myErrorPrint(12);
            return;
        }

        for(int i = 0; i < buttonCount; i++)
        {
            if(strcmp(currentCheckedButton, buttonIds[i]) == 0)
                return;
        }
        myErrorPrint(13);
    }
}

void progressBarCheck()
{
    if(currentProgress < 0 || currentProgress > currentMax)
        myErrorPrint(11);
}

void compTags()
{
    if(strcmp(stack[stackElements - 1], yytext) == 0)
```

```
        popStack();
    else
        myErrorPrint(1);
}

void myErrorPrint(int messageCode)
{
    if(copyInputFlag == 0)
    {
        char * errorText = (char *) malloc(sizeof(char *));

        switch(messageCode)
        {
            case 1:
                errorText = "START AND END TAGS DONT MATCH";
                break;
            case 2:
                errorText = "TAG CONTAINS IMPROPER ATTRIBUTE";
                break;
            case 3:
                errorText = "MULTIPLE INSTANCES OF ATTRIBUTE ARE NOT
ALLOWED";
                break;
            case 4:
                errorText = "TAG DOES NOT CONTAIN NECESSARY ATTRIBUTES";
                break;
            case 5:
                errorText = "WRONG DATA TYPE IN ATTRIBUTE VALUE";
                break;
            case 6:
                errorText = "ID VALUES MUST BE UNIQUE";
                break;
            case 7:
                errorText = "ROOT TAG MUST BE EITHER LINEARLAYOUT OR
RELATIVELAYOUT";
                break;
            case 8:
                errorText = "LINEAR LAYOUT TAG CANNOT BE EMPTY";
                break;
            case 9:
                errorText = "RADIO BUTTON TAGS CAN APPEAR ONLY INSIDE RADIO
GROUP TAGS";
                break;
            case 10:
                errorText = "RADIO GROUP TAGS CAN CONTAIN ONLY RADIO BUTTON
TAGS";
                break;
            case 11:
                errorText = "PROGRESS ATTRIBUTE VALUE MUST BETWEEN 0 AND
MAX";
                break;
            case 12:
                errorText = "RADIO GROUP MUST CONTAIN AT LEAST ONE RADIO
BUTTON WITH DEFINED ID FOR THE CHECKED BUTTON ATTRIBUTE TO BE USED";
                break;
            case 13:
```

```
                errorText = "CHECKED BUTTON NEEDS TO MATCH THE ID OF A RADIO
BUTTON INSIDE THE RADIO GROUP";
                break;
            case 14:
                errorText = "RADIO GROUP MUST CONTAIN AS MANY RADIO BUTTONS
AS INDICATED BY THE BUTTON QUANTITY ATTRIBUTE";
                break;
            case 15:
                errorText = "IMPROPER CHARACTER";
                break;
            case 16:
                errorText = "COMMENT CANNOT CONTAIN TWO OR MORE HYPHENS IN A
ROW";
                break;
        }

        printf("%sLine %d: %s\n\n", outputFormat, currentLine, errorText);

        copyInputFlag = 1;
    }
}

void yyerror(char const *msg)
{
    printf("%sLine %d: UNDEFINED SYNTAX ERROR\n\n", outputFormat,
currentLine);
}

int main()
{
    printf("%s", inputFormat);
    yyparse();
    return 0;
}
```