Θέματα Ιουνίου 2022

Θέμα 1 - Ιούνιος 2022 (4 μονάδες)
είχε 5 υποερωτήματα και έλεγε να κάνετε bottom up για το aabbaa στη γραμματική με
<S>::=a<A>|b<B>
<A>::=a<S>|a
<B>::=b<S>|b
να πείτε αν είναι LL(1) και να τη μετατρέψετε
να πείτε αν είναι κανονική γλώσσα και να δώσετε το regex
να σχεδιάσετε FSA
και να δώσετε παραδείγματα των συμβολοσειρών που περιγράφει καθώς και τη μικρότερη

Θέμα 2 - Ιούνιος 2022 (2 μονάδες)
Βρείτε τα ΤΠΑ, ΜΗ-ΤΠΑ, Καθολικό για ένα πρόγραμμα Pascal
και μετά βρείτε τι τυπώνει για a) call by value b) call by reference c) call by name

Το τριτο θεμα ηταν το c = a > ++b && b++ && --a και σου ζηταγε να βαλεις παρενθεσεις στα σωστα σημεια. Σου εδινε δυο φορες τιμες για a και b και ζηταγε την τελικη τιμη των a,b,c. Αυτο ηταν το κομματι της C.

Το τριτο ελεγε
c = a > ++b && b++ && --a

η Python, ΘΕΜΑ 4 ,ελεγε αυτο :
1) Συμπληρώστε τον παρακάτω κώδικα ώστε να έχει output το πρόγραμμα : [1 2 3 4]
```
def f(n):
    for x in n:
        .....
f([1,2,3,4])
```

και η απάντηση νομίζω ήταν print(x)
2) Τι επιστρέφει ο παρακάτω κωδικας :
```
for i in [3,4]:
    for j in [1,4]:
        if i < j : break;
        print(i-1 , end="")
    print(i-1 , end="")
print(i-1 , end="")
```

και η απάντηση ήταν
233
3)Τι πρέπει να συμπληρώσετε στον παρακάτω κωδικα python για να επιστραφεί [5,6,12,13]
print(x for x in range(15) ....)
και η απάντηση που έβαλα εγώ ήταν
if x in [5,6,12,13]
μπορει να εχω λαθος τις απαντήσεις

5ο θεμα ηταν συναρτησιακος
α) Ειχε ενα τμημα κωδικα με επαναληψη
```
R = 0; i = 0; j = 1000;
while i < j {
  R = R + j^i;
  j = j - 1; i = i + 1;
}
```
και το ηθελε σε recursive
δεν θυμαμαι αν ηθελε συγκεκριμενα σε functional να ειναι ή απλα μετατροπη σε recursive. Ηθελε το σωμα της συναρτησης και την πρωτη κληση παντως
μετα ειχε καποια statements σε functional γλωσσες και σου ελεγε να πεις σε τι κανουν evaluate. Ηταν σχετικα απλα, ήταν ενα map, ενα if, ενα cond (san switch). Και μετα ειχε δυο fold (ενα αθροισμα ενα γινομενο), ενα map και ενα define και σου ελεγε σε τι αποτιμαται οταν το καλουσε σε λιστα (1,2,3,4,5)
θυμαμαι τελευταιο ερωτημα ηταν να γραψεις την σειρα που γινονταν οι αναγωγες/αντικαταστασεις στην λ εκφραση
(λx.λy.λz x + y + z) 3 4 5

Η λυση που εκανα εγω για το θεμα 5 υποερ 1 ηταν η
```
Main :
   R= 0 ; i = 0 ; j = 1000;
   Def F(R,i,j) :
      R = R + j^i;
      j = j - 1;
      i = i + 1;
      if i < j :
         R = F(R,i,j);
      end if
      return R;
   R = F(R,i,j);
   return R;
```

Δεν ξερω αν ειναι σωστή
Mia Wallace — 06/20/2022
στο 3ο η απάντηση που ήθελε νομίζω είναι [x for x in range(15) if (x%7>4)] αυτή
∫∫((Nektarios)dx)dy — 06/20/2022
εχεις δικιο , σιγουρα αυτήν ήθελε
η δικια μου ειναι τσατσικη , δεν ξερω αν θα την πάρει για σωστή
Frozo — 06/21/2022
Και γω αυτή που ειπες εκανα, δεν βλεπω κανένα απολύτως λαθος
nocomm — 06/21/2022
ειχε κ ενα προγραμμα σε c να εξηγησεις τι κανει γραμμη γραμμη στο θεμα με τα ΤΠΑ
μη-ΤΠΑ ΚΠΑ
Poh — 06/21/2022
Δεν είμαι 100% αλλά νομίζω το 3b ήταν αυτό:
```
int main {
  int data[6] = {1, 2, 3, 4, 5, 6};
```

```
  int *p, *q;
  int a, b, c;
  p = data;
  q = p + 2;
  a = *p + *q;
  b = q[2] + data[2];
  *(q + 1) = 0;
  ++p; ++q;
  c = *p - *q + data[3];
  cout << a << " " << b << " " << c << endl;
  return 0;
}
```

και ζήταγε εξήγηση γραμμή γραμμή και τι τυπώνει το πρόγραμμα

Θέματα Σεπτεμβρίου 2021

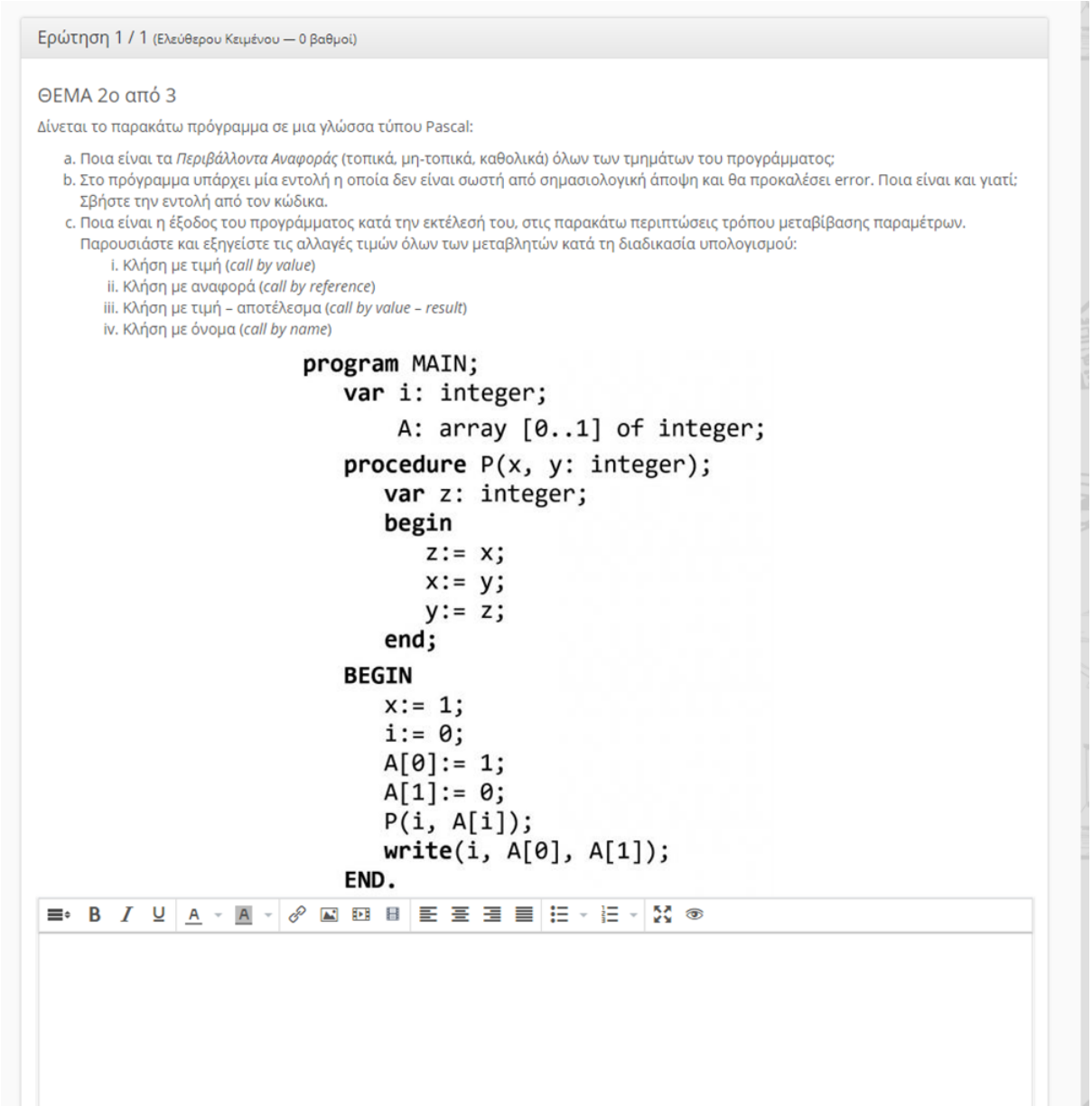
Ερώτηση 1 / 1 (Ελεύθερου Κειμένου — 0 βαθμοί)

ΘΕΜΑ 1ο από 3

Σας δίνονται οι παρακάτω τρεις (3) απλές γραμματικές σε συμβολισμό **BNF**.

a. Να περιγράψετε τις ακολουθίες δυαδικών αριθμών που παράγουν οι τρεις γραμματικές. Δώστε και παραδείγματα.
b. Αντιστοιχίστε τις ακολουθίες **0110101**, **110110**, **0100010**, **110110110**, **11111** στην/στις γραμματική/ές που τις παράγουν, ή σε καμία αν δεν μπορούν να παραχθούν.
c. Δείξτε αν η ακολουθία **10101** παράγεται ή όχι από την 3η γραμματική, κατασκευάζοντας τη *Στοίβα Ολίσθησης – Ελάττωσης Bottom-Up* συντακτικής ανάλυσης.
d. Ποια ή ποιες από τις γραμματικές είναι *κανονική* και γιατί;
e. Για την πρώτη κατά σειρά (με βάση με την παρακάτω αρίθμησή τους) από την/τις κανονικές γραμματικές που αναγνωρίσατε στο ερώτημα (d), παρουσιάστε αντίστοιχη *κανονική έκφραση*, καθώς και τον *Πίνακα Καταστάσεων – Μεταβάσεων* (εναλλακτικός τρόπος παρουσίασης του διαγράμματος καταστάσεων – μεταβάσεων για το αντίστοιχο ντετερμινιστικό πεπερασμένο αυτόματο).
f. Ποια ή ποιες από τις τρεις γραμματικές δεν είναι LL(1) και γιατί; Μετασχηματίστε την πρώτη κατά σειρά (με βάση με την παρακάτω αρίθμησή τους) από τις γραμματικές αυτές, σε ισοδύναμη LL(1) γραμματική. Τι παρατηρείτε στη νέα LL(1) γραμματική, όσον αφορά τις ιδιότητές της;

1. <A> ::= **0**<A> | **1**<A> | **0**
2. <B> ::= <B><B> | **0** | **1**
3. <C> ::= **0**<C>**0** | **1**<C>**1** | **0** | **1**



Ερώτηση 1 / 1 (Ελεύθερου Κειμένου — 0 βαθμοί)

ΘΕΜΑ 2ο από 3

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

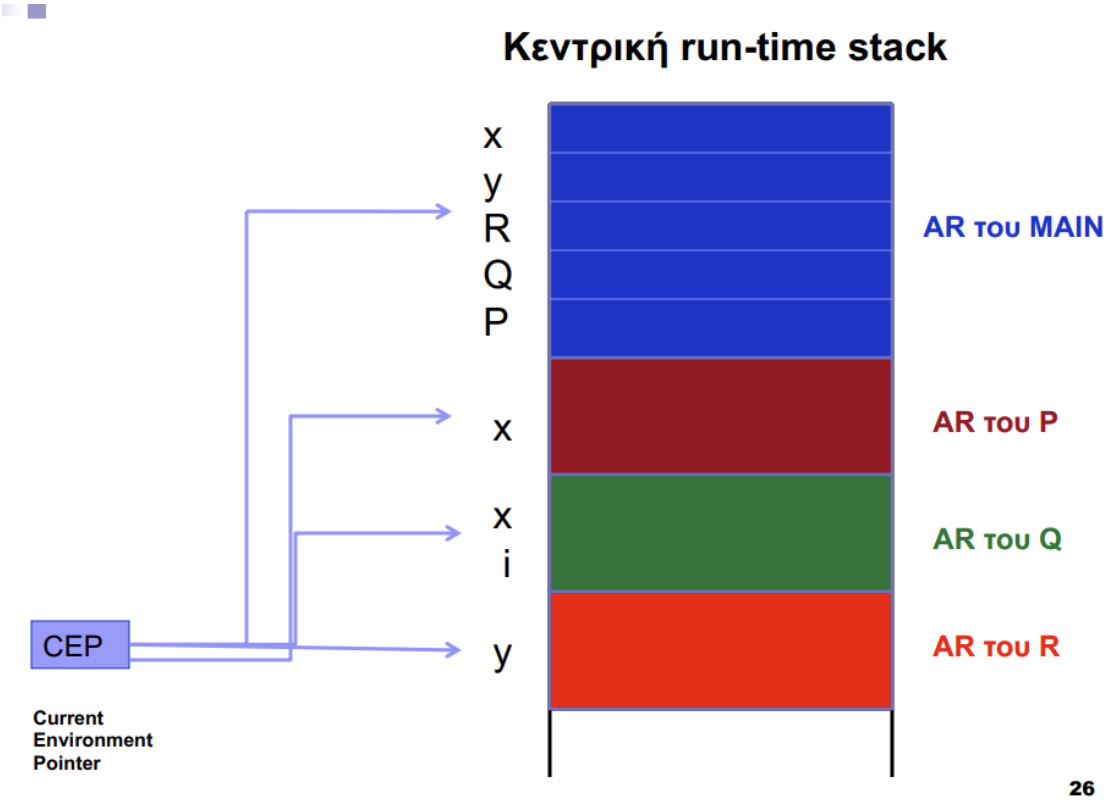
a. Ποια είναι τα *Περιβάλλοντα Αναφοράς* (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
b. Στο πρόγραμμα υπάρχει μία εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει error. Ποια είναι και γιατί; Σβήστε την εντολή από τον κώδικα.
c. Ποια είναι η έξοδος του προγράμματος κατά την εκτέλεσή του, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων. Παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού:
   i. Κλήση με τιμή (*call by value*)
   ii. Κλήση με αναφορά (*call by reference*)
   iii. Κλήση με τιμή – αποτέλεσμα (*call by value – result*)
   iv. Κλήση με όνομα (*call by name*)

```
program MAIN;
   var i: integer;
       A: array [0..1] of integer;
   procedure P(x, y: integer);
      var z: integer;
      begin
         z:= x;
         x:= y;
         y:= z;
      end;
   BEGIN
      x:= 1;
      i:= 0;
      A[0]:= 1;
      A[1]:= 0;
      P(i, A[i]);
      write(i, A[0], A[1]);
   END.
```

Σημειώσεις και εννοιες:

1. **Τοπικό Περιβάλλον Αναφοράς (ΤΠΑ)** ενός υπ/τος
   a. Το σύνολο των συνδέσεων ονομάτων με οντότητες που δημιουργούνται με την είσοδο σε ένα υποπρόγραμμα. Αντιπροσωπεύουν:
      i. Τοπικές μεταβλητές
      ii. Τυπικές παραμέτρους
      iii. Τοπικά υποπρογράμματα
2. **Μη-Τοπικό Περιβάλλον Αναφοράς (Μη-ΤΠΑ)** υπ/τος
   a. Το σύνολο των συνδέσεων ονομάτων με οντότητες που μπορούν να χρησιμοποιηθούν από το υπ/μα, αλλά **δεν δημιουργούνται με την είσοδο σε αυτό.**
3. **Καθολικό Περιβάλλον Αναφοράς (ΚΠΑ)** ενός υπ/τος

    a. **Το σύνολο των συνδέσεων ονομάτων με οντότητες που δημιουργούνται με την έναρξη του main και είναι διαθέσιμες στο υπ/μα.** Είναι μέρος του Μη-ΤΠΑ.
4. Προκαθορισμένο Περιβάλλον Αναφοράς (ΠΠΑ)
    a. Το σύνολο των προκαθορισμένων συνδέσεων από τη ΓΠ που είναι διαθέσιμες σε όλο το πρόγραμμα. Π.χ. MAXINT, read, write, …
5. Δυναμική Εμβέλεια (dynamic scope)
    a. μιας σύνδεσης, είναι το τμήμα εκτέλεσης του προγράμματος κατά το οποίο η σύνδεση υπάρχει ως μέρος κάποιου ΠΑ. Δηλαδή, αποτελείται από το σύνολο των ενεργοποιήσεων υπ/των στα οποία είναι ορατή η σύνδεση
6. Κανόνες Εμβέλειας σε Block-Structured Προγράμματα
    a. Οι δηλώσεις στην αρχή του block ορίζουν το ΤΠΑ του block. Κάθε αναφορά σε όνομα μέσα στο block (εκτός από τα εσωτερικά block), θεωρείται αναφορά στην τοπική δήλωση του ονόματος (αν υπάρχει).
    b. Αν γίνεται αναφορά σε όνομα στο block και δεν υπάρχει τοπική δήλωση (δηλαδή η σύνδεση δεν είναι μέρος του ΤΠΑ), τότε η δήλωση αναζητείται στο αμέσως πιο εξωτερικό block (δηλαδή στο Μη-ΤΠΑ) κ.ο.κ. έως το ΚΠΑ. Αν δεν βρεθεί εκεί, εξετάζεται το ΠΠΑ. Αν δεν βρεθεί και εκεί: ERROR.
    c. Αν ένα block περιλαμβάνει άλλο block, κάθε δήλωση στο εσωτερικό block (και τα εσωτερικά αυτού), είναι εντελώς κρυμμένη από το εξωτερικό block και δεν μπορεί να προσπελασθεί από αυτό.
    d. Αν ένα block έχει όνομα (στη C μπορεί και να μην έχει), το όνομα αυτό είναι μέρος του ΤΠΑ του block στο οποίο περιλαμβάνεται.
        i. Π.χ. αν στο main ενός προγράμματος Pascal περιλαμβάνεται ο ορισμός: procedure P (A: real); …… τότε το όνομα P είναι μέρος του ΤΠΑ του main, ενώ το A είναι μέρος του ΤΠΑ του P.
    e. Αν το block επιστρέφει τιμή (δηλαδή είναι function, όχι void), με την έναρξη εκτέλεσης του block, δημιουργείται και τοπική μεταβλητή με το όνομά του (δηλαδή, μέρος του ΤΠΑ του block).
7. Κανόνες εμβελειας:
    a. **Στατικός Κανόνας** Εμβέλειας (static scoping rule) **Για κάθε μεταβλητή του Μη-Τοπικού ΠΑ, ψάχνει στο αμέσως πιο εξωτερικό block στο κείμενο του προγράμματος**. Αν δεν βρεθεί εκεί, ψάχνει στο αμέσως πιο εξωτερικό block κ.ο.κ.
        i. C, C++, FORTRAN, COBOL, Pascal, Ada, …
    b. **Δυναμικός Κανόνας** Εμβέλειας (dynamic scoping) **Για κάθε μεταβλητή του Μη-Τοπικού ΠΑ, ψάχνει στη διαδικασία ή block που κάλεσε την τρέχουσα.** Αν δεν βρεθεί εκεί, ψάχνει στη διαδικασία ή block που κάλεσε αυτήν κ.ο.κ. Δηλαδή, εξετάζονται οι διαδικασίες και τα blocks με την αντίθετη σειρά από τη σειρά κλήσης τους.
        i. LISP, APL, …
8. **Υλοποίηση στοιβας εκτέλεσης (run-time stack)** (6)



a. 26

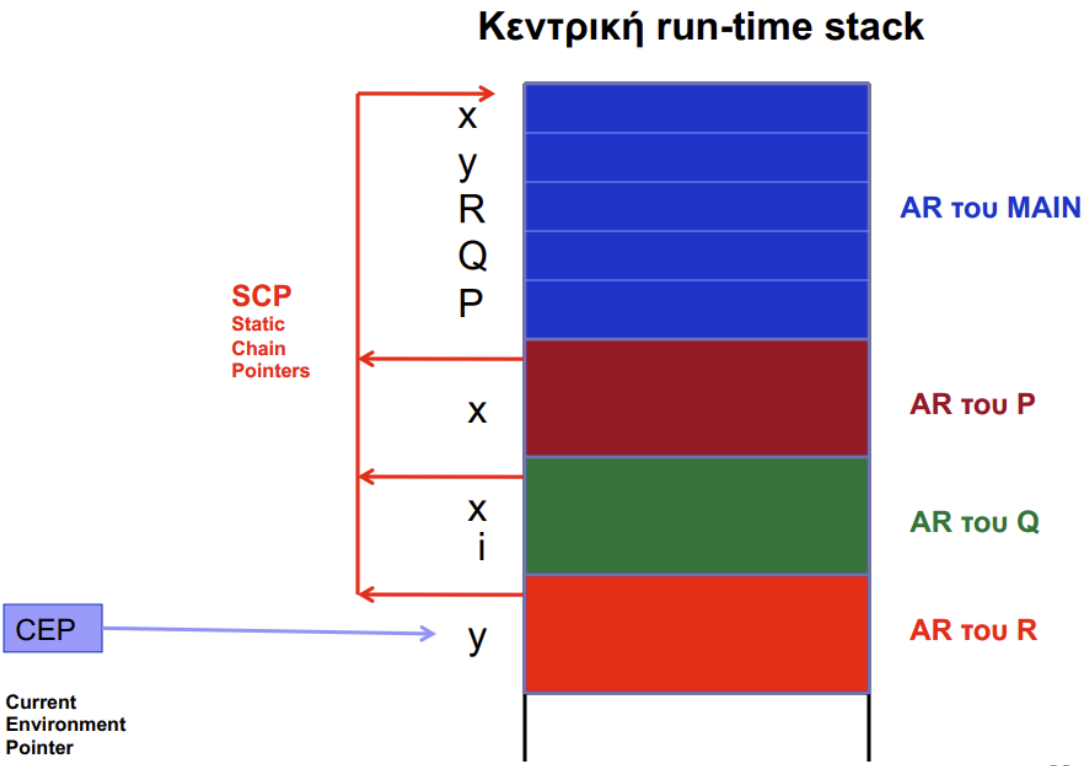
b. Κάθε **AR** δημιουργείται με την έναρξη εκτέλεσης του υποπρογράμματος, και διαγράφεται με τη λήξη της εκτέλεσής του.

c. Για την εύρεση των δηλώσεων των μεταβλητών που χρησιμοποιεί το τρέχον υπ/μα, γίνεται έλεγχος πρώτα στο AR που δείχνει ο CEP (Current Environment Pointer).

d. Στη συνέχεια, αναζητούνται οι δηλώσεις στα «από πάνω» AR. Αυτό, όμως, υλοποιεί το Δυναμικό Κανόνα Εμβέλειας.

e. **Για την υλοποίηση του Στατικού Κανόνα Εμβέλειας, απαιτείται η χρήση ειδικών δεικτών, των Static Chain Pointers (Δείκτες Στατικής Αλυσίδας) που δείχνουν στην πιο πρόσφατη ενεργοποίηση του εξωτερικού μπλοκ με βάση το κείμενο του προγράμματος.**

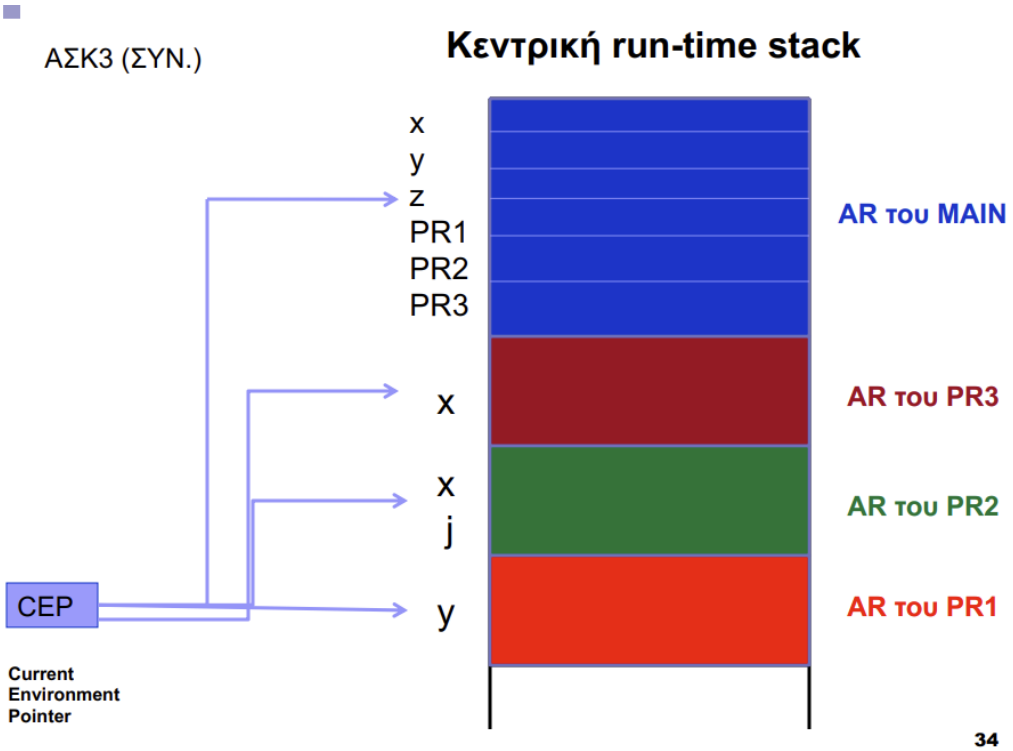
f.



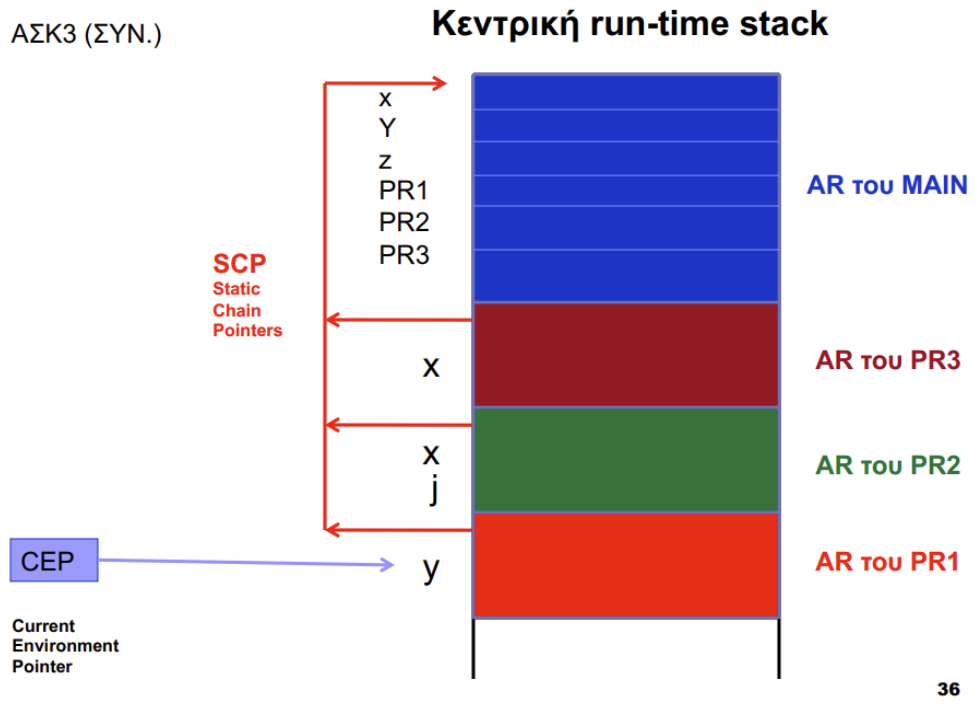
g. ΑΣΚΗΣΗ 3: Τί θα τυπώσει η main ρουτίνα, αν εφαρμόσουμε στατικό κανόνα εμβέλειας και τι αν εφαρμόσουμε δυναμικό κανόνα εμβέλειας? Σχεδιάστε το περιεχόμενο της run-time stack και του δείκτη CEP σε κάθε περίπτωση.

i.
```
Program main;
        var x,y,z: integer;
procedure PR1;
        var y: real;
begin
        x:= x + 1;
writeln(x);
end;
procedure PR2;
        var x: real; j: integer;
begin
        x:= 0.0;
PR1;
end;
procedure PR3;
        var x: boolean;
begin
PR2;
end;
BEGIN
        x:= 100;
PR3;
writeln(x);
END.
```

ii. Ερώτημα: Όταν εκτελείται η PR1, ποιο x τυπώνεται;
    1. Απάντηση: ΣΚΕ: x = 101 (από main, x=100) ΔΚΕ: x = 1.0 (από PR2, x=0.0)



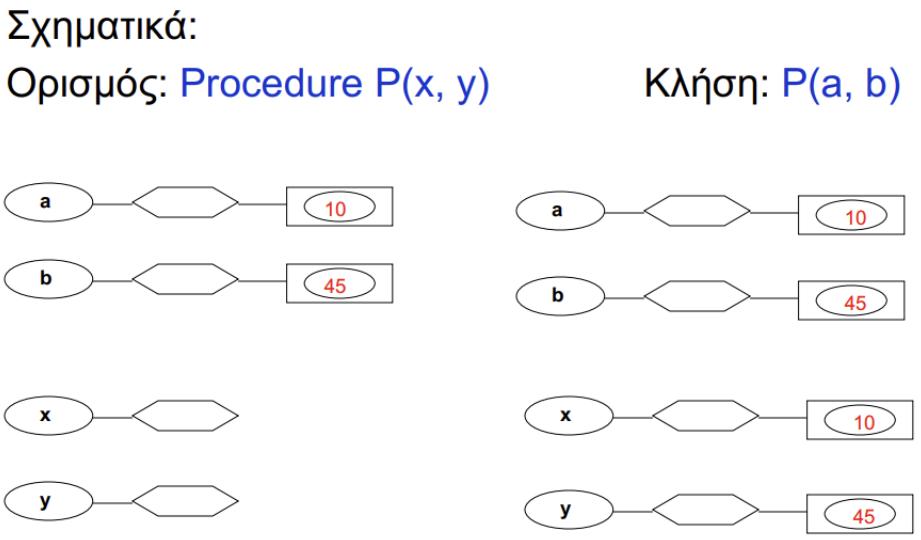
iii.



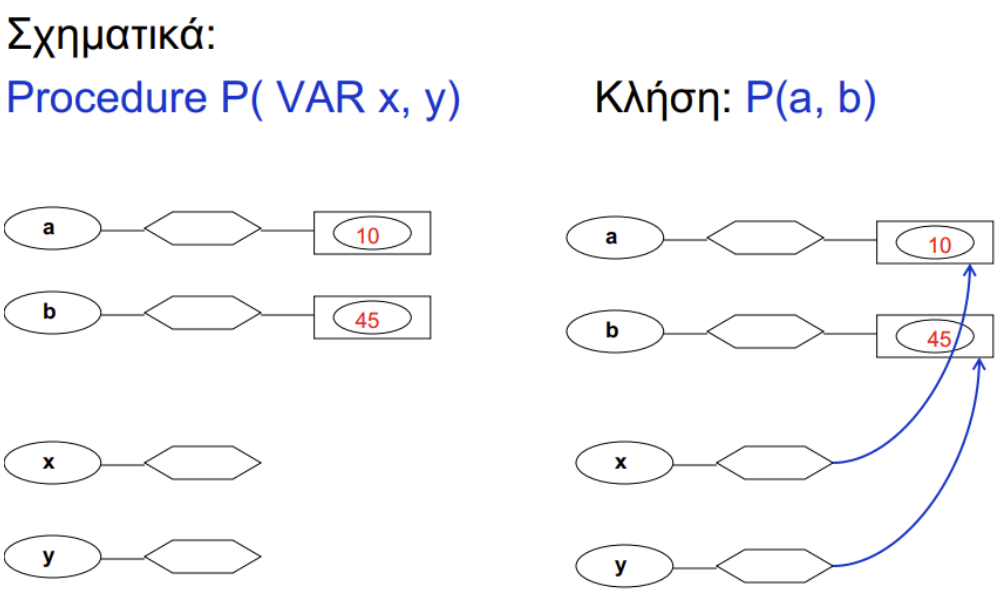
iv.

9. Μεταβίβαση Παραμέτρων (parameter passing) (7)
    a. Κλήση με Τιμή (call by value)
        i. Η πραγματική παράμετρος αποαναφοροποιείται και επιστρέφει μία τιμή, η οποία αντιγράφεται σε μια νέα θέση μνήμης, με την οποία συνδέεται το όνομα της τυπικής παραμέτρου. **Δηλαδή, πρακτικά δημιουργείται μια νέα (τοπική) μεταβλητή.**

ii. C, C++, Pascal (default)
iii. Πλεονέκτημα: Το ΥΠ μόνο διαβάζει την πραγματική παράμετρο, δεν έχει πρόσβαση για να την αλλάξει.
iv. Μειονέκτημα: Διπλασιασμός χρησιμοποιούμενης μνήμης
v.



b. Κλήση με Αναφορά (call by reference)
i. Αν η πραγματική παράμετρος χρησιμοποιεί μνήμη (π.χ. είναι μεταβλητή), αυτή η μνήμη συνδέεται με την τυπική παράμετρο, όταν γίνεται κλήση του ΥΠ. **Δηλαδή, πρακτικά η τυπική παράμετρος γίνεται pointer στην πραγματική παράμετρο.**
ii. FORTRAN, PL/1, Pascal (με VAR στις τυπικές παραμέτρους)
iii. C, C++ με χρήση pointers: void A(int *f) κλήση: A(&x)
iv. Κλήση με πραγματικές παραμέτρους σταθερές: P(var x) Κλήση: P(10) Αν αλλάζει η τιμή του x στο ΥΠ, η θέση μνήμης του λέγεται ανώνυμη μεταβλητή.
v. Πλεονεκτήματα: Απόδοση, γρήγορη αλλαγή τιμής της πραγματικής παραμέτρου.
vi.



c. Κλήση με Τιμή - Αποτέλεσμα (call by value – result)

i. Όταν η πραγματική παράμετρος είναι μεταβλητή, αποαναφοροποιείται και η τιμή αντιγράφεται σε μια νέα θέση μνήμης, όπως στην Κλήση με Τιμή. Η θέση αυτή μνήμης, χρησιμοποιείται στο σώμα του ΥΠ. **Κατά την έξοδο, η τιμή της τυπικής παραμέτρου αντιγράφεται στη θέση μνήμης της πραγματικής μεταβλητής.**

ii. ALGOL-W

Παράδειγμα 6.4 (call-by-value result)

```
program params;
   var i: integer;
   a: array[1..2] of integer;
   procedure p(x,y: integer);
   begin
      x := x + 1;
      i := i + 1;
      y := y + 1;
   end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   p( a[i],a[i] );
   output( a[1],a[2] );
end.
```

Call by Value Result:
x and **y** are **initialized at the beginning** of the procedure execution **with the values of the actual parameters**, and, **at the end** of the execution of the procedure, **are copied back to the *original* variables *addresses***:

```
x := 1      /* The value of a[i] */
y := 1      /* The value of a[i] */

x := 2      /* x + 1 */
i := 2      /* i + 1 */
y := 2      /* y + 1 */

a[1] := 2   /* the value of x is copied back to a[1] */
a[1] := 2   /* the value of y is copied back to a[1] (not a[2]!) */
```

and at the end the values **2, 2** are printed.

iii.

d. Κλήση με Όνομα (call by name)

i. Αφήνει τις πραγματικές παραμέτρους **χωρίς να υπολογιστεί η τιμή τους, μέχρι το χρονικό σημείο χρήσης τους στο ΥΠ.**

ii. Δηλαδή, οι πραγματικές παράμετροι αντιμετωπίζονται οι ίδιες σαν υποπρογράμματα χωρίς παραμέτρους (thunk), που εκτελούνται και υπολογίζεται η τιμή τους (για να δοθεί στην τυπική παράμετρο), με το τρέχον ΠΑ του προγράμματος ή ΥΠ το οποίο καλεί το τρέχον ΥΠ.

iii. Τότε, η τυπική παράμετρος συνδέεται με την πραγματική παράμετρο, όπως στην Κλήση με Αναφορά.

iv. Ο υπολογισμός της πραγματικής παραμέτρου, γίνεται εξ αρχής, κάθε φορά που χρησιμοποιείται η αντίστοιχη τυπική παράμετρος.

v. ALGOL-60, SIMULA (επιλογή του χρήστη)

Παράδειγμα 6.3 (call-by-name)

```
program params;
   var i: integer;
   a: array[1..2] of integer;
   procedure p(x,y: integer);
   begin
      x := x + 1;
      i := i + 1;
      y := y + 1;
   end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   p( a[i],a[i] );
   output( a[1],a[2] );
end.
```

Call by Name is *equivalent* to Call by Reference when simple variables are passed as parameters, but it is *different* when you pass an expression that denotes a memory location, like a subscript.

In this case the actual parameter is re-evaluated each time it is encountered. So in this case, this is the effect of the call of p(a[i],a[i]):

a[1] := 2 /* since i = 1, the result is equal to **a[1] + 1** */
i:= 2 /* i + 1 */
a[2] := 3 /* since i is now 2, the result is equal to **a[2] + 1** */
and at the end the values **2, 3** are printed.

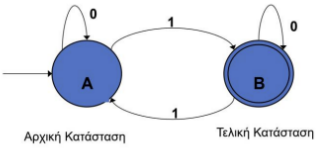
In practice the implementation calls an anonymous function (a "thunk"), each time it must evaluate a parameter.

vi.

10. Λεξική ανάλυση (DFA NFA) (3)

a.

Λεξική Ανάλυση (8)

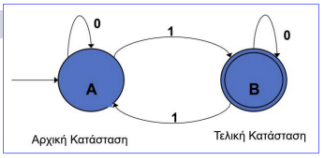


- Ισοδύναμη αναπαράσταση:

Πίνακας Καταστάσεων – Μεταβάσεων

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
A	0	A	ΟΧΙ
A	1	B	ΝΑΙ
B	0	B	ΝΑΙ
B	1	A	ΟΧΙ

Λεξική Ανάλυση (9)



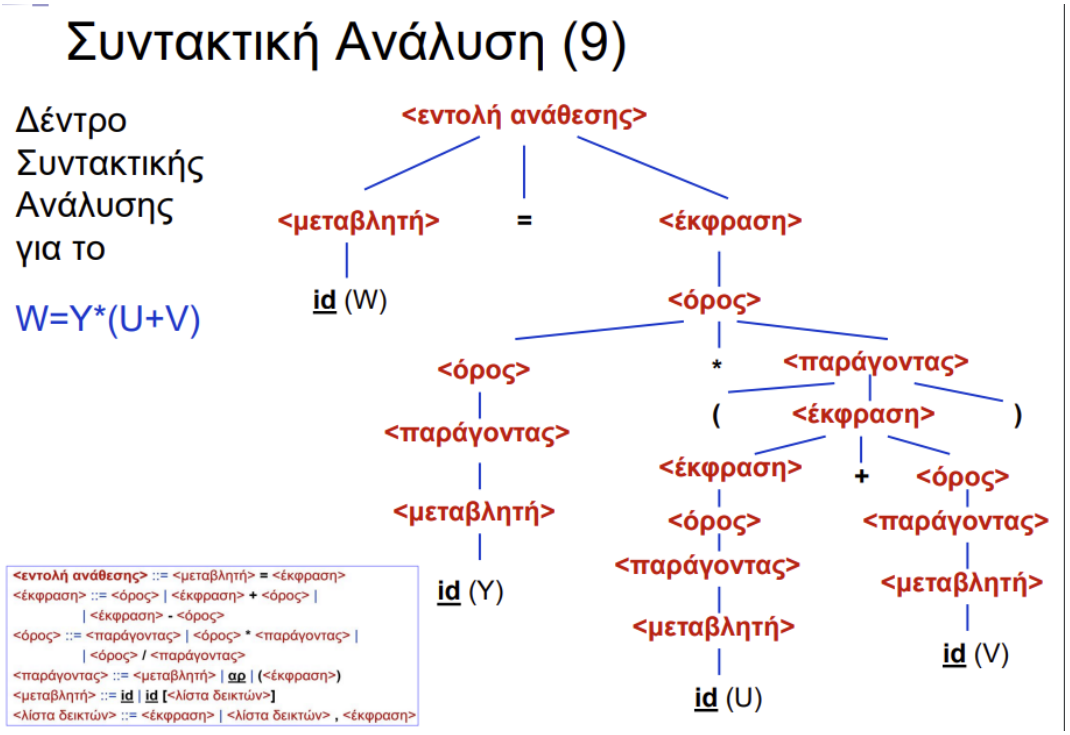
- Λειτουργία του FSA για την είσοδο **100101**:

Αναγνωσμένοι χαρακτήρες	Νέα Κατάσταση	Αποδοχή token
	A	ΟΧΙ
1	B	ΝΑΙ
10	B	ΝΑΙ
100	B	ΝΑΙ
1001	A	ΟΧΙ
10010	A	ΟΧΙ
100101	B	ΝΑΙ

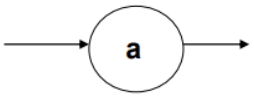
b.
c. Ένα Πεπερασμένο Αυτόματο έχει:
   i. Μία αρχική κατάσταση
   ii. Μία ή περισσότερες τελικές καταστάσεις
   iii. Ένα σύνολο μεταβάσεων
d. Κάθε string που ξεκινάει το Πεπερασμένο Αυτόματο από την αρχική κατάσταση, και τελειώνει σε μια τελική κατάσταση, είναι αποδεκτό token.
e. **Τα Πεπερασμένα Αυτόματα που χρησιμοποιούμε είναι ντετερμινιστικά.(DFA)**
f. **Μη-ντετερμινιστικά** είναι αυτά που έχουν **περισσότερες από μία μεταβάσεις με το ίδιο label.(NFA)**

11. Ορισμός της γραμματικής χωρίς συμφραζόμενα: **G={V,Σ,R,S}**
    a. **V**= Σύνολο **Μη Τερματικών συμβόλων** ,δηλαδή συμβόλων από τα οποία **αρχίζουν κανόνες**
    b. **Σ**= Σύνολο **Τερματικών συμβόλων** ,δηλαδή συμβόλων από τα οποία **δεν αρχίζουν κανόνες**
    c. **R**= οι κανόνες της γραμματικής
    d. **S**= το αρχικό σύμβολο της γραμματικής από το οποίο αρχίζουν οι κανόνες
12. BNF
    a. ::= ορίζεται ως
    b. | εναλλακτικός κανόνας – ή
    c. < … > Μη-τερματικό σύμβολο (συντακτική κατηγορία)
13. Parse Tree (Δέντρο Συντακτικής Ανάλυσης)

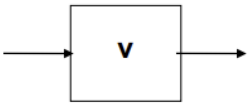
a.



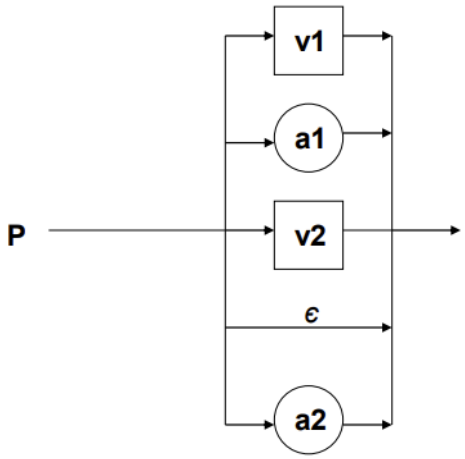
14. Συντακτικά Διαγράμματα κανόνων
    a. Τερματικό Σύμβολο a
        i.



    b. Μη-τερματικό Σύμβολο <v>
        i.

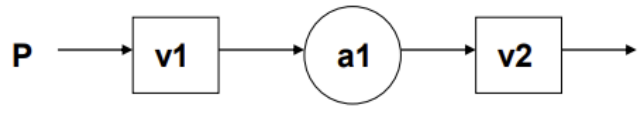


    c. <P>::= <v1> | a1 | <v2> | ε | a2
        i.



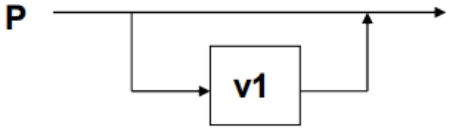
    d. <P>::= <v1> | a1 | <v2>

i.



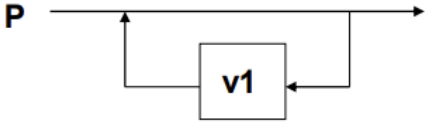
e. <P> ::= [<v1>] 0 ή 1 φορές το <v1> δηλ. <P>::= ε | <v1>

i.



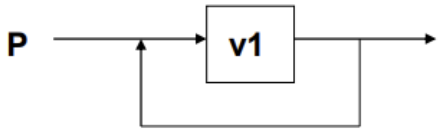
f. <P>::= { <v1> } 0 ή περισσότερες φορές το <v1> δηλ. <P>::= ε | <v1><P>

i.



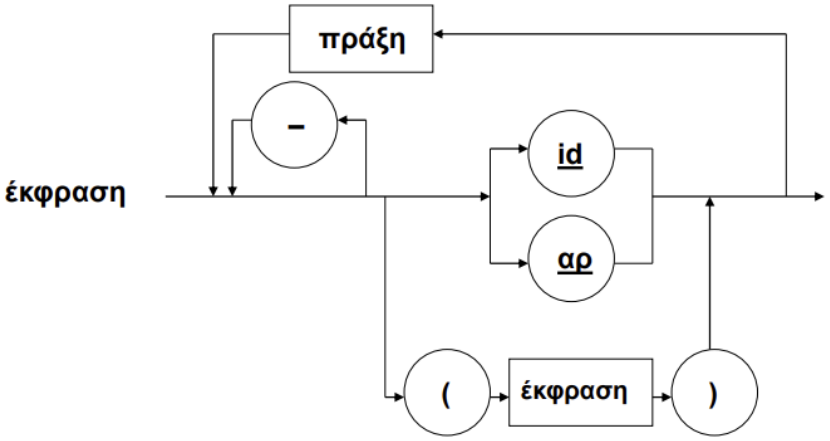
g. <P>::= <v1> | <v1><P> 1 ή περισσότερες φορές το <v1>

i.



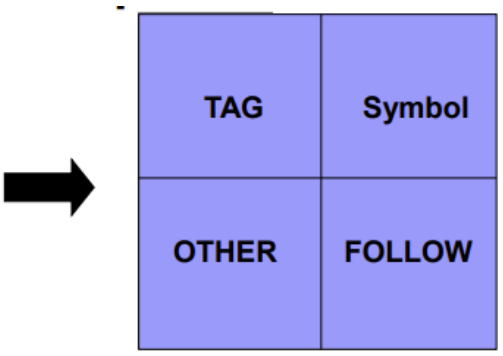
h. <έκφραση> ::= id | αρ | <έκφραση> <πράξη> <έκφραση> | | (<έκφραση>) | -<έκφραση>

i.

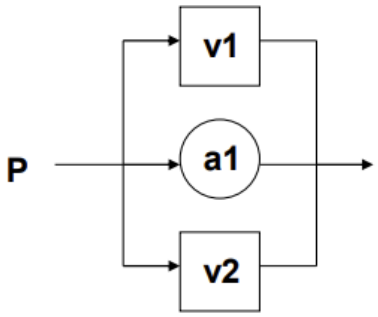


15. Ειδικές Δομές Δεδομένων:
    a. **TAG**: **TRUE** αν το Symbol είναι **τερματικό σύμβολο** FALSE αν το Symbol είναι **μη-τερματικό σύμβολο**
    b. **OTHER:** Pointer που **δείχνει τον επόμενο εναλλακτικό (παράλληλο) κόμβο** (**«nil» αν ο κόμβος είναι η τελευταία επιλογή**)
    c. **FOLLOW**: Pointer που **δείχνει τον επόμενο υποχρεωτικό (σειριακό) κόμβο** (**«nil» αν ο κόμβος είναι ο τελευταίος στη σειρά**)

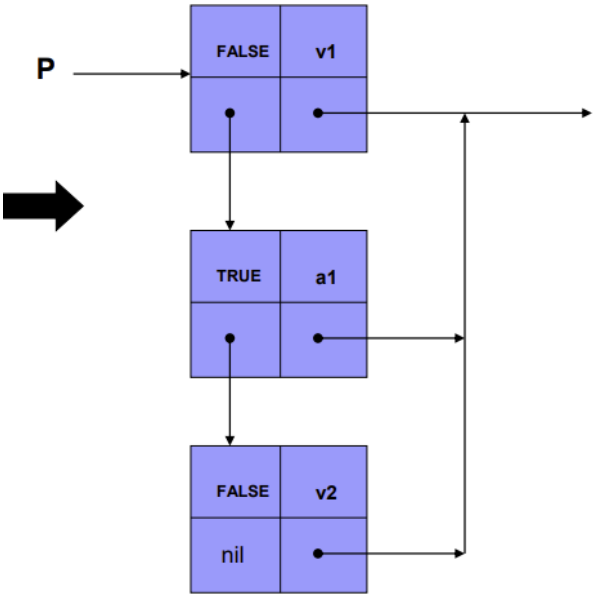
i.


TAG
Symbol
OTHER
FOLLOW

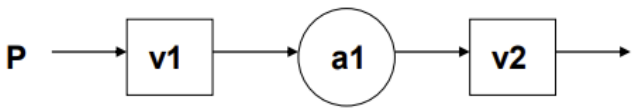
d.


P
v1
a1
v2

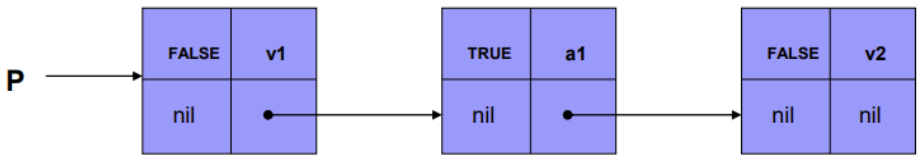
i.


P
FALSE
v1
TRUE
a1
FALSE
v2
nil

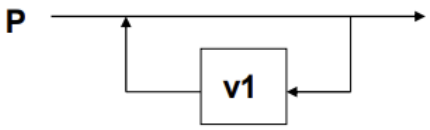
e.


P
v1
a1
v2

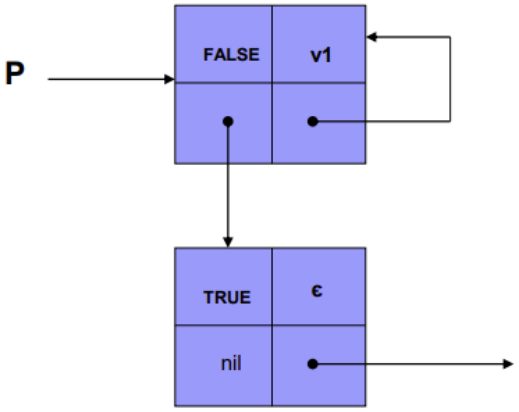
i.


P
FALSE
v1
nil
TRUE
a1
nil
FALSE
v2
nil
nil

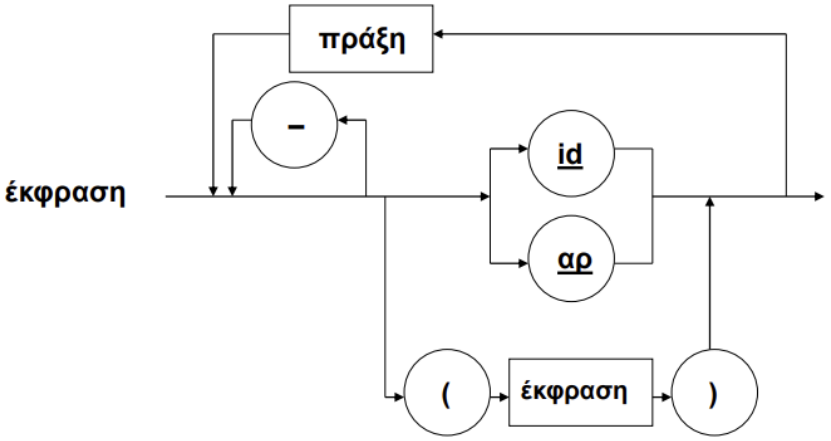
f.



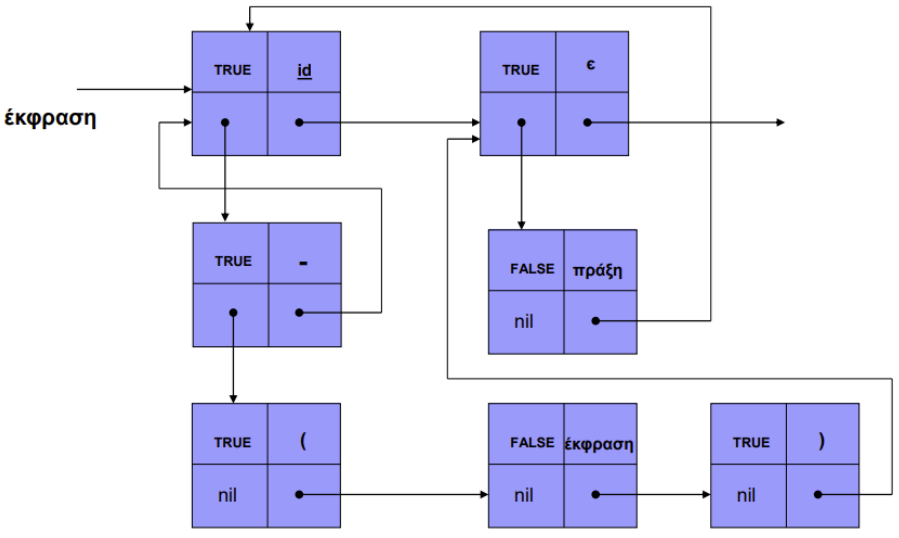
i.



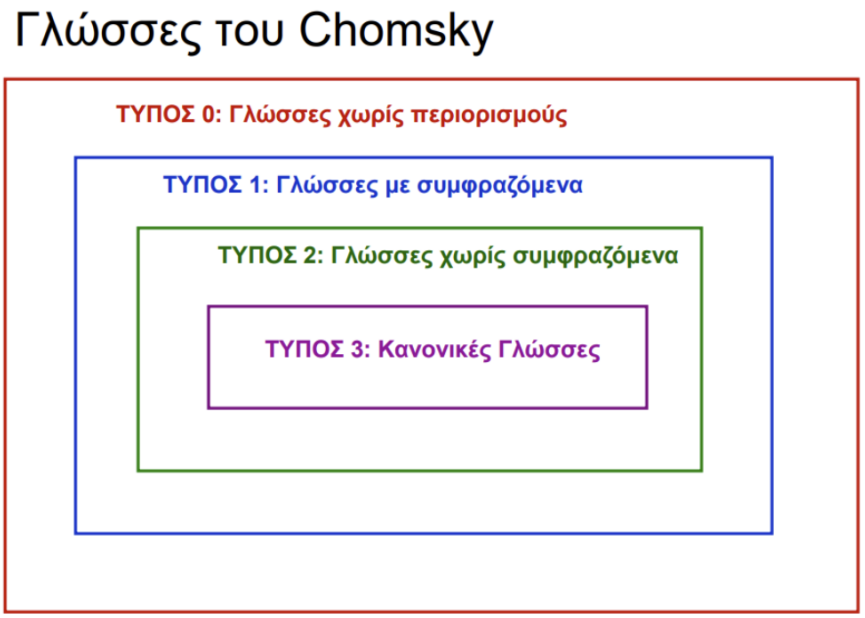
g.



i.



16. **Γλώσσες Chomsky**

a. 

17. **Διφορούμενες (ambiguous) Γραμματικές** Χωρίς Συμφραζόμενα είναι αυτές που υπάρχουν 2 ή περισσότερα δέντρα συντακτικής ανάλυσης για την ίδια παραγόμενη συμβολοσειρά.
18. **Κανονική Γραμματική** ονομάζεται αυτή που έχει στο δεξιό μερος κάθε κανόνα **το πολύ ένα μη τερματικό σύμβολο** και αυτό το σύμβολο αν υπάρχει **είναι το τελευταίο σύμβολο του κανόνα.**
    a. Από μία κανονική γραμματική μπορούμε να κατασκευάσουμε ένα DFA
19. **Γραμματική χωρίς συμφραζόμενα** είναι αυτή που τα αριστερά μελή αποτελούνται από ενα μή τερματικό σύμβολο.
20. **Γραμματική LL/1**
    a. Κανόνες:
        i. Να **μην** περιέχει **Αριστερή αναδρομή** δηλ. Κανόνες της μορφής **A->Aa**
        ii. Να **μην** έχει **εναλλακτικούς κανόνες** που **αρχιζουν με το ίδιο σύμβολο** δηλ. Κανόνες της μορφής **A->aB | aΓ**
        iii. Να **μην** έχει κανόνες τα δεξιά μερη των οποίων **παράγουν την κενή συμβολοσειρά** δηλ. **Όχι A->ε και B->ε**
    b. Κανόνες μετατροπής - Διόρθωσης Γραμματικής σε LL/1 Γραμματική:
        i. Απαλοιφή Αριστερής αναδρομής
            1. A->Aα1|Aα2|...|Aαn|β1|β2|...βm
            2. Η Απαλοιφή της Αριστερής Αναδρομής γίνεται ως εξής:
            3. A->β1B|β2B|...|βmB
            4. B->α1B|α2B|...|αnB|ε
        ii. Αριστερή Παραγοντοποίηση
            1. A->αβ1|αβ2|...|αβn
            2. Η Αριστερή Παραγοντοποίηση γίνεται ως εξής:
            3. A->αB
            4. B->β1|β2|...|βn
        iii. Κανόνες που οδηγουν σε ε
            1. A->ε
            2. B->ε

3. A->K
4. B->K
5. K->ε

iv. **Αντικατάσταση**

1. A->α1|α2|...|αn
2. B->β1Aβ2
3. A->α1|α2|...|αn
4. B->β1α1β2|β1α2β2|.....|β1αnβ2

21. Στοιβα Ολισθησης-Ελατωσης (Bottom-up Parser)

a. Κανόνες :
   i. **Ολίσθηση (shift)**: Αφαιρεί ένα σύμβολο από την αρχή του string και το βάζει στην κορυφή της στοίβας.
   ii. **Ελάττωση (reduce)**: Όταν στην κορυφή της στοίβας υπάρχει το δεξί μέλος παραγωγής. Αφαιρούνται αυτά τα σύμβολα από τη στοίβα και αντικαθίστανται από το αριστερό μέλος.
b. <S>::= r<B>
c. <B>::= <D> | <B>,<D>
d. <D>::= a | b
e. Συμβολοσειρά: ra,b
f.

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ε	ra,b EOF	Ολισθηση
1	r	a,b EOF	Ολισθηση
2	ra	,b EOF	Ελάττωση <D>::=a
3	r<D>	,b EOF	Ελάττωση <B>::= <D>
4	r<B>	,b EOF	Ολισθηση
5	r<B>,	b EOF	Ολισθηση
6	r<B>,b	EOF	Ελάττωση <D>::=b
7	r<B>,<D>	EOF	Ελάττωση <B>::= <B>,<D>
8	r<B>	EOF	Ελάττωση <S>::= r<B>
9		EOF	Αναγνώριση

g. Παρατηρήσεις:
   i. Για να βάλουμε στη στοίβα το αρχικό σύμβολο της γραμματικής πρέπει η συμβολοσειρά εισόδου να έχει εξαντληθεί αλλιώς θα έχουμε σφάλμα Ολισθησης - Ελάττωσης.
   ii. Επιλέγουμε τον κανόνα που διώχνει το μεγαλύτερο κομμάτι μέσα στη στοίβα. Αν δεν εφαρμόσουμε το σωστό κανόνα τότε θα έχουμε σφάλμα Ελάτωσσης-Ελάττωσης.

22. Στοίβα Πρόβλεψης-Ταιριάσματος (Top-Down Parser)

a. Κανόνες
   i. **Ταίριασμα συμβόλου:** Αν στην κορυφή της στοίβας βρίσκεται το **τερματικό σύμβολο a** και το **τρέχον σύμβολο του string εισόδου είναι επίσης a**, τότε το **a αφαιρείται από τη στοίβα** και διαβάζεται το επόμενο σύμβολο του string εισόδου.
   ii. **Πρόβλεψη:** Αν στην κορυφή της στοίβας βρίσκεται **το μη-τερματικό σύμβολο <A>**, το αντικαθιστούμε με **το δεξιό μέρος κάποιου κανόνα ορισμού του <A>, με τα σύμβολα σε αντίθετη σειρά.**
   iii. Αν καμία από τις δύο πράξεις δεν μπορεί να εφαρμοστεί, τότε υπάρχει συντακτικό σφάλμα.

b. <S>::= (<S>)<S> | ε
c. Συμβολοσειρά: ()
d.

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	() EOF	Πρόβλεψη <S> ::= (<S>)<S>
1	<S>)<S>(	() EOF	Ταίριασμα συμβόλου (
2	<S>)<S>	) EOF	Πρόβλεψη <S> ::= ε
3	<S>)	) EOF	Ταίριασμα συμβόλου)
4	<S>	EOF	Πρόβλεψη <S> ::= ε
5	ε	EOF	Αναγνώριση

e. Στη γραμματική πρέπει να υπάρχει κανόνας που που να οδηγεί στο ε. Αλλιώς οδηγούμαστε σε μή αναγνώριση.

23. Ιεραρχία Τελεστών

a.

FORTRAN	Pascal	C	Ada
**	NOT	!	** abs
* /	* / div mod AND	Postfix ++ --	* / mod
+ -	+ - OR	Prefix ++ --	Unary + -
.EQ. .NE. .GT. .LT. .LE. .GE.	= <> < <= > >=	Unary -	Binary + -
.NOT		* / %	
.AND.		Binary + -	
.OR.		< <= > >=	
		== !=	

		Bitwise (& ^ |)	
		&&	
		||	
		=	

24. Python

a. Μεταβλητές
   i. μπορεί να περιλαμβάνει λατινικούς χαρακτήρες ή αριθμούς (0…9) καθώς και το _
   ii. αλλά δεν μπορεί να αρχίζει με αριθμητικό ψηφίο.
   iii. Τύποι αριθμητικών μεταβλητών:
      1. ακέραιος (integer)
      2. δεκαδικός (floating point)
      3. συμβολοσειρά (string)
   iv. Με την εντολή type(x) εμφανίζεται ο τύπος μιας μεταβλητής
b. Σύμβολα πράξεων:
   i. + Πρόσθεση
   ii. - Αφαίρεση
   iii. * Πολλαπλασιασμός
   iv. / Διαίρεση
   v. // Ακέραια διαίρεση (πηλίκο)
   vi. % modulo (υπόλοιπο διαίρεσης)
   vii. ** Ύψωση σε δύναμη
c. Συντομογραφίες πράξεων:
   i. x += y Πρόσθεση του y στην τιμή του x (x = x + y)
   ii. x -= y Αφαίρεση του y από την τιμή του x (x = x – y)
   iii. x *= y Πολλαπλασιασμός του y με την τιμή του x (x = x * y)
   iv. x /= y Διαίρεση του y με την τιμή του x (x = x / y)
   v. x //= y Ακέραια διαίρεση του y με x (x = x // y)
   vi. x %= y. Υπόλοιπο της διαίρεσης του y με x (x = x % y)
   vii. x **= y Ύψωση του y στο x (x = x **y)
d. Συμβολοσειρές:
   i. Μια συμβολοσειρά είναι μια ακολουθία χαρακτήρων που περικλείεται από μονά ή διπλά εισαγωγικά Κάθε χαρακτήρας έχει τη θέση του στο αλφαριθμητικό οι οποίες ξεκινούν από τη θέση 0 . Π.χ. name=“Νικόλαος” (8 χαρακτήρες)
   ii. name[2:6] ->‘κολα’
e. Εντολές output
   i. print()
      1. print(ορίσματα, sep =‘δ’, end = ‘ε’)
      2. δ : διαχωριστής
      3. ε : τερματικός χαρακτήρας
         a. x = 124.456

4. **print("η τιμή του χ είναι:", x)**
    a. η τιμή του χ είναι: 124.456
5. **print("η τιμή του χ είναι %f" % (x))**
    a. η τιμή του χ είναι 124.456000
6. **print("η τιμή του χ είναι %1.2f" % (x))**
    a. η τιμή του χ είναι 124.46

f. Εντολές input
    i. input()
        1. x=input(“Δώσε μια τιμή:”)
        2. **Προσοχή: Η input() επιστρέφει συμβολοσειρά**

g. Δομή επανάληψης for
    i. for μέλος in αντικείμενο :
    ii. ομάδα εντολών 1
    iii. else:
    iv. ομάδα εντολών 2 (η ομάδα εντολών 2 δεν εκτελείται αν υπάρχει break)

h. for/break/continue
    i. for μέλος in αντικείμενο :
    ii. ομάδα εντολών 1
    iii. if συνθήκη :
    iv. continue
    v. if συνθήκη :
    vi. break
    vii. else:
    viii. ομάδα εντολών 2

i. range: επανάληψη for με απαρίθμηση

j. for i in range(start, end, step) : ομάδα εντολών 1

25. LISP

a. Συναρτήσεις στη Lisp ορίζονται με τη σύνταξη:
    i. `(defun όνομα_συνάρτησης (παράμετροι) τιμή)`

b. Υπολογισμός της τιμής μιας συνάρτησης στη Lisp γίνεται με τη σύνταξη:
    i. `(όνομα_συνάρτησης παράμετροι)`

c. Ορισμένες χρήσιμες συναρτήσεις είναι οι:
    i. (if e tv fv)
        1. επιστρέφει tv αν η e είναι αληθής (t) και fv αν είναι ψευδής (nil).
    ii. (+ a b)
        1. επιστρέφει a + b.
    iii. (- a b)
        1. επιστρέφει a - b.
    iv. (* a b)
        1. επιστρέφει a * b.
    v. (/ a b)
        1. επιστρέφει a / b.
    vi. (equal a b)
        1. επιστρέφει αληθές (t) αν a = b.

d. Παράδειγμα (ορισμός της συνάρτησης του παραγοντικού):
    i. `(defun factorial (n)`
    ii. `(if (equal n 0)`

iii. `1`
iv. `(* n (factorial (- n 1)))))`

e. Στη Lisp μπορούμε να ορίσουμε μια λίστα (*list*) από τιμές με τη σύνταξη '(τιμή1 τιμή2 τιμή3 ...)
f. Η ειδική τιμή nil παριστάνει την κενή λίστα.
g. Μπορούμε να αναφερθούμε σε σύμβολα με τη σύνταξη 'όνομα.
h. Παράδειγμα:
   i. `'(1 2 3 4 42)`
   ii. `'Word`
   iii. `'('Maria 'John 'Aliki)`
i. Βασικές συναρτήσεις επεξεργασίας λιστών είναι οι παρακάτω:
   i. (null list)
      1. επιστρέφει αληθές αν η λίστα είναι κενή
   ii. (cons val list)
      1. επιστρέφει τη λίστα list με πρώτο το στοιχείο val,
   iii. (car list)
      1. επιστρέφει το πρώτο στοιχείο μιας λίστας,
   iv. (cdr list)
      1. επιστρέφει τα υπόλοιπα (όλα εκτός από το πρώτο) στοιχεία μιας λίστας ή nil αν αυτά δεν υπάρχουν.
j. Με βάση τις συναρτήσεις αυτές μπορούμε να ορίσουμε πιο σύνθετες συναρτήσεις.
   i. # Length
   ii. Επιστρέφει το μήκος μιας λίστας.
      1. `(defun mylength (a)`
      2. `(if (null a`
      3. `0`
      4. `(+ (mylength (cdr a)) 1)))`
   iii. # Append
   iv. Ενώνει δύο λίστες.
      1. `(defun myappend (a b)`
      2. `(if (null a)`
      3. `B`
      4. `(cons (car a) (myappend (cdr a) b))))`
   v. # Reverse
   vi. Αντιστρέφει μια λίστα.
   vii. `(defun myreverse (a)`
   viii. `(if (null a)`
   ix. `nil`
   x. `(myappend (myreverse (cdr a)) (cons (car a) nil))))`
k. Μπορούμε να ορίσουμε μια συνάρτηση ανωτέρου βαθμού (*higher order function*) ορίζοντας ως ένα όρισμα της συνάρτησης μια άλλη συνάρτηση.
l. Τη συνάρτηση-όρισμα μπορούμε να την εφαρμόσουμε πάνω σε κάποιες τιμές με τη συνάρτηση apply η οποία λαμβάνει ως όρισμα μια συνάρτηση το πρώτο

όρισμα και μια λίστα με τα υπόλοιπα όρισματά της (ή nil) και επιστρέφει το αποτέλεσμα της συνάρτησης εφαρμοσμένο στο όρισμα.

m. Μπορούμε ακόμα να ορίσουμε δυναμικά μια τιμή τύπου συνάρτησης με τη σύνταξη:
   i. `(lambda (όρισμα) τιμή)`

n. Το παρακάτω παράδειγμα συνθέτουμε τις έννοιες αυτές για να ορίσουμε τις συναρτήσεις map και reduce.

   i. # Map

   ii. Η συνάρτηση map μετατρέπει μια λίστα σε μια άλλη με βάση μια συνάρτηση που έχει δοθεί ως όρισμα.

```
1. (defun mymap (f lst)
2. (if (null lst)
3. nil
4. (cons (apply f (cons (car lst) nil)) (mymap f
   (cdr lst)))))
```

   5. Έτσι μπορούμε π.χ. να διπλασιάσουμε να στοιχεία της λίστας '(1 2 3) με την κλήση:
   6. `(mymap (lambda (x) (* 2 x)) '(1 2 3))`
   7. `(2 4 6)`

   iii. # Reduce

   iv. Η συνάρτηση reduce συμπυκνώνει μια λίστα εφαρμόζοντας αναδρομικά τη συνάρτηση σε κάθε στοιχείο της λίστας αρχίζοντας από μια αρχική τιμή.

```
1. (defun myreduce (f v lst)
2. (if (null lst)
3. v
4. (apply f (cons (car lst) (cons (myreduce f v
   (cdr lst)) nil)))))
```

   v. Έτσι μπορούμε να ορίσουμε συναρτήσεις όπως τις:
      1. *sum* επιστρέφει το σύνολο των τιμών μιας λίστας
         a. `(defun mysum (lst) (myreduce '+ 0 lst))`
      2. *product* επιστρέφει το γινόμενο των τιμών μιας λίστας
         a. `(defun myproduct (lst) (myreduce '* 1 lst))`
      3. *alltrue* επιστρέφει αληθές αν όλες οι τιμές μιας λίστας είναι αληθείς
         a. `(defun alltrue (lst) (myreduce 'and t lst))`
      4. *anytrue* επιστρέφει αληθές αν τουλάχιστον μια τιμή μιας λίστας είναι αληθής
      5. `(defun anytrue (lst) (myreduce 'or nil lst))`