ΘΕΜΑ Α

Με ΑΜ: mppl 21081 θα έχουμε ??1?? αντί για ??(?? ??8?? αντί για ??)?? . Το Ντετερμινιστικό Αυτόματο Στοίβας έχει εκτελεστεί σε γλώσσα python και βρίσκεται στο αρχείο main.py , ο κώδικάς του όπιου φαίνεται και στην παρακάτω εικόνα :

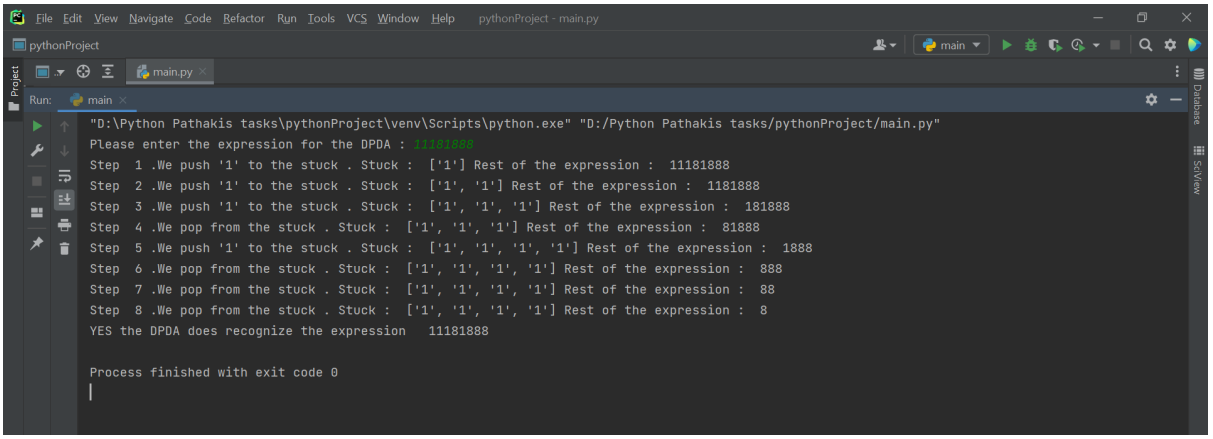
```python
# deterministic pushdown automaton DPDA in Python
def main():
    status = False
    myStack = []
    while status == False:
        status = True
        string = input("Please enter the expression for the DPDA : ")
        for i in range(len(string)):
            if string[i] != "1" and string[i] != "8":
                status = False
                print("The input hase to be exlusivly '1' or '8'")
                break
    count = 0
    for i in range(len(string)):
        if string[i] == "1":
            myStack.append(string[i])
            print("Step ",i+1,".We push '1' to the stuck . Stuck : ",myStack,"Rest of the expression : ", string[i:])
            count = count + 1
        elif string[i] == "8" and count > 0:
            myStack.pop
            print("Step ",i+1,".We pop from the stuck . Stuck : ",myStack,"Rest of the expression : ", string[i:])
            count = count - 1
        else:
            print("NO the DPDA does not recognize the expression  ", string)
            exit()
    if count == 0:
        print("YES the DPDA does recognize the expression  ", string)
    else:
        print("NO the DPDA does not recognize the expression  ", string)



if __name__ == '__main__':
    main()

```

Και εισάγοντας το “11181888” (που αντιστοιχεί στο “((()()))”) παίρνουμε :

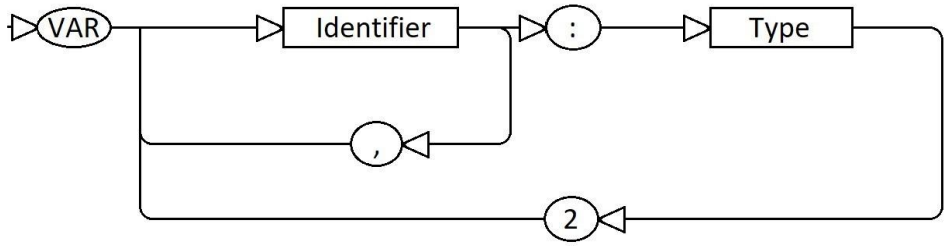


ΘΕΜΑ Β

Με ΑΜ : mppl 21081 έχουμε X = 1, y = 8, Z = 2 . Άρα ένα παράδειγμα τμήματος δηλώσεων μεταβλητών που είναι έγκυρο για την γλώσσα που έχει περιγραφεί είναι :

var num, sum , count : interger182
    tichet_price : real182
    fact_check : boolean182
    symbols : char182

Για το τμήμα δηλώσεων μεταβλητών που είναι έγκυρο για την γλώσσα που έχει περιγραφεί Το συντακτικό διάγραμμα είναι :



Οπού Type μπορεί να είναι τα integer18 ,real18, boolean18, char18, Και identifier είναι το όνομά της μεταβλητής

Η περιγραφή EBNF είναι :

Variable-declaration-part =
    var variable-declaration"2”{var variable-declaration"2”}
var variable-declaration =
    identifier-list":”type”18”
identifier-list =
    identifier{“,” identifier}
identifier =
    letter{letter|digit}

Οπου :

Type =
    integer|real|boolean|char

Και :

Digit =
    0|1|2|3|4|5|6|7|8|9
Letter =
    a|b|c|d|e|f|g|h|I|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Lex Προγραμμα :

```
Letter  [A-Za-z]
Digit [0-9]
Identifier {Letter}{Letter|Digit}*

%%

“integer18”          {return INTEGER18;}
“real18”             {return REAL18;}
“boolean18”          {return BOOLEAN18;}
“char18”             {return CHAR18;}
[ ]                  ;
[,]                  {return “,”;}
[2]                  {return 2;}
Identifier           {return Identifier;}
\n                   {return \n;}
.                    {return 0;}
%%
```

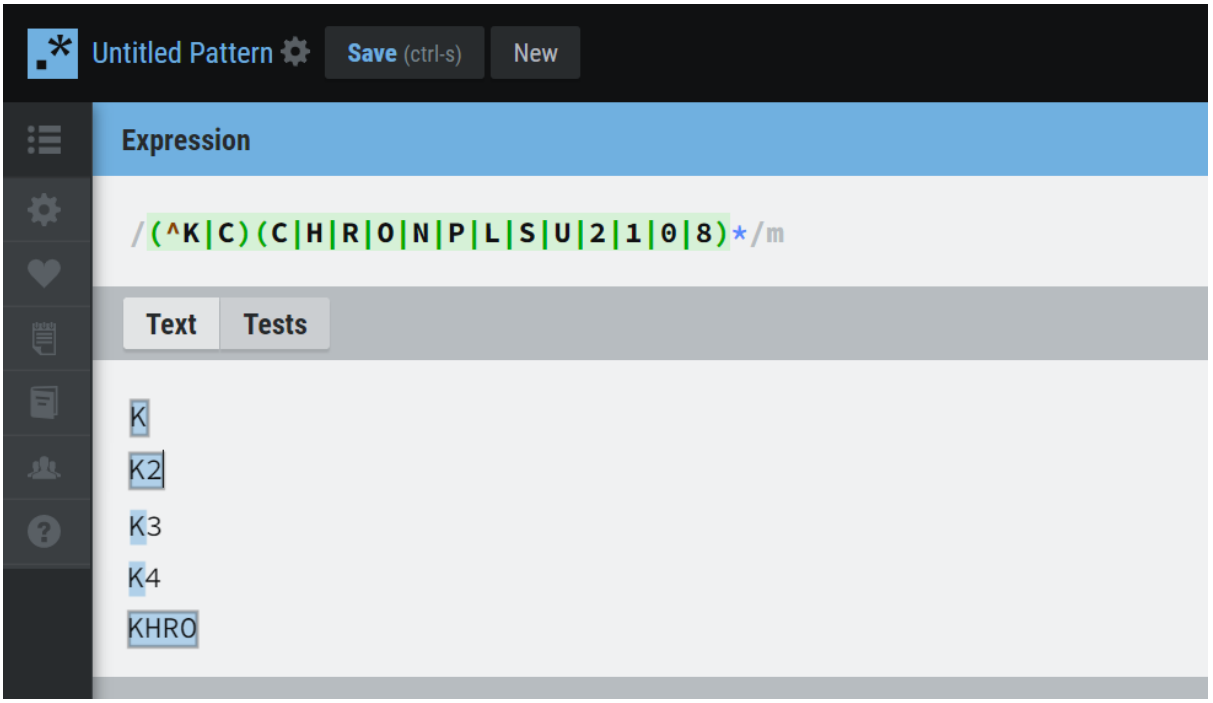
ΘΕΜΑ Γ

Με Ονοματεπώνυμο : Chronopoulos Konstantinos και AM : mppl 21081 έχουμε Στη νέα αυτή γλωσσά μια μεταβλητή δηλωνεται ως εξής:

- Το αρχικό γράμμα θα είναι είτε K η C .
- Επειτα μπορει να επαναλαμβάνονται οποιοδηποτε αριθμο φορων οι χαρακτιρες : “C” “H” “R” “O” “N” “O” “P” “U” “L” “S” και “2” “1” “0” “8” .

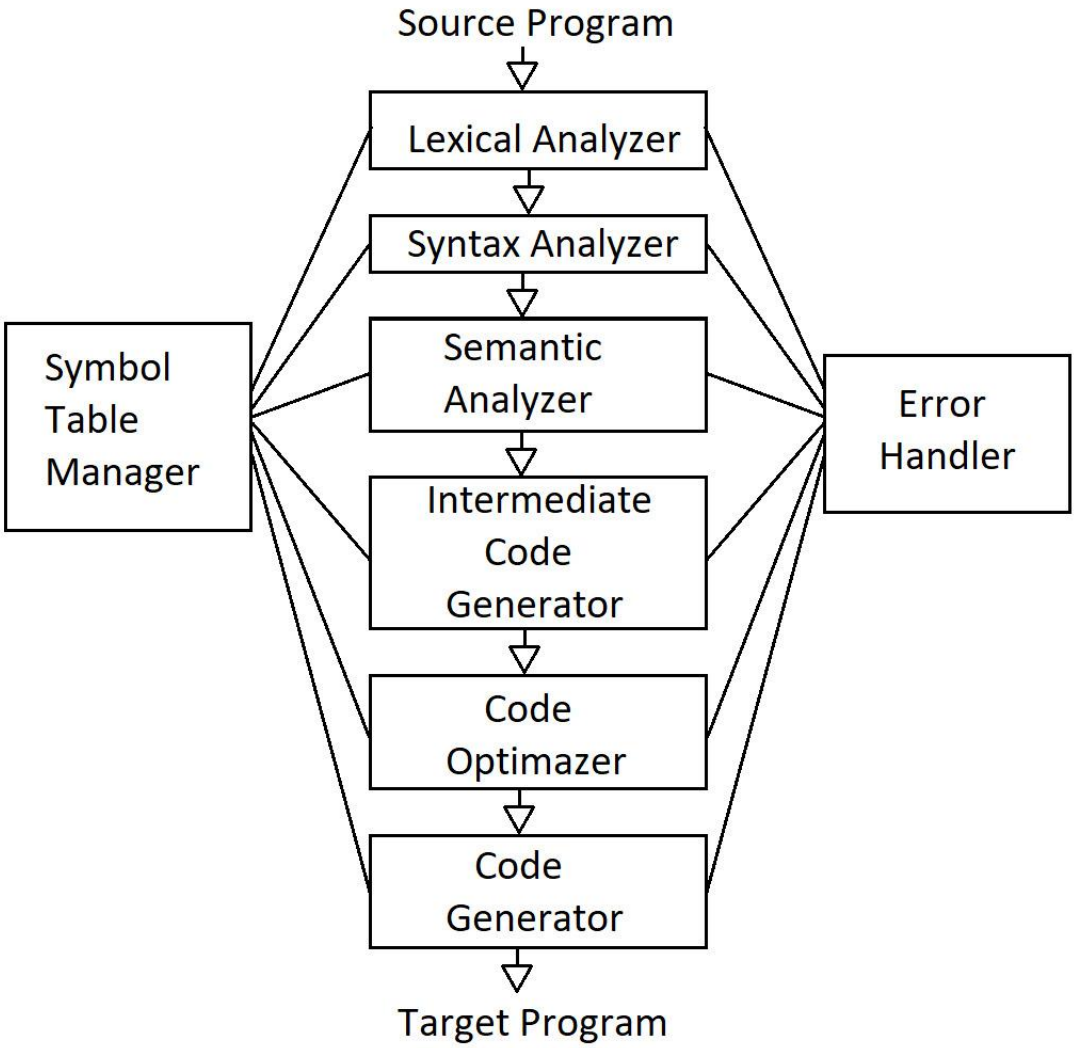
Η κανονικη εκφραση για τα ονοματα μεταβλητων θα ειναι :

/(^K|C)(C|H|R|O|N|P|L|S|U|2|1|0|8)*/



B)

Μεταγλωττιστή (compiler) ονομαζουμε ενα προγραμμα Ηλεκτρονικου Υπολογιστη που διαβαζει κωδικα γραμμενο σε μια γλωσσα προγραμματισμου (την πηγαια γλωσσα) και τον μεταφραζει σε ισοδυναμο κωδικα σε μια αλλη γλωσσα προγραμματισμου(τη γλωσσα στοχο). Συνηθως ο ορος “μεταγλωττιστης” χρησιμοποιειται κυριως για προγραμματα που μεταφραζουν μια γλωσσα προγραμματισμου υψηλου επιπεδου σε μια γλωσσα σε μια γλωσσα χαμηλου επιπεδου.
Στο παρακάτω διάγραμμα απεικονίζονται οι φασεις που περναει ο μεταγλωττιστής



Για την νεα γλωσσα προγραμματισμου μας δίνεται ο πηγαίος κώδικας :

```
int K = 2;
int K1 = 3;
int K2 = 4;
int K3 = 5;
int K4 = 4;
int K5 = 5;
int K6 = 4;
int K7 = 6;
int K8 = 7;
int K9 = 8;
if K==K1 {
        printf("K is equal to K1");
}
 iff A > A1 {
        printf("K is greater than K1");
}
K1=K4;
K=K3;
K5=K;
K6=K1;
K7=K8;
K9=5;
```

Λεκτικος Αναλυτης (lexical analyzer):

Ο Λεκτικος Αναλυτης ειναι η πρωτη φαση του μεταγλωττιστη . Ο Λεξικος Αναλυτης παιρνει ως εισοδο τον πηγεο κοδικα σαν συμβολοσειρα χαρακτηρων του προγραμματος και τον μετατρέπει σε μια συμβολοσειρα απο tokens(λεκτικών μονάδων) και αφαιρει τα επιπλεον κενα και σχολια .Τα tokens αυτα συνηθως μπορει να ειναι 1. Λεξεις κλειδια (keywords) 2. Τα αναγνωριστικα (identifiers) 3. Σταθερές (constants) 4. Οι τελεστες (operators).Μια σειρα χαρακτηρων η οποια δεν αναγνωριζεται ως σωστο token ειναι ενα λεξιλογικο λαθος πχ ο Λεκτικος αναλυτης (lexical analyzer) το “if” ως keyword αλλα δεν θα αναγνωρίσει το “iff”.
Το αποτελεσμα λοιπον του Λεκτικου αναλυτη θα ειναι :

N	Lexeme	Type	N	Lexeme	Type	N	Lexeme	Type
1	int	keywords	22	K4	error	43	=	operator
2	K	identifier	23	=	operator	44	7	identifier
3	=	operator	24	4	identifier	45	;	special
4	2	identifier	25	;	special	46	int	keywords
5	;	special	26	int	keywords	47	K9	error
6	int	keywords	27	K5	error	48	=	operator
7	K1	identifier	28	=	operator	49	8	identifier
8	=	operator	29	5	identifier	50	;	special
9	3	identifier	30	;	special	51	if	keywords
10	;	special	31	int	keywords	52	K	identifier
12	int	keywords	32	K6	error	53	==	operator
12	K2	identifier	33	=	operator	54	K1	identifier
13	=	operator	34	4	identifier	55	{	operator
14	4	identifier	35	;	special	56	printf	identifier
15	;	special	36	int	keywords	57	(	operator
16	int	keywords	37	K7	error	58	“K is equal to K1”	identifier
17	K3	error	38	=	operator	59	)	operator
18	=	operator	39	6	identifier	60	;	special
19	5	identifier	40	;	special	61	}	operator
20	;	special	41	int	identifier	62	iff	error
21	int	keywords	42	K8	identifier	63	A	error

N	Lexeme	Type	N	Lexeme	Type	N	Lexeme	Type
64	>	operator	75	K4	error	86	=	operator
65	A1	error	76	;	special	87	K1	identifier
66	{	operator	77	K	identifier	88	;	special
67	printf	identifier	78	=	operator	89	K7	error
68	(	operator	79	K3	error	90	=	operator
69	“Kis greater than K1”	identifier	80	;	special	91	K8	identifier
70	)	operator	81	K5	error	92	;	special
71	;	special	82	=	operator	93	K9	error
72	}	operator	83	K	identifier	94	=	operator
73	K1	identifier	84	;	special	95	5	identifier
74	=	operator	85	K6	error	96	;	special

Αρα ο λεκτικος αναλυτης θα αναγνωρισει τα K,K1,K2,K8 ως identifier αλλα τα K3,K4,K5, K6,K7,K9 και A , A1 δεν θα τα αναγνωρίζει ως identifier γιατι δεν τηρουν τις συνθηκες της γλωσσας.
Επισεις δεν θα αναγνωρίσει το “iff’ ως keyword .

Συντακτικος Αναλυτης(syntax analyzer):

Ο Συντακτικος Αναλυτης ειναι η δευτερη φαση του μεταγλωττιστη ,θα παρει ως εισοδο το παραπονο σειρα απο tokens(token streams) και τα αναλυει με βαση μια σειρα απο κανονες παραγωγης για να αναγνορισει λαθοι στον κωδικα .Ως εξοδο εχει ενα συντακτικό δένδρο (parse tree).
Ο Λεκτικος Αναλυτης αναγνωρίζει τα tokens με την βοηθεια κανονικων εκφρασεων (regular expressions) και κανονων , ο Λεκτικος Αναλυτης ομως δεν μπορεί να ελέγξει την συνταξη μιας προτασης λογο τον περιορισμων των κανονικων εκφρασεων πχ οι κανονικες εκφρασεις δεν μπορουν να αναγνωρισουν παρενθέσεις που ανοιγουν και κλεινουν ,Για Αυτο ο Συντακτικος αναλυτης χρησιμοποιεί context-free grammar που αναγνωριζεται απο τα push-down automata.

Σημασιολογικος Αναλυτης(Semantic Analyzer)

Ο Σημασιολογικος Αναλυτης(Semantic Analyzer) ειναι η τριτη φαση του μεταγλωττιστη .ο σημασιολογικής ανάλυσης βεβαιωνει οτι οι δηλώσεις και εκφρασεις του προγραμματος ειναι σημασιολογικα σοστες .Ο Σημασιολογικος Αναλυτης(Semantic

Analyzer) παιρνει ως είσοδο το συντακτικο δεντρο απο την προηγουμενη φαση αλλα χρησιμοποιεί και τον πινακα συμβολων (symbol table) για να δοκιμασει αν το προγραμα είναι Σημασιολογικα σωστο ,

Σημασιολογικα Λάθοι :
Παραδείγματα σημασιολογικων λαθων :

- Λαθος τυπος μεταβλητης.
- Μη δηλωμενες μεταβλητες.
- Λαθος χρηση δεσμευμενων identifier.