PROJECT ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ 2022

ΛΕΚΤΙΚΟΣ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ JSON ΑΡΧΕΙΟΥ ΣΕ FLEX – BISON

ΣΤΟΙΧΕΙΑ ΜΕΛΩΝ ΟΜΑΔΑΣ

- Βυθούλκα Βαρβάρα, 1059552, 5ο έτος, up1059552@upnet.gr
- Παπαδοπούλου Μαρία, 1072494, 3ο έτος, up1072494@upnet.gr
- Σούρλας Ζήσης, 1072477, 3ο έτος, up1072477@upnet.gr
- Σφήκας Θοδωρής, 1072550, 3ο έτος, up1072550@upnet.gr

ΠΑΡΑΔΟΧΕΣ

- Μία από τις βασικές παραδοχές που έγιναν κατά την διάρκεια υλοποίησης του project είναι στην διαχείριση των κενών. Όπως παρατίθεται στην εκφώνηση τα κενά οφείλουν να αγνοούνται όπου παρατηρούνται , φαινόμενο που πραγματοποιείται κατά βάση μέσω του FLEX όπως θα αναλύσουμε στην συνέχεια. Η παραδοχή συναντάται στα κενά μέσα στα json strings. Θεωρούμε πως όπως σε κάθε γλώσσα προγραμματισμού τα κενά στα strings έχουν σημασία και το string: “last” διαφέρει από το string “ last ”. Κρίνουμε πως μέσω αυτής της παραδοχής συμβαδίζουμε περισσότερο με τις βασικές ιδιότητες των strings στα Json αρχεία γενικότερα.
- Η BNF που εμφανίζεται είναι βασισμένη στην απλή και όχι στην extended BNF (οπότε τα σύμβολα (), [], {} δεν έχουν κάποια σημασία, είναι απλά σύμβολα των κανόνων).

ΤΕΛΙΚΑ ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ

Τα τελικά αρχεία που χρησιμοποιήθηκαν είναι τα: flex_final.l για τον flex και bison_final.y για τον bison. Στο zip περιλαμβάνεται ένα επιπλέον header file που γίνεται include από τα δύο παραπάνω αρχεία.

BNF

Όσες από τις παρακάτω μεταβλητές εμφανίζονται με κεφαλαία γράμματα, δεν αποτελούν μεταβλητές, αλλά τερματικά σύμβολα. Επειδή αυτά ορίζονται στο λεξικό αναλυτή - flex, δεν δίνουμε και εδώ τον ορισμό τους.

jfile ::= jfile1 | jfile2

jfile1 ::= { last_field , active_field }
jfile2::= { content_field , tpages_field, telements_field, lastb_field, numofelements_field , sort_field, first_field, size_field, number_field }

last_field::= last : jsonObj2

active_field ::= active : { gameid_field, drawid_field, drawtime_field, status_field, drawbreak_field, visualdraw_field, pricepoints_field, prizecat_field, wagerstat_field}

gameid_field ::= gameId : GAMEID_VALUE| gameId : POS_INT
drawid_field ::= drawId : POS_INT
drawtime_field ::= drawTime : POS_INT
status_field ::= status : STRING
drawbreak_field ::= drawBreak : POS_INT
visualdraw_field ::= visualDraw : POS_INT
pricepoints_field ::= pricePoint : { amountfield }
amountfield ::= amount : POS_INTR
win_num_field ::= winningNumbers : { list_field, bonus_field }
bonus_field ::= bonus : [POS_INT]
list_field ::= list : posintArray

posintArray ::= [WIN_POS_INT , secintArray]
secintArray ::= WIN_POS_INT, secintArray | WIN_POS_INT | POS_INT

wagerstat_field ::= wagerStatistics : { columns_field, wagers_field, addon_field }
columns_field ::= columns : POS_INT
wagers_field ::= wagers : POS_INT
addon_field = addOn : generaljsonArray

prizecat_field ::= prizeCategories : jsonArray

jsonArray ::= [jsonObj, secjsonArray]
secjsonArray ::= jsonObj, secjsonArray | jsonObj

generaljsonArray ::= [STRING , generaljsonArray2] | [POS_INT , generaljsonArray2] | [POS_INTR , generaljsonArray2] | [STRING] | [POS_INT] | [POS_INTR] | [ ]
generaljsonArray2 ::= STRING , generaljsonArray2 | POS_INT , generaljsonArray2 | POS_INTR , generaljsonArray2 | STRING | POS_INT | POS_INTR

jsonObj ::= {id1_field, divident_field, winners_field, distributed_field, jackpot_field, fixed_field, categorytype_field, gametype_field, mindistr_field} | {id2_field, divident_field, winners_field, distributed_field, jackpot_field, fixed_field, categorytype_field, gametype_field}

id1_field ::= id : 1
id2_field ::= id : POS_INT28
divident_field ::= divident : POS_INTR
winners_field ::= winners : POS_INT | winners : POS_INT28 | winners : 0 | winners : 1
distributed_field ::= distributed : POS_INTR
jackpot_field ::= jackpot : POS_INTR
fixed_field ::= fixed: POS_INTR
categorytype_field ::= categoryType : 0 | categoryType : 1
gametype_field ::= gametype : STRING

mindistr_field ::= minimumDistributed : POS_INTR

tpages_field ::= totalPages : POS_INT
telements_field ::= totalElements : POS_INT
numofelements_field :: = numberOfElements : POS_INT
size_field ::= size : POS_INT
number_field ::= number : POS_INT
content_field ::= content : jsonArray
lastb_field ::= last : BOOLEAN
first_field ::= first : BOOLEAN
jsonArray2 ::= [jsonObj2 , secjsonArray2] | [jsonObj2] | [ ]
secjsonArray2 ::= jsonObj2 , secjsonArray2 | jsonObj2

jsonObj2 ::= {gameid_field , drawid_field , drawtime_field , status_field , drawbreak_field , visualdraw_field , pricepoints_field , win_num_field , prizecat_field , wagerstat_field}

sort_field ::= sort : [sortArray]
sortArray ::= {direction_field , property_field , ignorecase_field , nullhandling_field , descending_field , ascending_field }
direction_field ::= direction : STRING
property_field ::= property : PROPERTY_STRING | property : STRING
ignorecase_field ::= ignoreCase : BOOLEAN
nullhandling_field ::= nullHandling : STRING
descending_field ::= descending : BOOLEAN
ascending_field ::= ascending : BOOLEAN

ΛΕΞΙΚΗ ΑΝΑΛΥΣΗ – FLEX

Αρχικά κατά την μελέτη της εργασίας μας στο FLEX παρατηρούνται τα options noyywrap , yylineno και nodedefault. Αρχικά το option noyywrap ορίζει στον flex να μην χρησιμοποιήσει το απαρχαιωμένο yywrapping και προτείνεται να χρησιμοποιείται σχεδόν πάντα κατά την δημιουργία flex lexers στο βιβλίο του O'Reilly. Συνεχίζοντας το option yylineno υλοποιεί αυτόματα την μέτρηση της γραμμής στην οποία βρίσκεται ο lexer απο το input file. Θα χρησιμοποιεί κατάλληλα κατά την διαδικασία του error handling η οποία αναλύεται στην συνέχεια. Τέλος το option nodedefault ορίζει στον Flex πως δεν πρέπει να υπάρχει περίπτωση εισόδου η οποία να μην αναγνωρίζεται και να παραλειφθεί. Το format json στο οποίο υλοποιούμε την λεκτική και την συντακτική ανάλυση είναι συγκεκριμένο και δεν επιτρέπει περιθώρια διαφοροποίησης έξω των προδιαγραφών που μας δίνονται.

Δημιουργήθηκαν οι απαιτούμενες κανονικές εκφράσεις (regexes ή patterns) για την αναγνώριση των τύπων δεδομένων που εμφανίζονται στα αρχεία εισόδου καθώς και για κάποιες αναγκαίες ειδικές τους περιπτώσεις (πχ PROPERTY_STRING, POS_INT28 για id στο [1,8] και id≠1). Όταν κάποιο pattern γίνει match, τότε η είσοδος τυπώνεται στην έξοδο και επιστρέφεται το αντίστοιχο token στο bison.

Ορισμένες κανονικές εκφράσεις μπορούν να γίνουν match μόνο όταν ο λεκτικός αναλυτής βρίσκεται σε κάποια κατάσταση. Εξηγούμε αρχικά τις καταστάσεις που ορίσαμε:

%s MORE_POSINTS κατάσταση η οποία αναφέρεται σε ύπαρξη περισσότερων θετικών ακεραίων: στα στοιχεία του πίνακα *prizeCategories*, που υπάρχει εντός του στοιχείου *last* στο *last_result.json* και εντός του πίνακα *content* του *range_result.json*, υπάρχουν πεδία τύπου θετικού ακεραίου με συγκεκριμένες τιμές. Τα πεδία αυτά είναι *categoryType = 0 ή 1* και η περίπτωση για *id =1*. Αν αναγνωρίσουμε την τιμή αυτών των πεδίων με την υπάρχουσα, γενική κανονική έκφραση για θετικούς ακεραίους, POS_INT θα γίνουν αποδεκτές πολλές παραπάνω τιμές από τις ζητούμενες. Άρα, χρειάζεται να επιστρέψουμε διαφορετικό token όταν αναγνωριστούν αυτά τα πεδία. Αυτός είναι και ο ρόλος της συγκεκριμένης κατάστασης: αρχικοποιείται μόλις αναγνωριστούν τα ονόματα των πεδίων και αν αναγνωρίσει επιτρεπτή τιμή, μεταφέρει τον λεκτικό αναλυτή στην αρχική κατάσταση και ύστερα επιστρέφει την τιμή που διάβασε (0 ή 1) σαν token στον bison, σύμφωνα με τους κανόνες που έχουμε ορίσει.
%s PROPERTY ανάλογη λειτουργία με την MORE_POSINTS, με σκοπό τον περιορισμό των επιτρεπόμενων strings στο πεδίο *property*.
%s GAMEID ανάλογη λειτουργία με την MORE_POSINTS, με σκοπό τον περιορισμό των τιμών του πεδίου *gameId* στις επιτρεπτές.
%s WIN ανάλογη λειτουργία με την MORE_POSINTS, με σκοπό τον περιορισμό των τιμών των στοιχείων του πίνακα *winningNumbers* στις επιτρεπτές. Μόνη διαφορά είναι ότι η επιστροφή του λεκτικού αναλυτή στην αρχική κατάσταση, γίνεται μόλις αναγνωριστεί το επόμενο στοιχείο του πίνακα *“list”*, δηλαδή το *“bonus”*, κι αυτό γιατί πρέπει όλοι οι αριθμοί εντός του πίνακα να βρίσκονται εντός του επιτρεπτού εύρους τιμών, οπότε πρέπει πρώτα να εξαντληθεί ο πίνακας και στη συνέχεια να αλλάξει κατάσταση ο αναλυτής.

Σημειώνεται ότι όλες οι καταστάσεις αρχικοποιούνται μόλις αναγνωριστεί το όνομα του πεδίου στο οποίο χρειάζεται να γίνει περιορισμός τιμών, με εξαίρεση την τελευταία, που αρχικοποιείται μόλις αναγνωριστεί το όνομα του πεδίου *“list”*. Η εξαίρεση αυτή θα μπορούσε να μην υπάρχει και δεν επηρεάζει κάπως τον κώδικα.

Για την ορθή λειτουργία του λεκτικού αναλυτή έχει χρησιμοποιηθεί και ένα flag με όνομα *stringflag*. Σκοπός αυτού του flag είναι να ξεχωρίσει αν η λέξη **“active”** στο αρχείο εισόδου, είναι όνομα στοιχείου (το στοιχείο *active* στο *last_result.json*) ή αν πρόκειται για την τιμή του πεδίου *status*, καθώς επιστρέφεται διαφορετικό token στο bison για κάθε περίπτωση (active για την πρώτη και STRING για τη δεύτερη). Για την επίτευξη των παραπάνω, το flag αρχικοποιείται με 1, δηλώνοντας ότι τα strings που θα αναγνωριστούν όσο έχει αυτή την τιμή, αντιστοιχούν σε ονόματα πεδίων/στοιχείων στα αρχεία εισόδου (γι’ αυτό και οι συνθήκες εντός του STRING action είναι ορισμένες ως «*εάν το string είναι <κάποιο string> και πρόκειται για όνομα πεδίου*», με τη χρήση της strcmp() και του stringflag αντίστοιχα). Μόλις αναγνωριστεί το όνομα του πεδίου *“status”*, το flag παίρνει την τιμή 0, δηλώνοντας πως ότι αναγνωρίζεται τώρα, είναι αλφαριθμητικό που αποτελεί την τιμή του πεδίου. Έτσι, δεν ισχύει καμία από τις συνθήκες εντός του STRING action. Ο κώδικας καταλήγει στο else, όπου το flag ξαναπαίρνει την τιμή 1 και ο

λεκτικός αναλυτής επιστρέφει STRING στο bison. Για οποιαδήποτε λέξη δεν αποτελεί όνομα πεδίου, δεν μας αφορά η τιμή του *stringflag*, διότι δεν θα ισχύει καμία από τις συνθήκες strcmp εντός του STRING action και θα επιστραφεί απλό STRING.

Γνωρίζουμε ότι η υλοποίηση του STRING action επιφέρει μεγάλο overhead, καθώς στη χειρότερη περίπτωση πρέπει να γίνουν όλοι οι έλεγχοι για να επιστραφεί STRING. Η επιλογή της συγκεκριμένης υλοποίησης βασίστηκε στην καθαρότητα κώδικα που προσφέρει και στο μειωμένο αριθμό κανονικών εκφράσεων που μπορούν να προκύψουν μιας και, αν είχαμε κάθε περίπτωση του String ξεχωριστά, θα έπρεπε να έχουμε ορίσει και ξεχωριστό regular expression για κάθε πεδίο.

ΣΚΕΠΤΙΚΟ ΓΥΡΩ ΑΠΟ ΤΟΝ ΣΧΕΔΙΑΣΜΟ ΤΟΥ BISON

Το σκεπτικό πάνω στο οποίο βασίστηκε ο σχεδιασμός του κώδικα Bison βασίζεται στον διαχωρισμό του αρχείου JSON σε κάποια νοητά επίπεδα. Συγκεκριμένα στο ανώτερο επίπεδο θεωρούμε ότι ο συντακτικός αναλυτής πρέπει να αναγνωρίζει έναν από τους δύο δυνατούς τύπους αρχείων. Εντός του κάθε αρχείου θα πρέπει να αναγνωρίζονται τα στοιχεία τα οποία αποτελούν τη βασική δομή του (πχ. last, active για το αρχείο last_result.json) . Μέσω των κατάλληλων κανόνων τα στοιχεία αυτά αναγνωρίζονται όταν αναγνωριστούν τα στοιχεία τα οποία τα αποτελούν (πχ gameId). Το κάθε στοιχείο αποτελείται από το όνομα του, το οποίο είναι token ταυτοποιούμενο από το λεκτικό αναλυτή, το σύμβολο «:» και την τιμή του η οποία είναι είτε token που ταυτοποιείται από το λεκτικό αναλυτή είτε κάποιος σύνθετος τύπος ο οποίος ορίζεται από περαιτέρω κανόνες του bison. Οι σύνθετοι αυτοί τύποι αφορούν arrays (απλά JSON arrays ή JSON arrays τα οποία περιέχουν στοιχεία τα οποία αφορούν τα συγκεκριμένα αρχεία) τα οποία μάλιστα ορίζονται με δύο κανόνες το κάθε ένα έτσι ώστε να καταστεί δυνατή η ύπαρξη κενών arrays.

ERROR REPORTING + C

Πρέπει να παρατηρήσουμε αρχικά πως ένα ορθό error handling μας δυσκόλεψε ιδιαίτερα κατά την υλοποίηση του project και σε αυτό αφιερώσαμε τον περισσότερο χρόνο, πόρους και προσπάθειες. Μία αρχική υλοποίηση αποτελούσε την δημιουργία επιπλέον κανόνων στον Bison με βασικό στόχο να προβλέψουμε την κάθε δυνατή περίπτωση λάθους και να παράγουμε ένα ορθό και συγκεκριμένο μήνυμα λάθους. Ωστόσο αυτοί οι επιπλέον κανόνες δημιουργούσαν ορισμένα swift/reduce conflicts τα οποία δεν μπορούσαμε σαν ομάδα να εξαλείψουμε. Αξίζει βέβαια να σημειωθεί οτι ο bison αυτόματα επιλύει τα swift/reduce conflicts προς το swift πράγμα που οδηγούσε το πρόγραμμα μας σε ορθό parsing του ζητούμενου json. Ωστόσο επιλέξαμε να αλλάξουμε την κατεύθυνση του error handling ώστε ο κώδικας να είναι πιο πρακτικός, συντηρήσιμος και συμπυκνωμένος.

Οι επιπλέον κανόνες που είχαν προστεθεί διαγράφθηκαν ενώ κρατήθηκε η υλοποίηση που είχε γίνει στον Flex και Bison πάνω στο error location. Η yylineno συγκρατεί την γραμμή στην οποία

βρίσκεται ο lexer ενώ προγραμματίστηκε σύμφωνα και με το βιβλίο του O'Reilly το current column με χρήση του defined YY_USER_ACTION (κώδικας που εκτελειται ύστερα από κάθε matching κανόνα ώστε να γίνεται update η μεταβλητή του column). Ύστερα στον bison βάζοντας το %locations ενεργοποιούμε το location tracking στον parser.

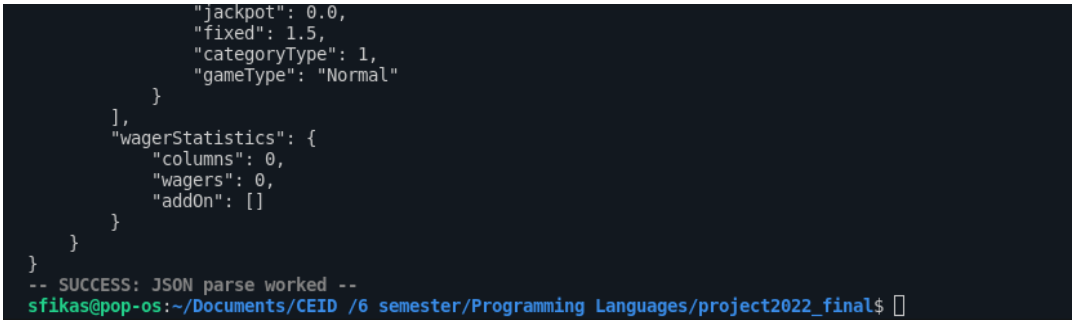
Καθώς το ζητούμενο json αρχείο "παρσάρεται" τυπώνεται το ορθό τμήμα του στο stdout (terminal στην περίπτωση που δεν προσδιορίζουμε κάτι διαφορετικό), ενώ στην περίπτωση λάθους κοκκινίζεται ακριβώς το σημείο που παράγει το λάθος χρησιμοποιώντας τα προαναφερθέντα locations. Στην συνέχεια έχουμε τροποποιήσει την βασική συνάρτηση λάθους yyerror που παράγεται ώστε πρώτον να εκφράζει εμφανώς το μήνυμα συντακτικού λάθους που παρατηρείται και δεύτερον να σημειώνει ορθά την γραμμή και την στήλη αυτού. Δεύτερον έχει πραγματοποιηθεί μία ακόμα συνάρτηση error reporting η οποία καλείται απο εμάς σε ορισμένα embedded actions του bison όποτε χρειάζεται για την παρατήρηση λάθους η οποία αξιοποιεί το YYLTYPE για την τύπωση του λάθους και της τοποθεσίας του. Τέλος ορίζοντας το parse.error verbose όταν ο bison αναγνωρίζει συντακρικό λάθος πέρα απο το generic syntax error παράγει το token που θα έπρεπε να επιστρέψει ο lexer. Το error reporting επομένως γίνεται ορθά γενικευμένα και εκτεταμένα.

Για την συνέχεια του error handling στο τρίτο ερώτημα έπρεπε αρχικά να βεβαιωθούμε για το πλήθος των Winning Numbers και των αντικειμένων του Prize Categories. Για αυτό το στόχο δημιουργήσαμε 2 auxilliary functions στον Bison οι οποίες χρησιμοποιώντας global counters για κάθε αντικείμενο του οποίου θέλουμε να μετρήσουμε τα instances καλούνται με embedded actions. Σε κάθε κάλεσμα της πρώτης ελέγχεται αν το counter ο οποίος και αυξάνεται ξεπερνάει τον αριθμό των max επιτρεπτών instances και σε αυτή την περίπτωση διεγείρεται ορθό μήνυμα λάθους και διακόπτεται η εκτέλεση του προγράμματος. Η Δεύτερη συνάρτηση καλείται με στόχο να ελέγξει την περίπτωση όπου εμφανίζονται λιγότερα instances απο τα ζητούμενα και σε αυτή την περίπτωση τυπώνεται πάλι κατάλληλο μήνυμα λάθους. Τέλος βεβαιωνόμαστε για τις ορθές τιμές των Winning Numbers μέσω των states του flex που προαναφέραμε με στόχο να βεβαιωνόμαστε πως επιτρεπτές τιμές διακυμαίνονται μεταξύ των 1 και 45.

EXPERIMENTS ON JSON FILES

Θα ακολουθήσουν εκτελέσεις του υλοποιημένου project με στόχο την ανάδειξη της ορθής λειτουργίας του και του error handling. Θα ακολουθήσουν ανα δύο φωτογραφίες, του αρχείου εισόδου και της εξόδου του myParser.exe. Στην αρχή θα δείξουμε το αποτέλεσμα του ορθού parsing των αρχείων last_result και range_result ενώ στην συνέχεια το μέρος της εισόδου που είναι λανθασμένο και το αντίστοιχο μήνυμα λάθους που παράγεται. Θα προσπαθήσουμε να αναδείξουμε πολλαπλά είδη λανθασμένου αρχείου και τα αποτελέσματα αυτών.

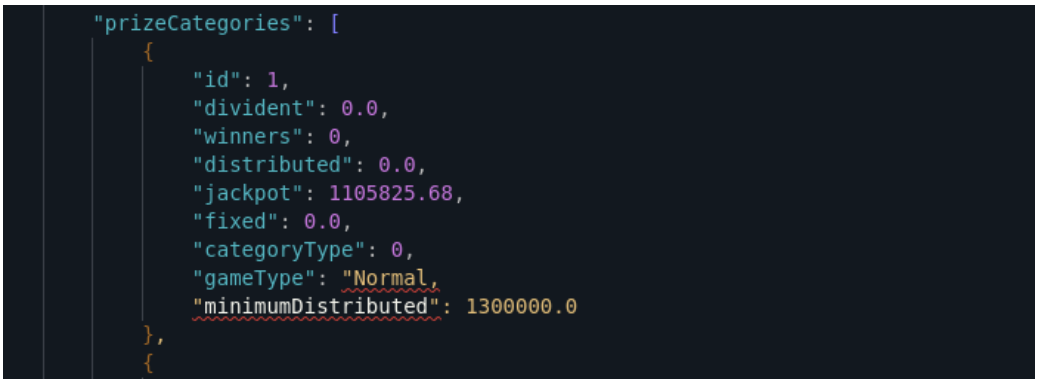
Εισάγοντας το σωστό αρχείο last_result.json, λαμβάνουμε τα παρακάτω αποτελέσματα



Εισάγοντας το αρχείο range_result.json χωρίς λάθη, λαμβάνουμε τα παρακάτω αποτελέσματα:

Λάθος είσοδος last + range για ΤΟ ΕΡΩΤΗΜΑ 1, 2 (SOS Unterminated string, αγκύλες, έλλειψη πεδίου, έλλειψη τιμής)

Ύπαρξη Unterminated String:



Το πρόγραμμα εμφανίζει στην έξοδο:

```
        "prizeCategories": [
            {
                "id": 1,
                "divident": 0.0,
                "winners": 0,
                "distributed": 0.0,
                "jackpot": 1105825.68,
                "fixed": 0.0,
                "categoryType": 0,
                "gameType": "Normal,
                            ^^^^^^^^
ERROR DETECTED: [ Unterminated String ] in line: 133, at columns: 29:37
-- JSON parsing failed --
```

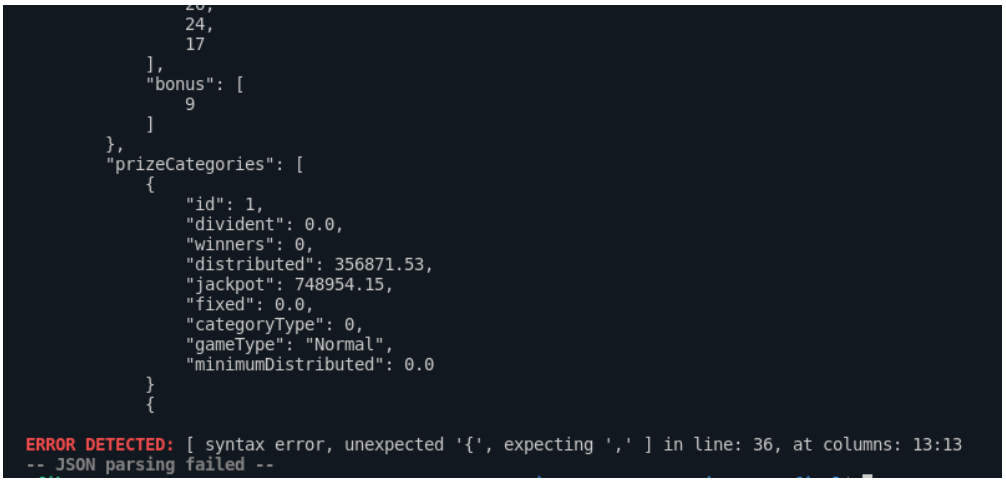
Έλλειψη αγκύλης:

```
{
    "last": {
        "gameId": 5104,
        "drawId": 2390,
        "drawTime": 164236320000000000000000000000,
        "status": "results",
        "drawBreak": 1800000,
        "visualDraw": 2390,
        "pricePoints":
            "amount": 0.5
        },
```

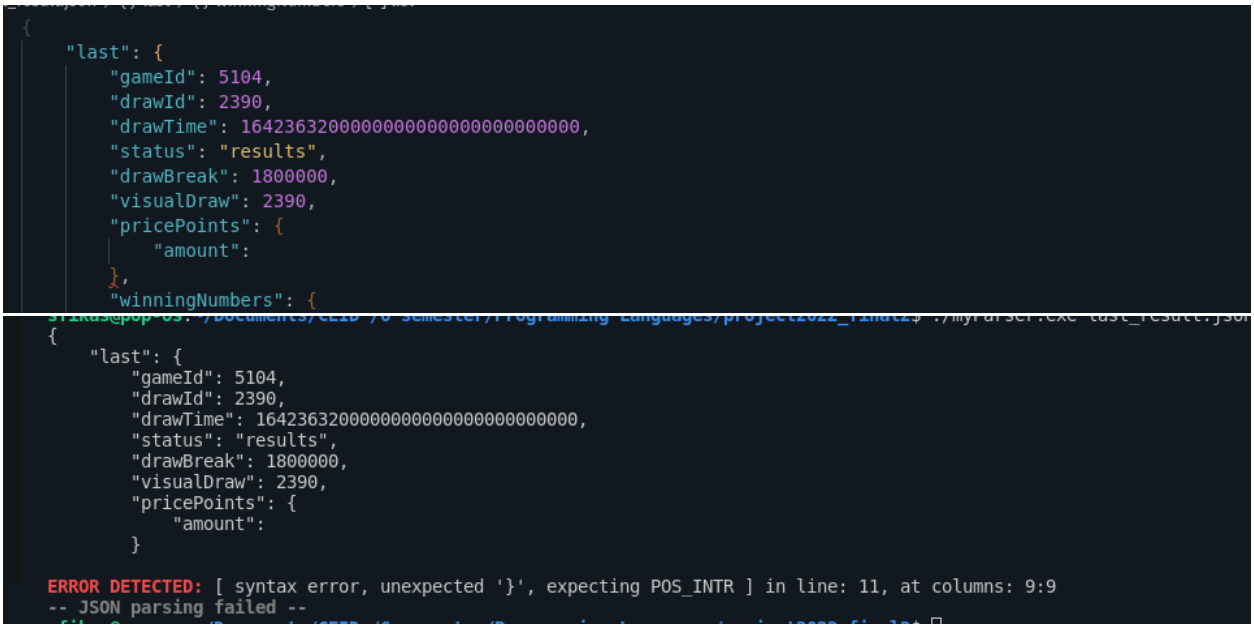
```
sfikas@pop-os:~/Documents/CEID /6 semester/Programming Languages/project2022_final2$ ./myParser.exe last_result.json
{
    "last": {
        "gameId": 5104,
        "drawId": 2390,
        "drawTime": 164236320000000000000000000000,
        "status": "results",
        "drawBreak": 1800000,
        "visualDraw": 2390,
        "pricePoints":
            "amount"
             ^^^^^^^
ERROR DETECTED: [ syntax error, unexpected amount, expecting '{' ] in line: 10, at columns: 13:20
-- JSON parsing failed --
```

Έλλειψη του χαρακτήρα ‘,’:

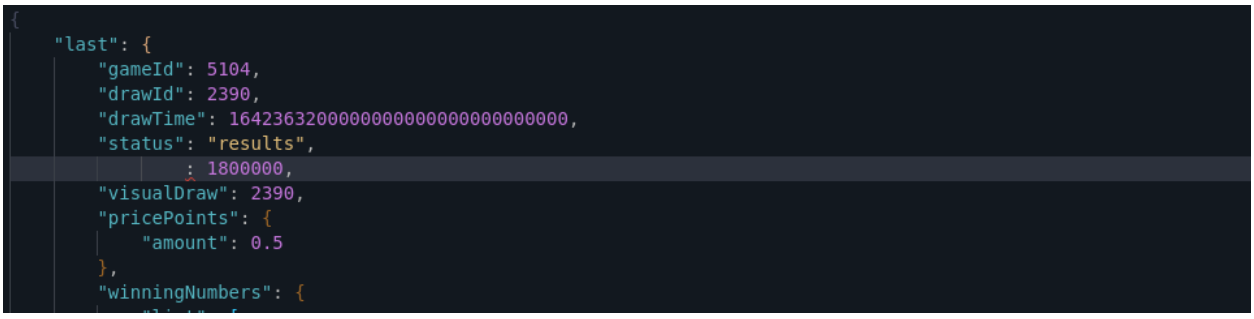
```
    "prizeCategories": [
        {
            "id": 1,
            "divident": 0.0,
            "winners": 0,
            "distributed": 356871.53,
            "jackpot": 748954.15,
            "fixed": 0.0,
            "categoryType": 0,
            "gameType": "Normal",
            "minimumDistributed": 0.0
        }
        {
            "id": 2,
            "divident": 22575.97,
            "winners": 4,
            "distributed": 55178.93,
            "jackpot": 35124.97,
            "fixed": 0.0,
            "categoryType": 0
```

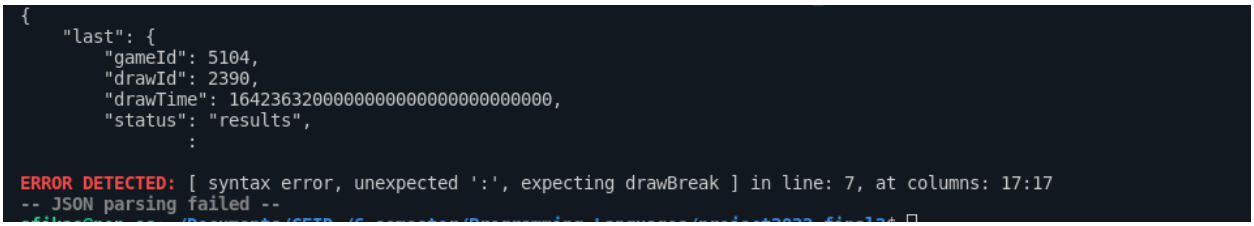


Έλλειψη τιμής:

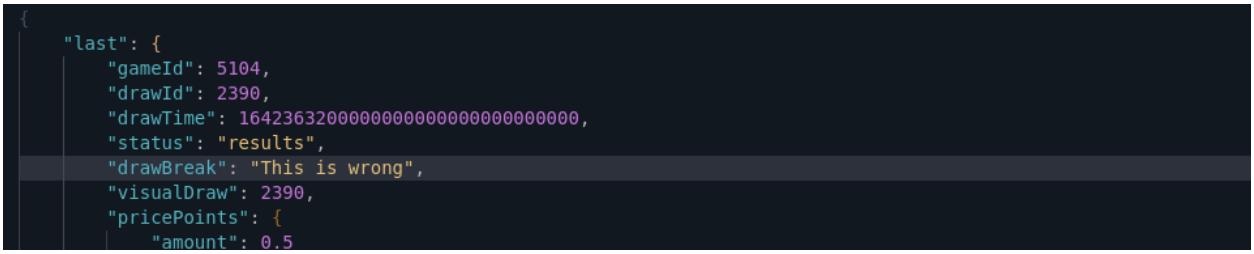


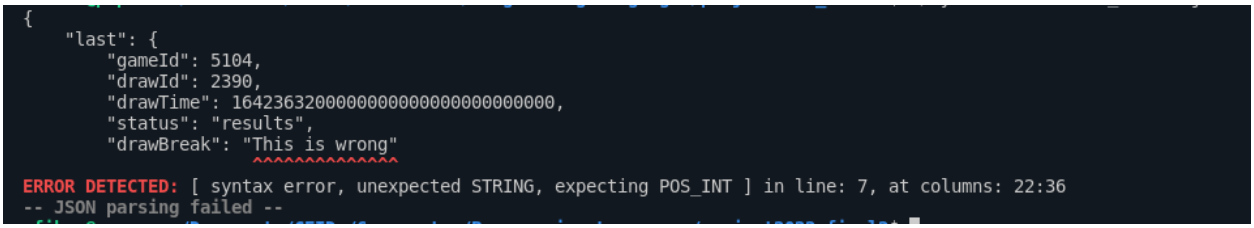
Έλλειψη πεδίου:



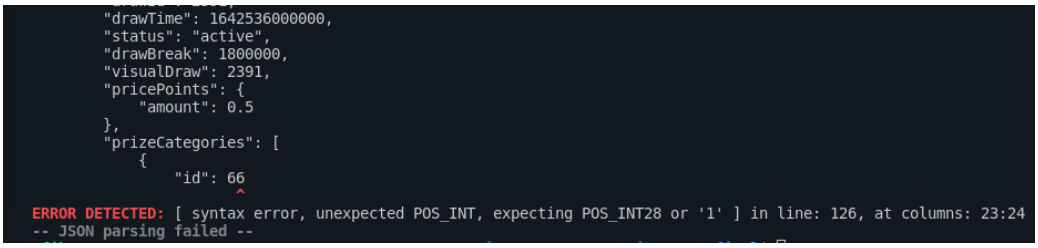


Λάθος τύπος δεδομένων:

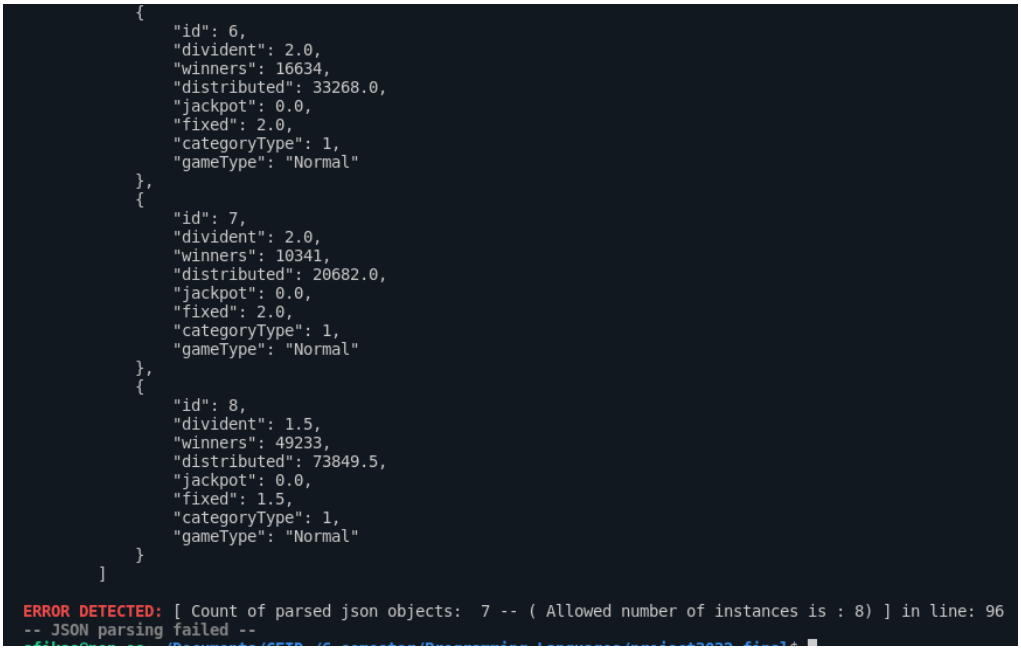


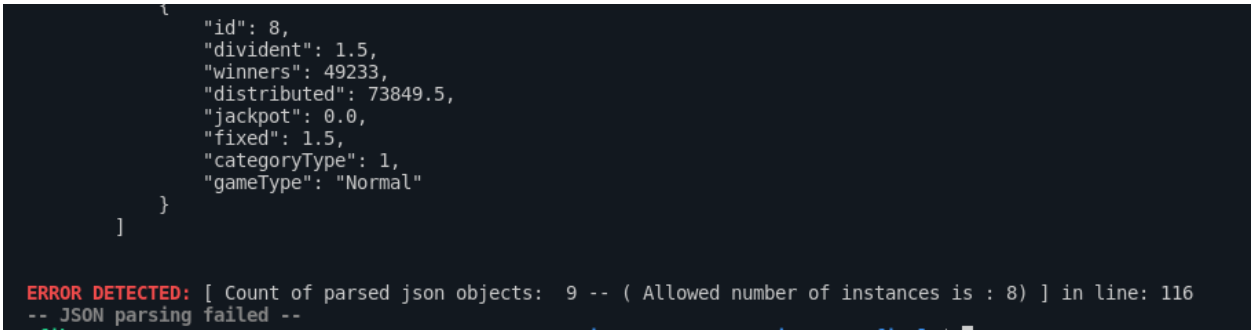


Χρησιμοποιώντας μη επιτρεπτή τιμή για το πεδίο gameId, το αποτέλεσμα που λαμβάνουμε είναι:

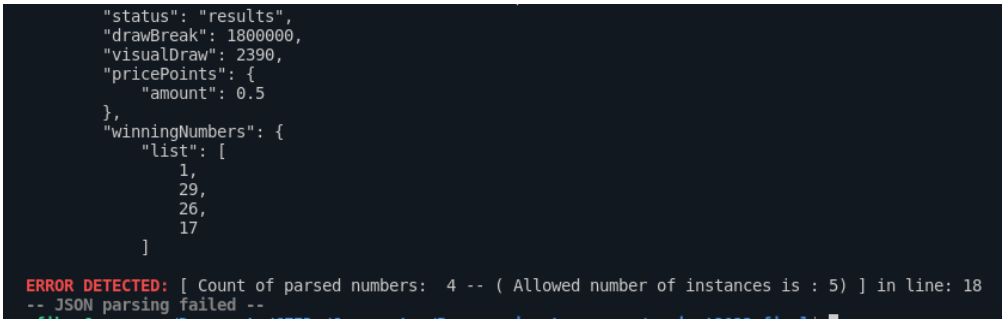


Όταν το στοιχείο “prizeCategories” δεν περιέχει ακριβώς 8 JSON αντικείμενα (η πρώτη εικόνα δείχνει το αποτέλεσμα για 7 JSON objects και η επόμενη για 9), η απάντηση από τον myParser φαίνεται στην επόμενη εικόνα:

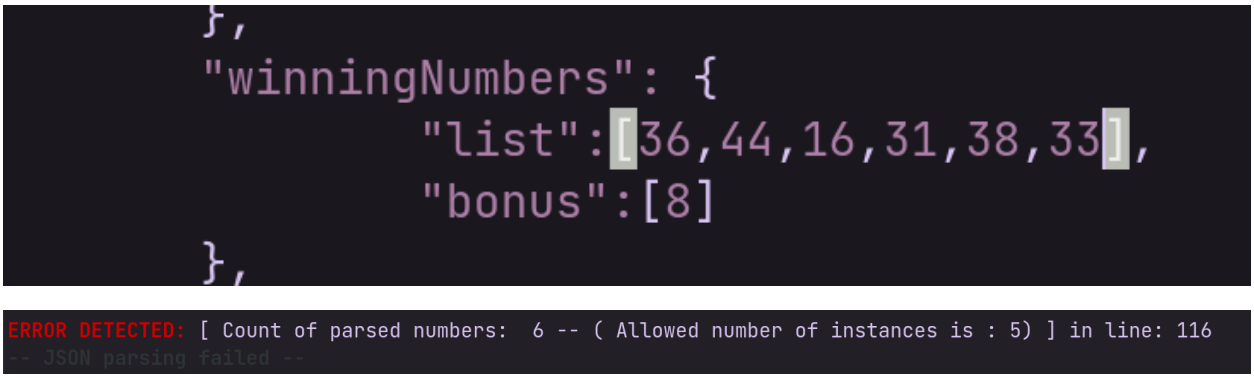




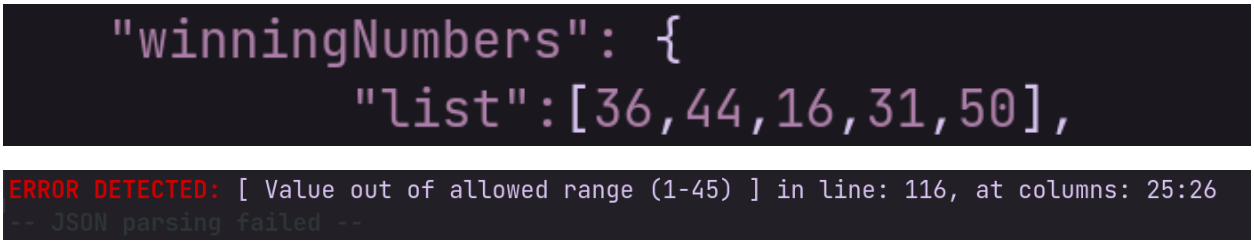
Όταν το πίνακας list περιέχει λιγότερους από 5 αριθμούς στο επιτρεπτό διάστημα, το αποτέλεσμα που λαμβάνουμε από τον myParser, φαίνεται στην 2η εικόνα:



Παρόμοια, όταν ο ίδιος πίνακας περιέχει 6 αριθμούς, το μήνυμα είναι αντίστοιχο.



Τέλος, όταν το πλήθος των αριθμών στον πίνακα είναι το ζητούμενο, αλλά υπάρχουν αριθμοί εκτός του επιτρεπτού εύρους, η απάντηση του Myparser φαίνεται στην επόμενη εικόνα:



--- ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ ---