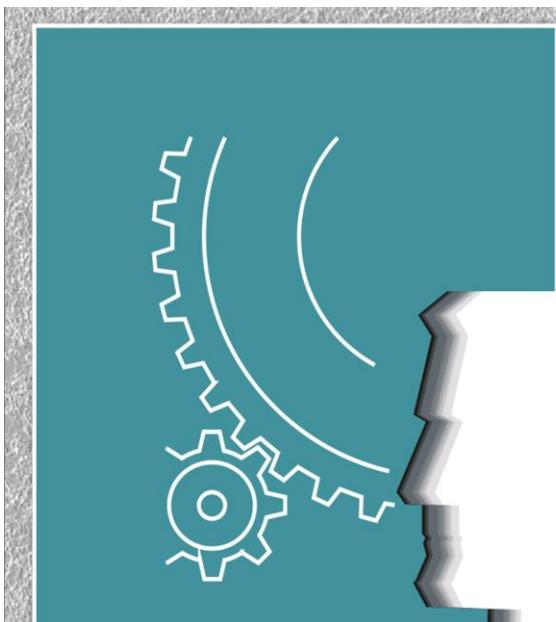


ΣΗΜΕΙΩΣΕΙΣ ΣΤΙΣ ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ



ανάλυση
COMPUTER

ΕΡΓΑΣΤΗΡΙΟ ΕΛΕΥΘΕΡΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

*...λυση
σπουδων*

ΑΓ. ΑΝΔΡΕΟΥ 130-132 & ΓΟΥΝΑΡΗ, Τ.Κ. 26222, ΠΑΤΡΑ

ΤΗΛ: 310-097, 324-900 - ΤΗΛ/FAX: 313-547

E-MAIL: janelisj@otenet.gr, janelisk@otenet.gr

Computer – Ανάλυση

Περιεχόμενα

1 ΘΕΜΑΤΑ ΜΕ ΠΕΡΙΒΑΛΛΟΝΤΑ ΑΝΑΦΟΡΑΣ, ΣΤΟΙΒΑ ΕΚΤΕΛΕΣΗΣ ΚΑΙ ΚΛΗΣΗ ΥΠΟΠΡΟΓΡΑΜΜΑΤΩΝ	7
1.1 ΦΡΟΝΤΙΣΤΗΡΙΟ 2011	7
1.2 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2016.....	13
1.3 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2009.....	16
1.4 ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2009.....	18
1.5 ΘΕΜΑ 4 ΣΕΠΤΕΜΒΡΙΟΣ 2012.....	20
1.6 ΘΕΜΑ 1 ΆΤΥΠΗ ΝΟΕΜΒΡΙΟΣ 2011.....	22
1.7 ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2012 ΚΑΙ ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2015	24
1.8 ΆΣΚΗΣΗ 1 ΑΠΟ ΦΡΟΝΤΙΣΤΗΡΙΟ ΙΟΥΝΙΟΣ 2022	27
1.9 ΘΕΜΑ 5 ΙΟΥΝΙΟΣ 2012.....	29
1.10 ΘΕΜΑ 3- ΙΟΥΝΙΟΣ 2019 -ΟΜΑΔΑ Α	32
1.11 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2010 ΚΑΙ ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2020	35
1.12 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2010	38
1.13 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2011 ΚΑΙ ΣΕΠΤΕΜΒΡΙΟΣ 2017 ΚΑΙ ΣΕΠΤΕΜΒΡΙΟΣ 2022	40
1.14 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2014.....	44
1.15 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2013.....	47
1.16 ΘΕΜΑ 5 ΙΟΥΝΙΟΣ 2008 ΚΑΙ ΦΕΒΡΟΥΑΡΙΟΣ 2017	49
1.17 ΦΡΟΝΤΙΣΤΗΡΙΟ 2007	50
1.18 ΘΕΜΑ 1 ΆΤΥΠΗ 2009.....	52
1.19 ΘΕΜΑ 2 -ΙΟΥΝΙΟΣ 2015	54
1.20 ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2021	61
1.21 ΘΕΜΑ 2 -ΣΕΠΤΕΜΒΡΙΟΣ 2015.....	66
1.22 ΆΣΚΗΣΗ 1 ΦΡΟΝΤΙΣΤΗΡΙΟ 2015	70
1.23 ΆΣΚΗΣΗ 2 ΦΕΒΡΟΥΑΡΙΟΣ 2018	74
1.24 ΘΕΜΑ 4 ΣΕΠΤΕΜΒΡΙΟΣ 2016.....	77
1.25 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2017.....	79
1.26 ΘΕΜΑ 3 ΦΕΒΡΟΥΑΡΙΟΣ 2019.....	81
1.27 ΦΡΟΝΤΙΣΤΗΡΙΟ 2019	83
1.28 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2018	84
1.29 ΘΕΜΑ 3- ΙΟΥΝΙΟΣ 2019 -ΟΜΑΔΑ Β.....	89
1.30 ΘΕΜΑ 3- ΣΕΠΤΕΜΒΡΙΟΣ 2019 -ΟΜΑΔΑ Α	91
1.31 ΘΕΜΑ 1- ΦΕΒΡΟΥΑΡΙΟΣ 2020	93
1.32 ΘΕΜΑ 2- ΙΟΥΝΙΟΣ 2020	96
1.33 ΘΕΜΑ 2 ΦΕΒΡΟΥΑΡΙΟΣ 2021	99
1.34 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2021	102
1.35 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2020	106
1.36 ΆΣΚΗΣΗ 2 ΑΠΟ ΦΡΟΝΤΙΣΤΗΡΙΟ ΙΟΥΝΙΟΣ 2022	111
1.37 ΘΕΜΑ 1 ΦΕΒΡΟΥΑΡΙΟΣ 2023.....	114
1.38 ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2023.....	122
2 ΘΕΜΑΤΑ ΚΑΙ ΑΣΚΗΣΕΙΣ ΜΕ NFA, DFA ΚΑΙ ΚΑΝΟΝΙΚΕΣ ΕΚΦΡΑΣΕΙΣ.....	125
2.1 ΒΑΣΙΚΗ ΘΕΩΡΙΑ	125
2.2 ΑΣΚΗΣΕΙΣ ΜΕ DFA, NFA ΚΑΙ ΚΑΝΟΝΙΚΕΣ ΕΚΦΡΑΣΕΙΣ	126
2.2.1 Παραδείγματα με DFA	126
2.2.2 Παραδείγματα με NFA	130
2.2.3 Παραδείγματα Κανονικών Εκφράσεων.....	134
2.2.4 Παραδείγματα με DFA, NFA και Κανονικές Εκφράσεις.....	136
2.3 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2012.....	138
2.4 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2008.....	140
2.5 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2007.....	141
2.6 ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2009.....	141
2.7 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2010.....	142
2.8 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2013.....	143
2.9 ΘΕΜΑ 4 ΙΟΥΝΙΟΣ 2012.....	143
2.10 ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2014.....	146
2.11 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2014 ΚΑΙ ΘΕΜΑ 1 ΦΕΒΡΟΥΑΡΙΟΣ 2018	147
2.12 ΘΕΜΑ 2 ΆΤΥΠΗ 2011.....	148
2.13 ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2015.....	149
2.14 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2015.....	151
2.15 ΘΕΜΑ 1 ΦΕΒΡΟΥΑΡΙΟΣ 2015.....	152
2.16 ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2016.....	154
2.17 ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2016.....	155
2.18 ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2017.....	157
2.19 ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2017.....	158
2.20 ΘΕΜΑ 1 ΦΕΒΡΟΥΑΡΙΟΣ 2019.....	159

2.21	ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2019-ΟΜΑΔΑ Α.....	160
2.22	ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2019-ΟΜΑΔΑ Β.....	160
2.23	ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2018.....	161
2.24	ΘΕΜΑ 1 ΦΕΒΡΟΥΑΡΙΟΣ 2020.....	163
2.25	ΘΕΜΑ 3 ΣΕΠΤΕΜΒΡΙΟΣ 2012.....	166
2.26	ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2021.....	167
2.27	ΦΡΟΝΤΙΣΤΗΡΙΟ 2023 – ΆΣΚΗΣΗ 4	169
3	ΘΕΜΑΤΑ ΚΑΙ ΑΣΚΗΣΕΙΣ ΜΕ ΓΡΑΜΜΑΤΙΚΕΣ ΧΩΡΙΣ ΣΥΦΜΡΑΖΟΜΕΝΑ	171
3.1	ΟΡΙΣΜΟΣ ΓΡΑΜΜΑΤΙΚΗΣ ΧΩΡΙΣ ΣΥΜΦΡΑΖΟΜΕΝΑ.....	171
3.1.1	Παράδειγμα 1 με Γραμματική Χωρίς Συμφραζόμενα.....	171
3.1.2	Παράδειγμα 2 με Γραμματική Χωρίς Συμφραζόμενα.....	173
3.2	ΚΑΤΑΣΚΕΥΗ ΓΡΑΜΜΑΤΙΚΩΝ ΧΩΡΙΣ ΣΥΜΦΡΑΖΟΜΕΝΑ	176
3.3	ΘΕΜΑΤΑ ΜΕ ΚΑΤΑΣΚΕΥΗ ΓΡΑΜΜΑΤΙΚΩΝ ΓΧΣ.....	181
3.3.1	Θέμα 5 Ιούνιος 2010	181
3.3.2	Θέμα 2 Άτυπη Νοέμβριος 2011.....	182
3.3.3	Θέμα 2 Σεπτέμβριος 2012.....	185
3.3.4	Θέμα 1 Ιούνιος 2009	187
3.3.5	Θέμα 3 Σεπτέμβριος 2004.....	188
3.4	ΘΕΜΑΤΑ ΜΕ ΓΡΑΜΜΑΤΙΚΕΣ LL/1	189
3.4.1	Προϋποθέσεις Γραμματικής LL/1	189
3.4.2	Κανόνες Μετατροπής-Διόρθωσης Γραμματικής σε LL/1.....	189
3.4.3	Παράδειγμα 1 με LL/I γραμματική	190
3.4.4	Παράδειγμα 2 με LL/I γραμματική	190
3.4.5	Θέμα 3 Σεπτέμβριος 2010.....	191
3.4.6	Θέμα 3 Άτυπη Φεβρουάριος 2012 και Σεπτέμβριος 2016.....	194
3.5	ΘΕΜΑΤΑ ΜΕ PARSERS.....	196
3.5.1	Παράδειγμα με Στοίβα Ολίσθησης-Ελάττωσης (Bottom-Up Parser).....	196
3.5.2	Παράδειγμα με Στοίβα Πρόβλεψης-Ταιριάσματος (Top-Down Parser).....	196
3.5.3	Θέμα 4 Ιούνιος 2012	197
3.5.4	Θέμα 3 Σεπτέμβριος 2013 και Σεπτέμβριος 2022	199
3.5.5	Θέμα 5 Σεπτέμβρης 2010.....	202
3.5.6	Θέμα 5 Ιούνιος 2012	206
3.5.7	Θέμα 2 Ιούνιος 2014	207
3.5.8	Θέμα 3 Ιούνιος 2015	210
3.5.9	Θέμα 3 Σεπτέμβριος 2015 και Άτυπη 2016	213
3.5.10	Θέμα 3 Ιούνιος 2015 και Θέμα 3 Φεβρουάριος 2018	215
3.5.11	Θέμα 4 Ιούνιος 2017	217
3.5.12	Θέμα 4 Σεπτέμβριος 2017.....	222
3.5.13	Θέμα 4 Φεβρουάριος 2019	225
3.5.14	Θέμα 1 Ιούνιος 2019-Ομάδα A	228
3.5.14.1	Θέμα 1 Ιούνιος 2019-Ομάδα Β και Φεβρουάριος 2023	233
3.5.15	Φροντιστήριο 2019	237
3.5.16	Θέμα 3 Σεπτέμβριος 2016	239
3.5.17	Θέμα 3 Σεπτέμβριος 2018	241
3.5.18	Θέμα 3 Σεπτέμβριος 2019	245
3.5.19	Θέμα 3 Φεβρουάριος 2020	247
3.5.20	Θέμα 1 Ιούνιος 2020	250
3.5.21	Θέμα 1 Σεπτέμβριος 2020	253
3.5.22	Θέμα 1 Φεβρουάριος 2021	256
3.5.23	Θέμα 1 Σεπτέμβριος 2021	259
3.5.24	Θέμα 4 Ιούνιος 2021	265
3.5.25	Θέμα 1 Ιούνιος 2022	268
3.5.26	Φροντιστήριο 2023 – Άσκηση 1	273
3.5.27	Φροντιστήριο 2023 – Άσκηση 2	276
3.5.28	Φροντιστήριο 2023 – Άσκηση 3	280
3.5.29	Θέμα 1 Ιούνιος 2023	284
4	ΘΕΜΑΤΑ ΘΕΩΡΙΑΣ ΚΑΙ ΚΩΔΙΚΑ	287
4.1	ΘΕΜΑ 4 ΣΕΠΤΕΜΒΡΙΟΣ 2013	287
4.2	ΘΕΜΑ 1 ΣΕΠΤΕΜΒΡΙΟΣ 2012, 2015 ΚΑΙ ΣΕΠΤΕΜΒΡΙΟΣ 2018	287
4.3	ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2012-OXI	290
4.4	ΘΕΜΑ 1 ΙΟΥΝΙΟΣ 2013	292
4.5	ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2014	294
4.6	ΘΕΜΑ 5 ΙΟΥΝΙΟΣ 2014	295
4.7	ΘΕΜΑ 4 ΣΕΠΤΕΜΒΡΙΟΣ 2014 ΚΑΙ ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2017	296
4.8	ΘΕΜΑ 5 ΣΕΠΤΕΜΒΡΙΟΣ 2014	298

4.9	ΘΕΜΑ 4 ΙΟΥΝΙΟΣ 2015.....	299
4.10	ΘΕΜΑ 3 ΦΕΒΡΟΥΑΡΙΟΣ 2020.....	302
4.11	ΘΕΜΑ 5 ΙΟΥΝΙΟΣ 2015.....	304
4.12	ΘΕΜΑ 5 ΙΟΥΝΙΟΣ 2017.....	305
4.13	ΘΕΜΑ 2 ΆΤΥΠΗ ΝΟΕΜΒΡΙΟΣ 2011.....	306
4.14	ΘΕΜΑ 6 ΙΟΥΝΙΟΣ 2017.....	306
4.15	ΘΕΜΑ 5 ΣΕΠΤΕΜΒΡΙΟΣ 2017 ΚΑΙ ΣΕΠΤΕΜΒΡΙΟΣ 2019.....	308
4.16	ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2022 – ΕΡΩΤΗΜΑ Α	310
4.17	ΘΕΜΑ 3 ΙΟΥΝΙΟΣ 2022 – ΕΡΩΤΗΜΑ Β	313
4.18	ΘΕΜΑ 2 ΣΕΠΤΕΜΒΡΙΟΣ 2022	315
4.19	ΘΕΜΑ 2 ΙΟΥΝΙΟΣ 2023.....	316
5	PYTHON	319
5.1	ΜΕΤΑΒΛΗΤΕΣ	319
5.2	ΠΑΡΑΔΕΙΓΜΑ ΔΗΛΩΣΗΣ ΚΑΙ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΜΕΤΑΒΛΗΤΩΝ	319
5.3	ΕΝΤΟΛΕΣ ΕΙΣΟΔΟΥ/ΕΞΟΔΟΥ	320
5.3.1	Απλή Εντολή <i>Eξόδου print</i>	320
5.3.2	Εντολή <i>Eξόδου print με αποτελέσματα σε άγκιστρα</i>	320
5.3.3	Εντολή <i>Eξόδου print με καθορισμό μορφοποίησης</i>	320
5.3.4	Εντολή <i>Εισόδου</i>	321
5.3.4.1	Casting	321
5.4	ΜΕΡΙΚΕΣ ΔΙΑΦΟΡΕΣ PYTHON ΑΠΟ C, C++, JAVA	322
5.5	ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ ΣΤΗΝ PYTHON	322
5.5.1	Συντομογραφίες Αριθμητικών πράξεων	323
5.5.2	Boolean (Λογικοί) Τελεστές	323
5.6	ΤΕΛΕΣΤΕΣ == ID ΚΑΙ IS	324
5.7	ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS)	325
5.7.1	Συναρτήσεις Διαχείρισης Συμβολοσειρών.....	330
5.7.2	Πράξεις Συμβολοσειρών.....	333
5.8	ΔΟΜΕΣ ΕΝΤΟΛΩΝ ΣΤΗΝ PYTHON	334
5.8.1	Δομή Επιλογής - Εντολή if.....	334
5.8.2	Δομές Επανάληψης- Εντολή for	336
5.8.3	Δομές Επανάληψης- Εντολή while	339
5.9	ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΣΤΗΝ PYTHON	341
5.9.1	Λίστες (Lists)	341
5.9.2	Πλειάδες (Tuples)	350
5.9.3	Σύνολα (sets)	351
5.9.4	Λεξικά (Dictionaries)	353
5.10	ΣΥΝΑΡΤΗΣΕΙΣ (FUNCTIONS)	358
5.10.1	Εντολή return	363
5.11	ΣΥΝΑΡΤΗΣΕΙΣ ΛΑΜΒΔΑ (LAMBDA FUNCTIONS)	365
5.12	ΑΝΑΔΡΟΜΗ ΣΕ PYTHON	368
5.13	ΚΛΑΣΕΙΣ/ΑΝΤΙΚΕΙΜΕΝΑ (CLASSES/OBJECTS)	371
5.14	ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ ΣΤΗΝ PYTHON	375
5.15	ΘΕΜΑΤΑ ΣΕ PYTHON	379
5.15.1	Θέμα 1 ^ο Ιούνιος 2021	379
5.15.2	Θέμα 8 ^ο Ιούνιος 2021	379
5.15.3	Θέμα 2 ^ο Ιούνιος 2021.....	380
5.15.4	Θέμα 3 ^ο Ιούνιος 2021.....	380
5.15.5	Θέμα 4 ^ο Ιούνιος 2021.....	381
5.15.6	Θέμα 5 ^ο Ιούνιος 2021-OXI.....	381
5.15.7	Θέμα 6 ^ο Ιούνιος 2021.....	383
5.15.8	Θέμα 7 ^ο Ιούνιος 2021.....	383
5.15.9	Θέμα 9 ^ο Ιούνιος 2021-OXI.....	384
5.15.10	Πρόγραμμα Python για την εκτύπωση περιττών αριθμών σε μια λίστα	385
5.15.11	Θέμα 4 ^η Ομάδα Α Ιούνιος 2019	386
5.15.12	Πρόγραμμα Python για εναλλαγή του πρώτου και του τελευταίου στοιχείου σε μια λίστα	387
5.15.13	Πρόγραμμα Python που ελέγχει την ύπαρξη στοιχείου σε μια λίστα	388
5.15.14	Πρόγραμμα Python για Αντιστροφή Λίστας	389
5.15.15	Πρόγραμμα Python για μέτρηση Αρτιων και Περιττών σε μια Λίστα	390
5.15.16	Πρόγραμμα Python για εκτύπωση θετικών αριθμών και μηδενικών σε μια Λίστα	392
5.15.17	Πρόγραμμα Python που μετρά το πλήθος θετικών και αρνητικών σε μια Λίστα.....	393
5.15.18	Πρόγραμμα Python για τη διαγραφή της N-οστής εμφάνισης μιας δοθείσας λέξης σε μια λίστα - OXI.....	395
5.15.19	Πρόγραμμα Python για τη διαγραφή πολλαπλών στοιχείων από μια λίστα - OXI	397
5.15.20	Θέμα 4 ^η Ομάδα Α Ιούνιος 2019	399
5.15.21	Πρόγραμμα Python για έλεγχο αν ένα string είναι παλίνδρομο ή όχι.....	400
5.15.22	Πρόγραμμα Python εφαρμογής της Αναδικής Αναζήτησης (Binary Search) -OXI	401

5.15.23	Πρόγραμμα Python για τη μέτρηση εμφανίσεων τιμών σε μια λίστα χρησιμοποιώντας λεξικά.....	403
5.15.24	Θέμα 4 Ομάδα B Ιούνιος 2019	405
5.15.25	Πρόγραμμα Python για τον υπολογισμό του αθροίσματος όλων των στοιχείων ενός λεξικού	406
5.15.26	Θέμα 5α Σεπτέμβριος 2019.....	406
5.15.27	Θέμα 2α Ιούνιος 2020	407
5.15.28	Θέμα 2β Ιούνιος 2020.....	407
5.15.29	Θέμα 3 Ιούνιος 2020.....	408
5.15.30	Θέμα 4 Ιούνιος 2020.....	408
5.15.31	Θέμα 1 Σεπτέμβριος 2020.....	409
5.15.32	Θέμα 2 Σεπτέμβριος 2020.....	409
5.15.33	Θέμα 3 Σεπτέμβριος 2020.....	410
5.15.34	Θέμα 3 Σεπτέμβριος 2020.....	410
5.15.35	Θέμα 4 και 5 Σεπτέμβριος 2020.....	411
5.15.36	Θέμα 6 Σεπτέμβριος 2020.....	411
5.15.37	Θέμα με Dictionary	412
5.15.38	Θέμα 5 Σεπτέμβριος 2021 με Dictionary	412
5.15.39	Θέμα 7 Σεπτέμβριος 2020 με Dictionary	413
5.15.40	Θέμα 1 Σεπτέμβριος 2021 με Dictionary	413
5.15.41	Θέμα 3 Σεπτέμβριος 2021 με Dictionary	414
5.15.42	Θέμα 1 Φεβρουάριος 2021	414
5.15.43	Θέμα 2 Φεβρουάριος 2021	416
5.15.44	Θέμα 1 Σεπτέμβριος 2021.....	419
5.15.45	Θέμα 2 Σεπτέμβριος 2021.....	421
5.15.46	Θέμα 3 Σεπτέμβριος 2021.....	421
5.15.47	Θέμα 4 Σεπτέμβριος 2021.....	422
5.15.48	Θέμα 6 Σεπτέμβριος 2021.....	423
5.15.49	Θέμα 7 Σεπτέμβριος 2021.....	424
5.15.50	Θέμα 7 Σεπτέμβριος 2021.....	425
5.15.51	Θέμα 7 Σεπτέμβριος 2021.....	425
5.15.52	Θέμα 11 Σεπτέμβριος 2021.....	426
5.15.53	Θέμα 8 Σεπτέμβριος 2021.....	426
5.15.54	Θέμα 9 Σεπτέμβριος 2021.....	427
5.15.55	Θέμα 10 Σεπτέμβριος 2021.....	427
5.15.56	Θέμα 12 Σεπτέμβριος 2021.....	427
5.15.57	Θέμα 13 Σεπτέμβριος 2021.....	428
5.15.58	Θέμα 14 Σεπτέμβριος 2021.....	429
5.15.59	Θέμα 15 Σεπτέμβριος 2021.....	429
5.15.60	Θέμα 16 Σεπτέμβριος 2021.....	430
5.15.61	Θέμα 1 Σεπτέμβριος 2021.....	431
5.15.62	Θέμα 4 Σεπτέμβριος 2021.....	431
5.15.63	Θέμα 5 Σεπτέμβριος 2021.....	432
5.15.64	Θέμα 6 Σεπτέμβριος 2021.....	433
5.15.65	Θέμα 7 Σεπτέμβριος 2021.....	433
5.15.66	Θέμα 8 Σεπτέμβριος 2021.....	434
5.15.67	Θέμα 9 Σεπτέμβριος 2021.....	435
5.15.68	Θέμα 4 Ιούνιος 2022.....	437
5.15.69	Θέμα 3 Σεπτέμβριος 2022.....	439
5.15.70	Θέμα Φεβρουάριος 2023	440
5.15.71	Θέμα Ιούνιος 2023.....	442
5.16	ΘΕΜΑΤΑ ΣΕ PYTHON ΑΠΟ ΗΛΕΚΤΡΟΛΟΓΟΥΣ	444
5.16.1	Θέμα 1.....	444
5.16.2	Θέμα 2.....	444
5.16.3	Θέμα 3.....	444
5.16.4	Θέμα 4.....	444
5.16.5	Θέμα 5-OXI.....	446
5.16.6	Θέμα 6.....	446
5.17	ΘΕμΑ 7.....	446
5.18	ΘΕμΑ 8.....	447
5.19	ΘΕμΑ 9.....	447
5.20	ΘΕμΑ 10.....	448
6	Α-ΛΟΓΙΣΜΟΣ	449
6.1	ΈΝΝΟΙΕΣ ΣΥΝΑΡΤΗΣΙΑΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ	449
6.2	ΛΟΓΙΣΜΟΣ -Λ- ΘΕΩΡΗΤΙΚΕΣ ΈΝΝΟΙΕΣ	450
6.3	ΕΛΕΥΘΕΡΗ ΚΑΙ ΔΕΣΜΕΥΜΕΝΗ ΜΕΤΑΒΛΗΤΗ.....	452
7	ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ SCALA	453

7.1	ΒΑΣΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ	453
7.2	ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ	453
7.3	ΠΑΡΑΔΕΙΓΜΑΤΑ SCALA	454
7.3.1	Σύνταξη Κώδικα σε Scala.....	454
7.3.2	Βασικές Εντολές Scala.....	454
7.3.3	Διαφορές Scala και Java	454
7.3.4	Παράδειγμα 4	454
7.3.5	Διερμηνευτής Εντολών Scala.....	455
7.3.6	Ανόνυμες Συναρτήσεις (<i>Anonymous Functions</i>)	457
7.3.7	Currying.....	458
8	ΣΥΝΑΡΤΗΣΙΑΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (FUNCTIONAL PROGRAMMING).....	461
8.1	ΜΕΙΩΣΕΙΣ-ΈΛΑΤΤΩΣΕΙΣ (REDUCTIONS)	461
8.2	ΕΝΑΛΛΑΚΤΙΚΕΣ ΜΕΙΩΣΕΙΣ.....	462
8.3	ΔΙΑΦΟΡΕΤΙΚΟΙ ΤΡΟΠΟΙ ΥΠΟΛΟΓΙΣΜΟΥ ΤΟΥ SQUARE.....	462
8.4	EAGER EVALUATION STRATEGY VS LAZY EVALUATION	463
8.4.1	Πιθανό Θέμα: Παράδειγμα 1 με Eager Evaluation και Lazzy Evaluation.....	463
8.4.2	Πιθανό Θέμα: Παράδειγμα 2 με Eager Evaluation και Lazzy Evaluation.....	463
8.4.3	Πιθανό Θέμα: Παράδειγμα 3-Εκτέλεση Περιττών Υπολογισμών	464
8.4.4	Πλεονεκτήματα-Μειονεκτήματα.....	464
8.4.5	Διπλές Μειώσεις Υποεκφράσεων	464
8.5	ΣΤΡΑΤΗΓΙΚΗ ΜΕΙΩΣΗΣ ΓΡΑΦΟΥ (GRAPH REDUCTION)	465
8.5.1	Παράδειγμα με μείωση Γράφου (Graph Reduction).....	465
8.5.2	Μείωση Γράφου για την εντολή let	465
8.5.3	Μειώσεις για συναρτήσεις υψηλότερης τάξης και currying.....	466
9	ΣΥΝΑΡΤΗΣΙΑΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ – LISP -SCHEME	467
9.1	ΈΝΝΟΙΕΣ ΣΥΝΑΡΤΗΣΙΑΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ	467
9.2	ΣΥΝΟΨΗ ΤΗΣ SCHEME.....	468
9.3	ΘΕΩΡΗΤΙΚΕΣ ΒΑΣΕΙΣ: ΛΟΓΙΣΜΟΣ - Λ.....	469
9.4	ΕΠΙΕΞΕΡΓΑΣΙΑ BIG DATA ΚΑΙ MAPREDUCE.....	469
9.5	ΣΥΝΑΡΤΗΣΙΑΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (FUNCTIONAL PROGRAMMING).....	469
9.6	LISP	469
9.6.1	Συναρτήσεις.....	470
9.6.2	Άλλα Χαρακτηριστικά	470
9.6.3	Αναδρομή σε Lisp	471
9.6.4	MapReduce.....	472
9.7	ΘΕΜΑΤΑ ΣΕ ΣΥΝΑΡΤΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ	475
9.7.1	Θέματα με Αποτίμηση Εκφράσεων με Fold και Map	475
9.7.1.1	Θέμα 5β Σεπτέμβριος 2019	475
9.7.1.2	Θέμα 5α Ιούνιος 2022	475
9.7.1.3	Θέμα 5α Ιούνιος 2023.....	476
9.7.1.4	Θέμα 5b Ιούνιος 2022	476
9.7.1.5	Θέμα 5c Ιούνιος 2022.....	477
9.7.1.6	Θέμα 5c Ιούνιος 2021.....	478
9.7.1.7	Θέμα Ιούνιος 2022	479
9.7.1.8	Πιθανό Θέμα - Σχήματα σε Lisp (Lis Scheme) με lambda expressions	479
9.7.1.9	Πιθανό Θέμα - Σχήματα σε Lisp (Lis Scheme) με Map/Fold.....	480
9.7.2	Διάφορα Θέματα.....	481
9.7.2.1	Θέμα 5γ Σεπτέμβριος 2019	481
9.7.3	Θέματα με σχήματα σε Lisp (Lis Scheme) με lambda expressions	483
9.7.3.1	Θέμα 5 Ομάδα B Ιούνιος 2019.....	483
9.7.4	Θέματα με σταδιακή αποτίμηση σε λ-λογισμό	487
9.7.4.1	Θέμα Ιούνιος 2022	487
9.7.4.2	Θέμα Ιούνιος 2023	487
9.7.5	Θέματα με αντικατάσταση Αναδρομής με Επανάληψη	488
9.7.5.1	Θέμα Ιούνιος 2022	488
9.7.5.2	Θέμα Σεπτέμβριος 2022	488
9.7.5.3	Θέμα Ιούνιος 2023	489
9.7.5.4	Θέμα 5 Ομάδα A Ιούνιος 2019	489
9.7.6	Θέματα με συμπλήρωση λογικών εκφράσεων σε γλώσσα scheme	490
9.7.6.1	Θέμα 5b Ιούνιος 2021	490
9.7.6.2	Θέμα Ιούνιος 5α 2022.....	490
9.8	ΘΕΜΑΤΑ ΙΟΥΝΙΟΣ 2023.....	ΣΦΑΛΜΑ! ΔΕΝ ΕΧΕΙ ΟΡΙΣΤΕΙ ΣΕΛΙΔΟΛΕΙΚΤΗΣ.

1 ΘΕΜΑΤΑ ΜΕ ΠΕΡΙΒΑΛΛΟΝΤΑ ΑΝΑΦΟΡΑΣ, ΣΤΟΙΒΑ ΕΚΤΕΛΕΣΗΣ ΚΑΙ ΚΛΗΣΗ ΥΠΟΠΡΟΓΡΑΜΜΑΤΩΝ

1.1 Φροντιστήριο 2011

Δίνεται το παρακάτω πρόγραμμα σε Pascal:

```
program MAIN;
var
  i, j, k, m: integer;
procedure Q(i, m: integer);
begin
  i:=i + k;
  m:=j + 1;
  write(i, j, k, m); (A)
end;
procedure P(i, j:integer);
var
  k: integer;
begin
  k:=3;
  i:=i+k;
  j:=j+k;
  Q(i, j);
end;
begin
  i:=1;
  j:=2;
  k:=4;
  P(i, k);
  write(i, j, k); (B)
end.
```

(a) Υποθέτοντας ότι χρησιμοποιείται στατικός κανόνας εμβέλειας να ορίσετε τα περιβάλλοντα αναφοράς (τοπικό, μη-τοπικό και καθολικό) των MAIN, P, Q

(b) Να απαντήστε στο προηγούμενο ερώτημα αν χρησιμοποιείται δυναμικός κανόνας εμβέλειας

(c) Με χρήση στατικού κανόνα εμβέλειας να παρουσιάσετε τις τιμές που θα εκτυπωθούν στις γραμμές A και B όταν η μεταβίβαση παραμέτρων γίνεται με:

- (1) call by value
- (2) call by value result
- (3) call by reference

(d) Με χρήση δυναμικού κανόνα εμβέλειας να παρουσιάσετε τις τιμές που θα εκτυπωθούν στις γραμμές A και B όταν η μεταβίβαση παραμέτρων γίνεται με:

- (1) call by value
- (2) call by value result
- (3) call by reference

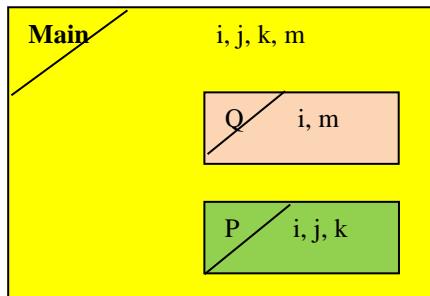
Απάντηση

Παρατήρηση

Για το main (κύριο πρόγραμμα) και για κάθε υποπρόγραμμα και για το main δημιουργούμε ένα ξεχωριστό μπλοκ.

Θεωρία

Στατικός κανόνας εμβέλειας σημαίνει ότι γράφουμε τα μπλοκ των υποπρογραμμάτων με τη σειρά που τα συναντάμε στον κώδικα όπως φαίνεται στην ακόλουθη εικόνα:



Περιβάλλοντα Αναφοράς

- Στο Τοπικό περιβάλλον γράφουμε τις μεταβλητές και τα υποπρογράμματα που δηλώνονται **ΜΟΝΟ στο μπλοκ που βρισκόμαστε**
- Στο ΜΗ τοπικό περιβάλλον γράφουμε τις μεταβλητές και τα υποπρογράμματα που δηλώνονται **σε ΟΛΑ ΤΑ ΠΕΡΙΒΑΛΛΟΝΤΑ μπλοκ αυτού στο οποίο βρισκόμαστε**
- Στο Καθολικό περιβάλλον γράφουμε τις μεταβλητές και τα υποπρογράμματα που δηλώνονται **ΜΟΝΟ ΣΤΟ ΠΙΟ ΕΞΩΤΕΡΙΚΟ ΠΕΡΙΒΑΛΛΟΝ μπλοκ αυτού στο οποίο βρισκόμαστε**

Τα περιβάλλοντα Αναφοράς των MAIN, P και Q είναι τα ακόλουθα:

Main

Τοπικό: i, j, k, m, P, Q

Μη-Τοπικό και Καθολικό: —

Παρατήρηση

Στην Pascal ΔΕΝ ΟΠΙΖΕΤΑΙ ΠΟΤΕ Μη-Τοπικό και Καθολικό περιβάλλον για τη MAIN

Q

Τοπικό: i, m

Μη-Τοπικό και Καθολικό: j, k, P, Q

Παρατήρηση

Όταν υπάρχει μόνο ένα περιβάλλον μπλοκ γύρω από αυτό που βρισκόμαστε, τότε το ΜΗ ΤΟΠΙΚΟ ΚΑΙ ΤΟ ΚΑΘΟΛΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΤΑΥΤΙΖΟΝΤΑΙ. Επίσης οι μεταβλητές αναφέρονται μόνο σε ένα μπλοκ δηλαδή δεν κάνουμε διπλοδήλωση μεταβλητών ενώ τα υποπρογράμματα τα δηλώνουμε όσες φορές χρειάζεται.

P

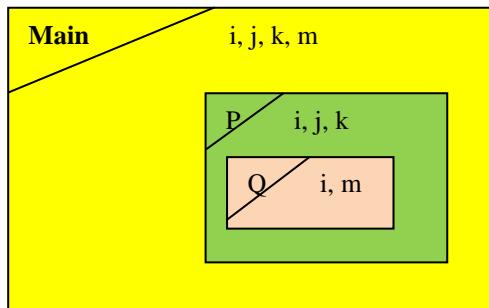
Τοπικό: i, j, k

Μη-Τοπικό και Καθολικό: m, P, Q

(b)

Θεωρία

Δυναμικός κανόνας εμβέλειας σημαίνει ότι γράφουμε τα μπλοκ των υποπρογραμμάτων με τη σειρά που εκτελούνται και όχι με τη σειρά που τα συναντάμε στον κώδικα.



Main

Τοπικό: i, j, k, m, P, Q

Μη-Τοπικό και Καθολικό: —

Παρατήρηση

Στην Pascal ΔΕΝ ΥΠΑΡΧΕΙ ΠΟΤΕ Μη-Τοπικό και Καθολικό περιβάλλον για τη MAIN

Q

Τοπικό: i, m

Μη-Τοπικό: j, k (από P), P, Q (από Main)

Καθολικό: P, Q

Παρατήρηση

Όταν υπάρχει μόνο ένα περιβάλλον block από αυτό που βρισκόμαστε, τότε το ΜΗ ΤΟΠΙΚΟ ΚΑΙ ΚΑΘΟΛΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΤΑΥΤΙΖΟΝΤΑΙ. Το αμέσως περιβάλλον block του Q είναι το P

P

Τοπικό: i, j, k

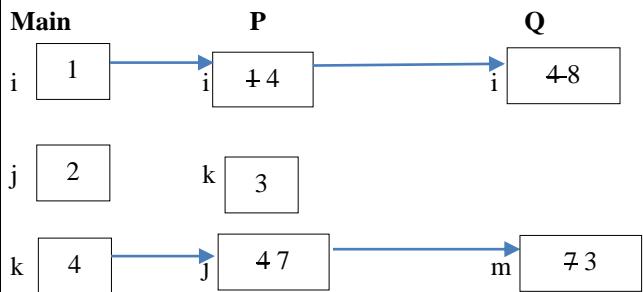
Μη-Τοπικό και Καθολικό: m, P, Q

(c)

Θεωρία για Τρόπους Κλήσης

- Στο **Call by value** καλούμε ένα υποπρόγραμμα, του μεταβιβάζουμε τιμές, αυτές καταχωρούνται σε αντίστοιχες παραμέτρους, το υποπρόγραμμα εκτελείται και όταν τελειώσει ΔΕΝ επιστρέφει αποτελέσματα μέσω των παραμέτρων του (δηλ. οι παράμετροι του ΔΕΝ επιστρέφουν πίσω στις παραμέτρους του καλούντος προγράμματος)
- Στο **Call by value-result** καλούμε ένα υποπρόγραμμα, του μεταβιβάζουμε τιμές, αυτές καταχωρούνται σε αντίστοιχες παραμέτρους, το υποπρόγραμμα εκτελείται και όταν τελειώσει επιστρέφει αποτελέσματα μέσω των παραμέτρων του (δηλ. οι παράμετροι του υποπρογράμματος επιστρέφουν αντίστροφα πίσω στις παραμέτρους του καλούντος προγράμματος όταν το υποπρόγραμμα τελειώσει)
- Στο **Call by reference** καλούμε ένα υποπρόγραμμα, του μεταβιβάζουμε τις διευθύνσεις μεταβλητών σε παραμέτρους (όχι τις τιμές τους) και οι διευθύνσεις αυτές καταχωρούνται σε δείκτες. Μέσω των δεικτών τροποποιούμε απευθείας τις μεταβλητές του main είτε του καλούντος υποπρογράμματος

Call by value



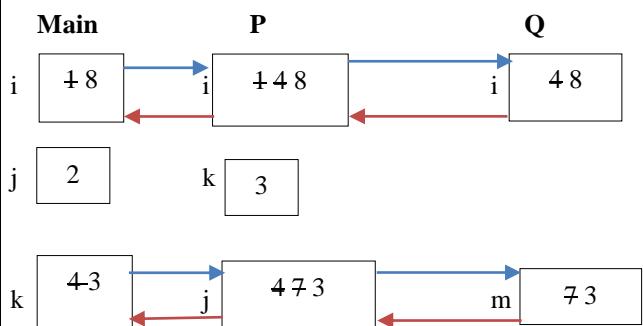
Τυπώνονται οι τιμές: 8, 2, 4, 3 στη γραμμή (A)

Τυπώνονται οι τιμές: 1, 2, 4 στη γραμμή (B)

Παρατήρηση

Όταν σε υποπρόγραμμα δεν υπάρχει δηλωμένη μια μεταβλητή που χρησιμοποιείται σε αυτό, τότε την παίρνουμε από το ΑΜΕΣΩΣ ΕΠΟΜΕΝΟ ΠΕΡΙΒΑΛΛΟΝ μπλοκ.

Call by value-result



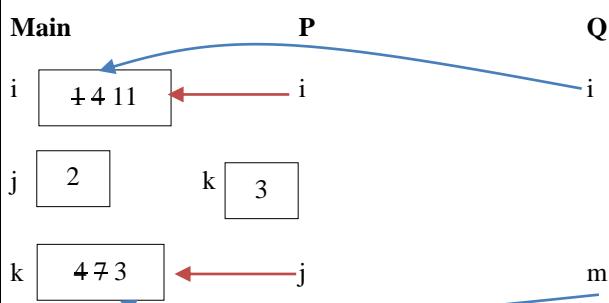
Τυπώνονται οι τιμές: 8, 2, 4, 3 στη γραμμή (A)

Τυπώνονται οι τιμές: 8, 2, 3 στη γραμμή (B)

Παρατήρηση

Όταν σε υποπρόγραμμα δεν υπάρχει μια μεταβλητή που χρησιμοποιούμε την παίρνουμε από το ΑΜΕΣΩΣ ΠΕΡΙΒΑΛΛΟΝ BLOCK

Call by reference



Τυπώνονται οι τιμές: 11, 2, 3, 3 στη γραμμή (A)

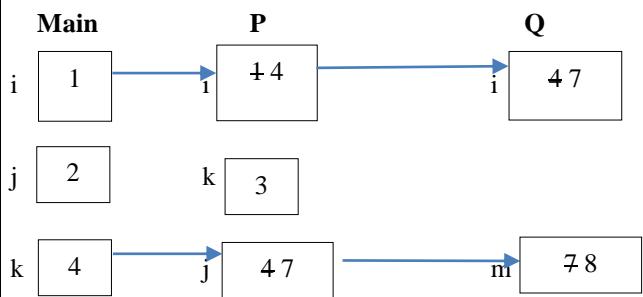
Τυπώνονται οι τιμές: 11, 2, 3 στη γραμμή (B)

Παρατήρηση

Όταν μεταβιβάζουμε δείκτη σε δείκτη κάνουμε το δεύτερο δείκτη να δείχνει όπου και ο πρώτος.

(d)

Call by value



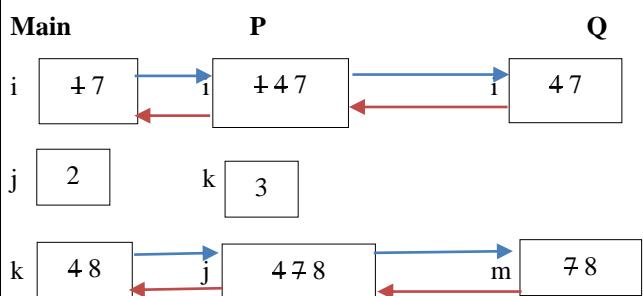
Τυπώνονται οι τιμές: 7, 7, 3, 8

Τυπώνονται οι τιμές: 1, 2, 4

Παρατήρηση

Εδώ αλλάζει το ΑΜΕΣΩΣ ΠΕΡΙΒΑΛΛΟΝ BLOCK σε σχέση με το στατικό κανόνα εμβέλειας. Το αμέσως περιβάλλον block της procedure Q είναι η procedure P

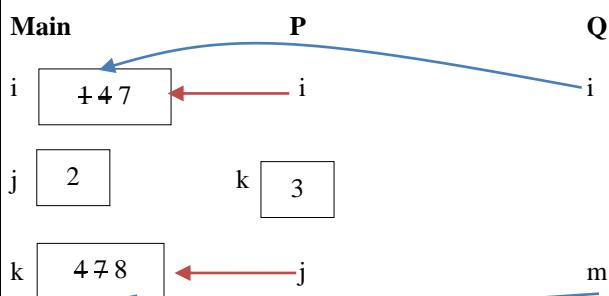
Call by value-result



Τυπώνονται οι τιμές: 7, 7, 3, 8

Τυπώνονται οι τιμές: 7, 2, 8

Call by reference



Τυπώνονται οι τιμές: 7, 8, 3, 8

Τυπώνονται οι τιμές: 7, 2, 8

1.2 Θέμα 1 Σεπτέμβριος 2016

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί στατικό κανόνα εμβέλειας.

program MAIN;

var

i: **integer**;

A: **array** [1..2] **of integer**;

procedure F(x, y, z: **integer**);

begin

x:=x+1;

y:=z;

z:=z + 1;

end;

begin

i:=1; A[1]:=10; A[2]:=11;

F(i, A[i], i);

write(i, A[1], A[2]);

end.

(a) Να ορίσετε τα περιβάλλοντα αναφοράς (τοπικό, μη-τοπικό και καθολικό) όλων των τμημάτων του προγράμματος

(b) Ποιες τιμές τυπώνονται με την τελευταία εντολή στο κυρίως πρόγραμμα αν η μεταβίβαση γίνει με:

(1) call by value

(2) call by value result

(3) call by reference

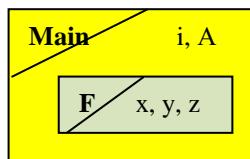
(4) call by name

Απάντηση

(a)

Παρατήρηση

Όταν υπάρχει μόνο ένα υποπρόγραμμα ακόμα και να μην αναφέρει τον κανόνα εμβέλειας δεν έχει σημασία γιατί ο στατικός και ο δυναμικός κανόνας εμβέλειας ταυτίζονται



Τα περιβάλλοντα Αναφοράς των MAIN και F είναι τα ακόλουθα:

Main

Τοπικό: i, A, F (κλήση)

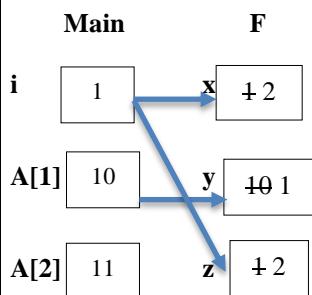
Μη-Τοπικό και Καθολικό: —

F

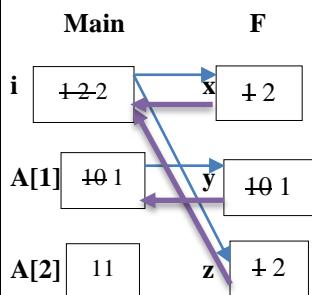
Τοπικό: x, y, z

Μη-Τοπικό και Καθολικό: i, A, F (κλήση)

(b)

Call by value

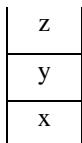
Τυπώνονται οι τιμές: 1, 10, 11 στο Main

Call by value-result

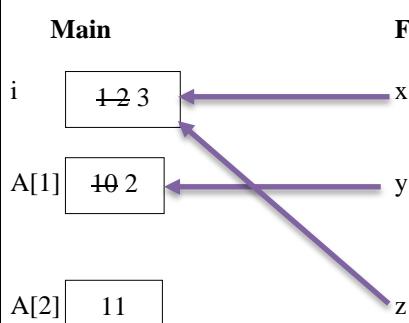
Τυπώνονται οι τιμές: 2, 1, 11 στο Main

Παρατήρηση για το call by value result

Οι παράμετροι μια συνάρτησης μπαίνουν σε μια στοίβα με τη σειρά που γράφονται στην παρένθεση με τις παραμέτρους της συνάρτησης δηλ.



Όταν τερματίσει η συνάρτηση οι παράμετροι βγαίνουν από τη στοίβα με τη μέθοδο LIFO. Άρα εδώ πρώτα επιστρέφει πίσω το z και καταχωρείται στο I, μετά εξάγεται το y και καταχωρείται στο A[1] και τέλος το x και καταχωρείται στο i.

Call by reference

Τυπώνονται οι τιμές: 3, 2, 11 στο Main

Παρατήρηση για το call by reference

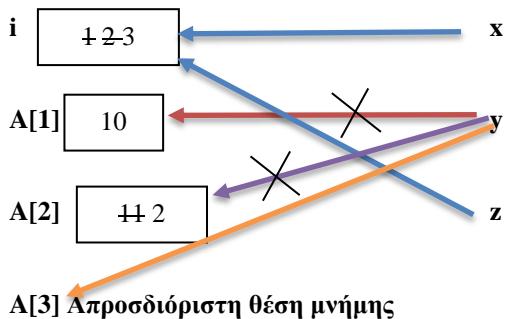
Όταν δύο διαφορετικοί δείκτες δείχνουν στην ίδια θέση μνήμης αυτό ονομάζεται φαινόμενο ψευδωνυμίας.

Call by name

To call by name είναι μια μείζη του call by value και του call by reference. Συγκεκριμένα όταν οι παράμετροι που μεταβιβάζονται είναι μεταβλητές, τότε το call by name ταυτίζεται με το call by reference, ενώ όταν οι παράμετροι που μεταβιβάζονται είναι σταθερές ή παραστάσεις τότε το call by name ταυτίζεται με το call by value

Main

F



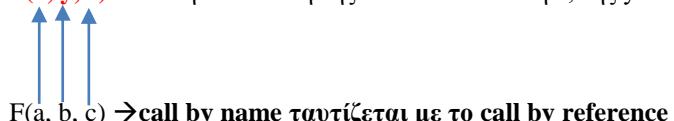
Τυπώνονται οι τιμές: 3, 10, 2 στο Main

Παρατήρηση για τους πίνακες στο call by name

Στο call by name και μόνο όταν έχουμε πίνακες της μορφής $A[i]$ όταν αλλάξει ο δείκτης i του στοιχείου του πίνακα, τότε ο αντίστοιχος δείκτης που δείχνει στο στοιχείο αυτό, καταργείται από το στοιχείο αυτό και δείχνει αυτόματα στο νέο στοιχείο $A[i]$ ανάλογα με τη νέα τιμή του i .

Παραδείγματα με Call By Name

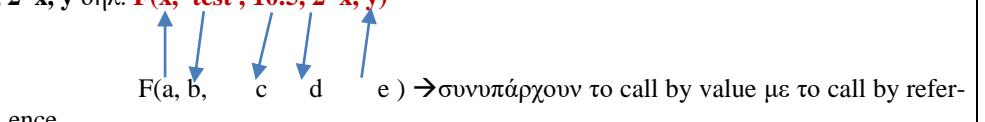
Καλείται η F με παραμέτρους x, y, z δηλ. $F(x, y, z)$ οπότε η διεύθυνση της x πάει στο δείκτη a , της y στο δείκτη b και της z στο δείκτη c



Καλείται η F με παραμέτρους $5, 'test', 10.5$ δηλ. $F(5, 'test', 10.5)$



Καλείται η F με παραμέτρους $x, 'test', 10.5, 2*x, y$ δηλ. $F(x, 'test', 10.5, 2*x, y)$



1.3 Θέμα 2 Ιούνιος 2009

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί στατικό κανόνα εμβέλειας.

```
program MAIN;  
var  
  y, x: integer;  
  
function f(x: integer):integer;  
begin  
  if x=0 then  
    y:=1;  
  else  
    y:=f(x-1)*y+1;  
  f:=y;  
end;  
  
begin  
  y:=1;  
  x:=f(2);  
  write(x, y);  
end.
```

Δεδομένου ότι χρησιμοποιείται **στατικός κανόνας εμβέλειας και call by value**:

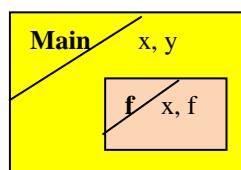
- Να γράψετε τα περιβάλλοντα αναφοράς
- Τι εκτυπώνει η write;
- Να δώσετε το **run-time stack και τις τιμές των τοπικών μεταβλητών για κάθε activation record** αφού κληθεί και η τελευταία αναδρομή.

Απάντηση

Παρατήρηση για τις Function

Η function είναι ένας δεύτερος τύπος υποπρογράμματος στην Pascal (μετά τις procedures) που το όνομα της έχει διπλό ρόλο. Συγκεκριμένα το όνομα μιας function είναι και όνομα υποπρογράμματος και ταυτόχρονα είναι και όνομα μεταβλητής στην οποία καταχωρείται το τελικό αποτέλεσμα που επιστρέφει η function όταν τελειώσει. Η τελευταία εντολή σε κάθε function είναι η καταχώριση του τελικού αποτελέσματος που θα επιστρέψει στο όνομα της. Εδώ η τελευταία εντολή της function f είναι η **f:=y;**

a)



Τα περιβάλλοντα Αναφοράς των MAIN και f είναι τα ακόλουθα:

Main

Τοπικό: x, y, f (κλήση)

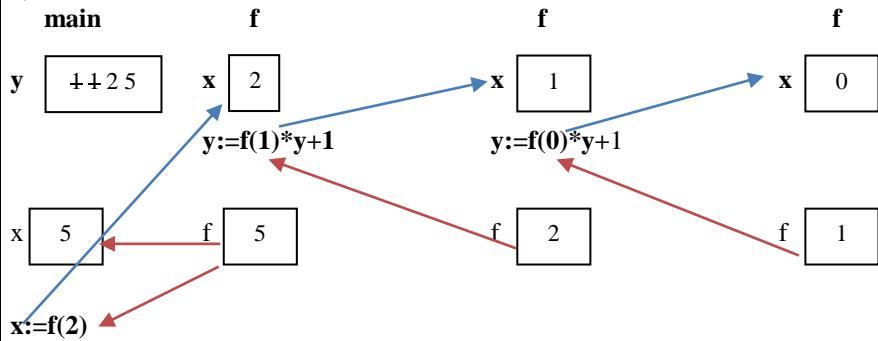
Μη-Τοπικό και Καθολικό: –

f

Τοπικό: x, f (τιμή)

Μη-Τοπικό και Καθολικό: y, f (κλήση)

b)



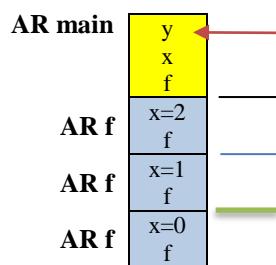
H write(x, y) στο main εκτυπώνει 5, 5

c)

Παρατήρηση για Στοίβα Εκτέλεσης

Η στοίβα εκτέλεσης (run time stack) είναι μνήμη που αποθηκεύει τα μπλοκ των υποπρογραμμάτων, τα οποία ονομάζονται activation records-AR, ακριβώς με τη σειρά που εκτελούνται. Στο κάθε μπλοκ υποπρογράμματος βάζουμε όλες τις μεταβλητές του υποπρογράμματος ακριβώς με τη σειρά που δηλώνονται στο μπλοκ αυτό. Στη στοίβα εκτέλεσης τοποθετούμε στο τέλος και τους στατικούς δείκτες που δείχνουν σε μεταβλητές που χρησιμοποιούνται σε ένα μπλοκ υποπρογράμματος αλλά δεν δηλώνονται σε αυτό.

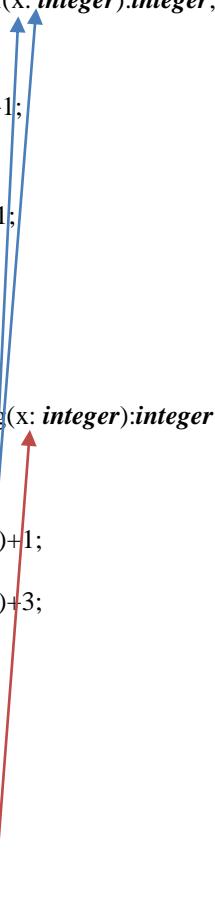
Η στοίβα εκτέλεσης για τη συγκεκριμένη άσκηση μέχρι και την τελευταία αναδρομή είναι η ακόλουθη:



1.4 Θέμα 1 Ιούνιος 2009

Δίνεται το ακόλουθο πρόγραμμα:

```
program MAIN;  
  
var  
  
y, z: integer;  
  
function f(x: integer):integer;  
begin  
    x:=x+1;  
    y:=x  
    x:=x+1;  
    f:=y;  
end;  
  
function g(x: integer):integer;  
begin  
    y:=f(x)+1;  
    x:=f(y)+3;  
    g:=x;  
end;  
  
begin  
    y:=7;  
    z:=g(y);  
    write (y, z);  
end.
```



Να βρείτε ποιες τιμές τυπώνει το πρόγραμμα αν εφαρμόσουμε:

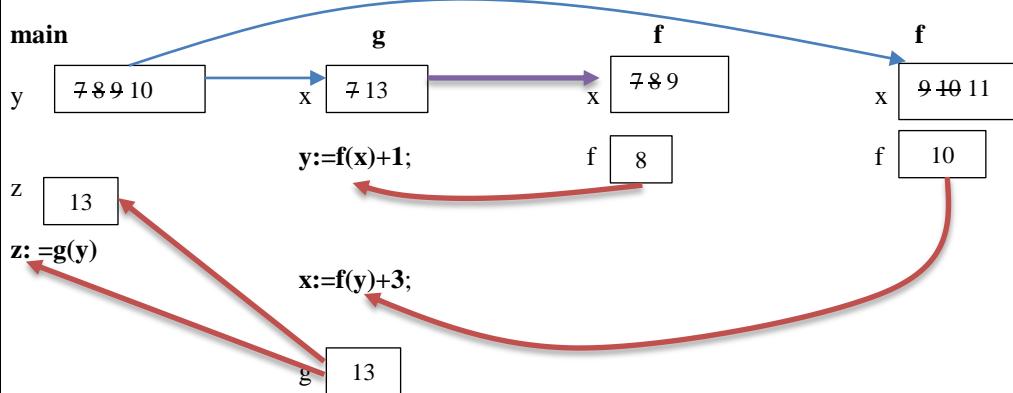
- a) call by value
- b) call by value-result
- c) call by reference

Απάντηση

Θεωρία για υποπρογράμματα στην Pascal

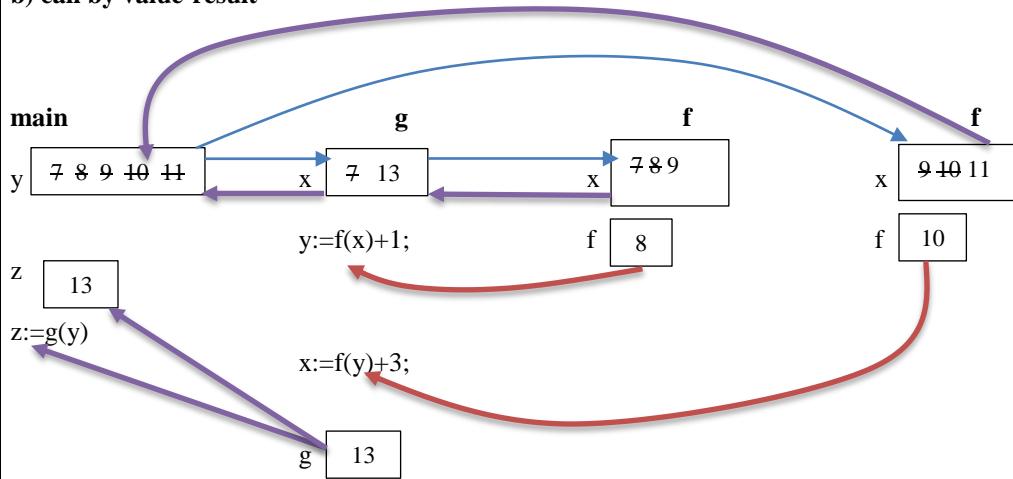
Στην Pascal υπάρχουν 2 τύποι υποπρογραμμάτων: οι procedures και οι functions. Οι procedures μπορούν να επιστρέφουν αποτελέσματα μέσω των παραμέτρων τους αν ο τρόπος κλήσης τους είναι call by value-result. Το όνομα μιας function έχει διπλό ρόλο: είναι και όνομα υποπρογράμματος αλλά ταυτόχρονα είναι και όνομα μεταβλητής στο οποίο καταχωρείται το τελικό αποτέλεσμα που επιστρέφει η function (δηλ. μια function επιστρέφει πάντα μόνο ένα αποτέλεσμα μέσω του ονόματος της).

a) call by value



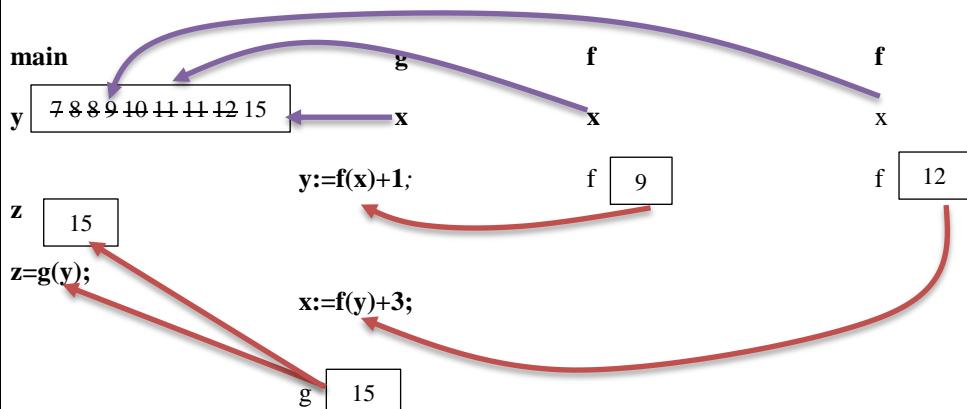
Στο call by value τυπώνονται οι τιμές 10, 13 στο main στην εντολή write(y, z)

b) call by value-result



Στο call by value-result τυπώνονται οι τιμές 13, 13 στο main στην εντολή write(y, z)

c) call by reference



Στο call by value-reference τυπώνονται οι τιμές 15, 15 στο main στην εντολή write(y, z)

1.5 Θέμα 4 Σεπτέμβριος 2012

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί στατικό κανόνα εμβέλειας και η μεταβίβαση παραμέτρων γίνεται με κλήση με τιμή.

```
program MAIN;  
var x, y: integer;  
  
function f(x: integer): integer;  
begin  
  if x=0 then y:=1 else y:=f(x-1)*y+1;  
  f:=y+1;  
end;
```

BEGIN

```
y:=4;  
x:=f(3);  
writeln(x, y);
```

END.

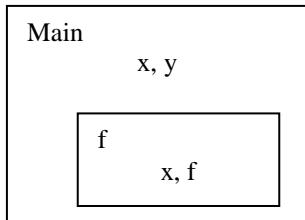
(a) Υποθέτοντας ότι χρησιμοποιείται στατικός κανόνας εμβέλειας να ορίσετε τα περιβάλλοντα αναφοράς (τοπικό, μη-τοπικό και καθολικό) όλων των τμημάτων του προγράμματος

(b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (Activation Records-AR) με τις τιμές των τοπικών μεταβλητών για κάθε AR μόλις έχει γίνει η κλήση της function f για τελευταία φορά

(c) Ποιες τιμές θα τυπωθούν στο τέλος για τα x και y; Παρουσιάστε την εξέλιξη και τις αλλαγές τιμών των μεταβλητών κατά τη διαδικασία υπολογισμού

Απάντηση

(a)



Περιβάλλοντα Αναφοράς

Main

Τοπικό: x, y, f (κλήση)

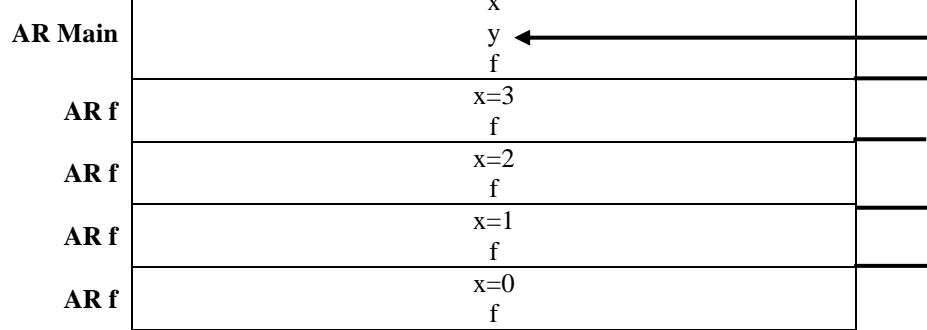
Μη-Τοπικό και Καθολικό: —

f

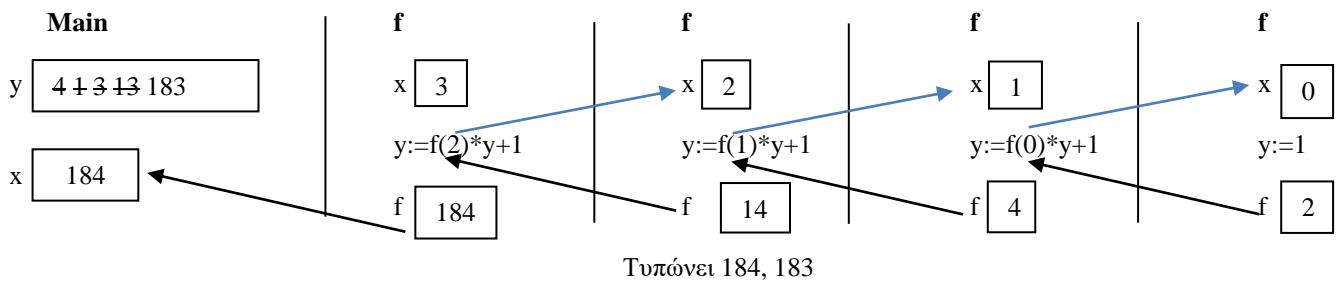
Τοπικό: x, f (τιμή)

Μη-Τοπικό και Καθολικό: y, f (κλήση)

(b)



(c)



1.6 Θέμα 1 Ατυπη Νοέμβριος 2011

Δίνεται ο ακόλουθος κώδικας σε μια γλώσσα τύπου C:

```
int a;
void P(int x, int y)
{
    ++x;
    ++y;
}
void main()
{
    a=4;
    P(a, a);
    print(a);
}
```

- (a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη τοπικά καθολικά) όλων των τμημάτων του προγράμματος;
- (b) Παρουσιάστε τη στοίβα εκτέλεσης (Run-time Stack) των εγγραφών ενεργοποίησης (Activation Records-AR) μόλις έχει γίνει η κλήση της P
- (c) Ποια τιμή θα τυπωθεί στο τέλος για το a αν η μεταβίβαση παραμέτρων γίνει με:
 - (i) Κλήση με τιμή (call by value)
 - (ii) Κλήση με αναφορά (call by reference)
 - (iii) Κλήση με τιμή-αποτέλεσμα (call by value-result)

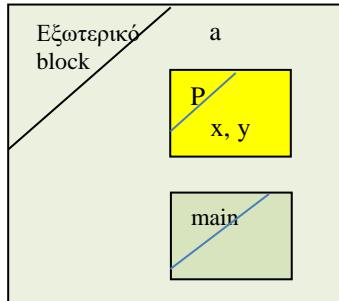
Σε ποια από τις παραπάνω εμφανίζεται **το φαινόμενο της ψευδωνυμίας**, σε ποιο τμήμα του προγράμματος και με ποιο τρόπο;

Απάντηση

(a)

Main: Τοπικό: —

Μη τοπικό-Καθολικό: a, main, P



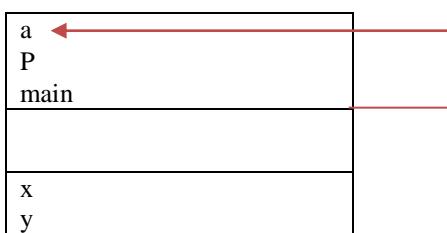
P: Τοπικό: x, y

Μη τοπικό-Καθολικό: a, main, P

(b)

Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)

AR εξωτερικού block

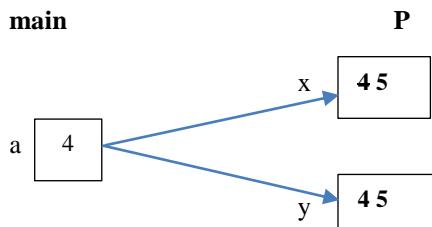


Η στοίβα εκτέλεσης βασίζεται αποκλειστικά στο call-by-value

Παρατήρηση: Στη στοίβα εκτέλεσης σε κώδικα C το 1^o AR που μπαίνει στη στοίβα είναι το AR του εξωτερικού μπλοκ

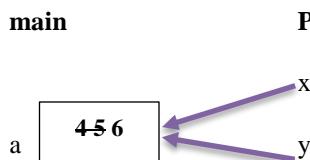
c)

(i) Κλήση με Τιμή (call by value)



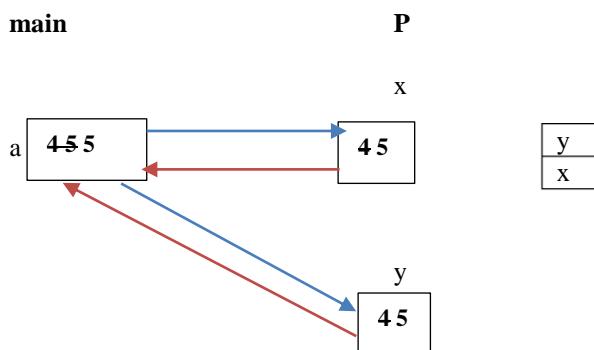
Τυπώνεται 4 στο main

(ii) Κλήση με Αναφορά (call by reference)



Τυπώνεται 6 στο main

(iii) Κλήση με Τιμή-Αποτέλεσμα (call by value-result)



Τυπώνεται 5 στο main

Παρατήρηση 1

Οι παράμετροι των υποπρογραμμάτων μπαίνουν σε μια στοίβα βάσει της σειράς που γράφονται δηλ.

y
x

Άρα όταν τελειώσει η P πρώτα γυρίζει το y και μετά γυρίζει το x .

Παρατήρηση 2

Το φαινόμενο της ψευδωνυμίας εμφανίζεται μόνο στην κλήση με αναφορά και συγκεκριμένα εμφανίζεται όταν μεταβιβάζεται η τι-
μή της παραμέτρου α ταυτόχρονα στους δείκτες x και y (δηλ. όταν δύο ή περισσότεροι δείκτες δείχνουν στην ίδια θέση μνήμης αυ-
τό ονομάζεται φαινόμενο ψευδωνυμίας). Μπορούμε να τροποποιήσουμε την τιμή της μεταβλητής με οποιοδήποτε δείκτη.

1.7 Θέμα 3 Ιούνιος 2012 και Θέμα 2 Ιούνιος 2015

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

```

program MAIN;
var      i:integer;
         a:real;

function S(i, Lo, Hi: integer; term: real):real;
var Temp: real;
begin
  Temp:=0;
  for i:=Lo to Hi do Temp:=Temp + term;
  S:=Temp;
end;
BEGIN
  i:=1;
  a:=S(i, 1, 100, 1/i);
  write(a);
END.
```

Τι υπολογίζει εδώ η συνάρτηση S και τι τυπώνεται στο τέλος με την εντολή `write(a)` στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων:

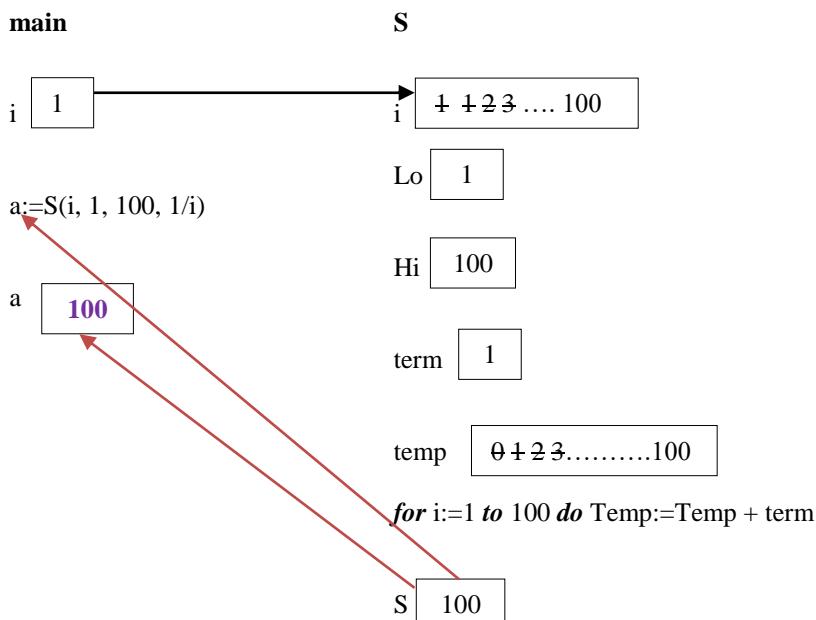
(a) Κλήση με τιμή (call by value)

(b) Κλήση με όνομα (call by name)

Δώστε το αποτέλεσμα στην περίπτωση (a) με αριθμό και στην περίπτωση (b) με μαθηματικό τύπο. Εξηγήστε σύντομα τις διαδικασίες υπολογισμού και στις δύο περιπτώσεις:

Απάντηση

(a) Κλήση με τιμή (call by value)



Analytiká

Εκτελείται η ακόλουθη επανάληψη:

i=1 temp=0+1

i=2 temp=1+1

i=3 temp=1+1+1

i=100 temp=1+1+1+....+1=100

Τι κάνει η function

Υπολογίζεται το άθροισμα $1+1+1+\dots+1$ (100 φορές) = 100. Στο τέλος καταχωρείται το αποτέλεσμα που είναι 100 στο όνομα S της function που είναι μεταβλητή μέσω της οποίας η function επιστρέφει το αποτέλεσμα της στο main. Άρα στη μεταβλητή a του main επιστρέφεται και καταχωρείται η τιμή 100 όπως φαίνεται στο επόμενο σχήμα:

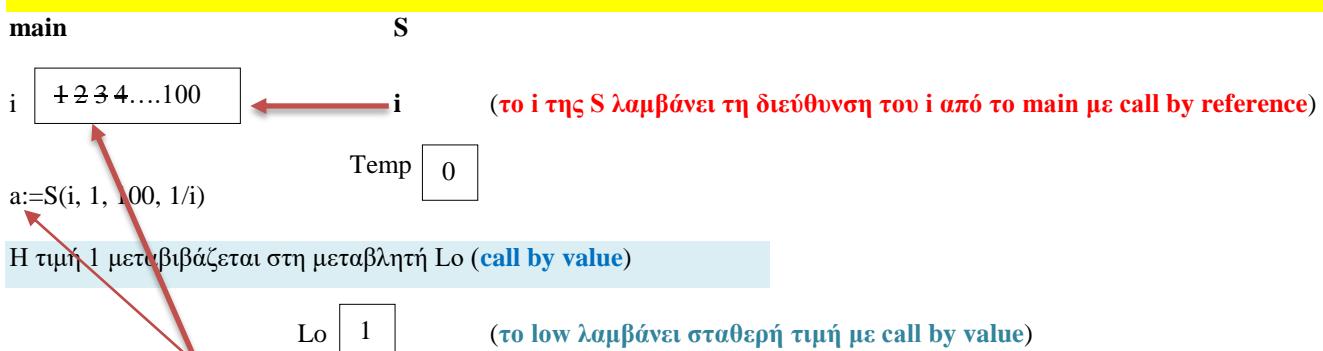


Στο τέλος τυπώνεται η τιμή 100 για το a.

(b) Κλήση με όνομα (call by name)

Στην κλήση της S έχουμε:

Η διεύθυνση της `i` από το `main` μεταβιβάζεται στο δείκτη `i` της `function S` (**call by reference**) διότι το `i` από το `main` είναι απλή μεταβλητή



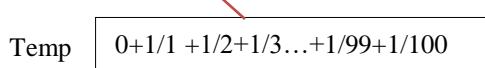
Η τιμή 100 μεταβιβίζεται στη μεταβλητή `Hi` (**call by value**)



Η διεύθυνση του i στο 1/i μεταβιβάζεται στη μεταβλητή term (call by reference)

Term (το Term λαμβάνει τη διεύθυνση του ι από το main με call by reference)

~~for i:=1 to 100 do Temp:=Temp + term~~



$$a \quad \boxed{1+1/2+1/3\dots+1/100}$$

S 1/1+1/2+1/3 ±1/100

Στο τέλος της πρώτης σειράς οι παραπάνω στασηί $1/1 + 1/2 + 1/3 + \dots + 1/100$ για το α.

Πιο Αναλυτικά

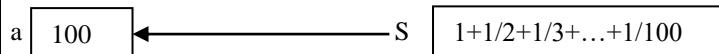
i=1 temp=0+1/1
i=2 temp=1/1+1/2
i=3 temp=1/1+1/2+1/3

.....

i=100 temp=1+1/2+1/3+...+1/100

Υπολογίζεται το άθροισμα **1+1/2+1/3+...+1/100**

Στο τέλος καταχωρείται το αποτέλεσμα που είναι η παράσταση $1/1+1/2+1/3+\dots+1/100$ στο όνομα S της function που είναι μεταβλητή και επιστρέφεται μέσω αυτής στη μεταβλητή a του main όπως φαίνεται στο επόμενο σχήμα:



Παρατήρηση

- Όταν στο **call by name** μεταβιβάζουμε **σταθερές τιμές** όπως το 1 και 100, αυτό ταυτίζεται με το **call by value**
- Όταν στο **call by name** μεταβιβάζουμε **μεταβλητές** όπως το i, αυτό ταυτίζεται με το **call by reference**
- Όταν στο **call by name** μεταβιβάζουμε αριθμητικές παραστάσεις, αυτό ταυτίζεται με το **call by value**
- **Στο παράδειγμα μας η μεταβίβαση του το 1/i είναι call-by-reference στο i**

1.8 Ασκηση 1 από Φροντιστήριο Ιούνιος 2022

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο **στατικός κανόνας εμβέλειας**:

```

program MAIN;
var i, a: integer;
function F(j, b, c, d: integer): integer;
var e: integer;
begin
e:= 1;
for j:= b to c do e:= e*d;
F:= e
end;
BEGIN
i:= 1;
e:= i;
a:=F(i, 1, 10, i);
write(a)
END

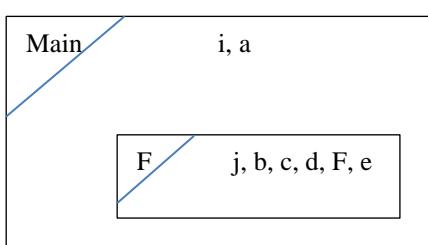
```

- α) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- β) Στον παραπάνω κώδικα υπάρχει μία εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error. Ποια είναι αυτή και γιατί; "Σβήστε" την εντολή αυτή από τον κώδικα.
- γ) Τι υπολογίζει η συνάρτηση F και τι τυπώνεται στο τέλος με την εντολή **write(a)**, 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)

Δώστε το αποτέλεσμά σας είτε με αριθμό, είτε με μαθηματικό τύπο

Απάντηση

a)



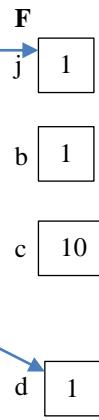
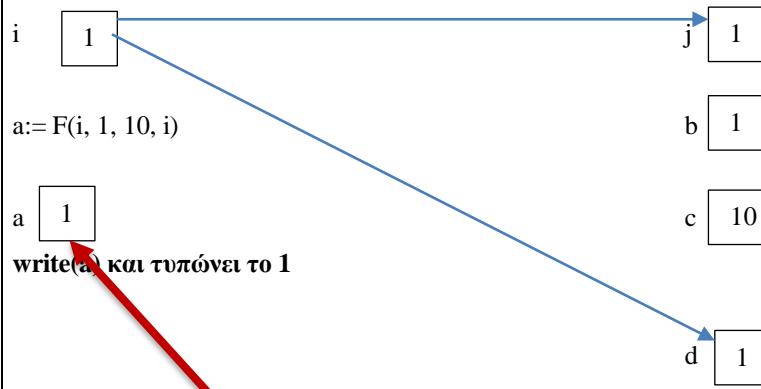
Main: Τοπικό: i, a, F (κλήση) Μη Τοπικό και Καθολικό: –

F: Τοπικό: j, b, c, d, F (τιμή), e Μη Τοπικό και Καθολικό: i, a, F (κλήση)

- β) Η εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error είναι η **e:= i**; διότι η μεταβλητή e δεν δηλώνεται στο main

γ) i) Call by value

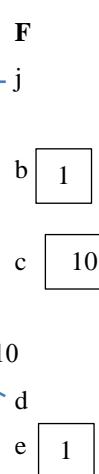
main



```

for j:=1 to 10 do e:= e*d
j=1, e=1*1=1
j=2, e=1*1=1
j=3, e=1*1=1
.....
j=10, e=1*1=1
F:=1

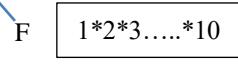
```



```

for j:=1 to 10 do e:= e*d
j=1, e=1*1
j=2, e=1*2
j=3, e=1*2*3
.....
j=10, e=1*2*3*....*10

```



1.9 Θέμα 5 Ιούνιος 2012

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal όπου η κλήση των παραμέτρων γίνεται με αναφορά (call by reference).

```
program MAIN;
  var x, y, z:integer;
  function A(w: integer): integer;
    begin
      y:=10;
      A:=x*w;
    end;
  function B(x: integer): integer;
    var y:integer;
    begin
      y:=13;
      B:=A(y);
    end;
BEGIN
  x=3; y:=2;
  z:=B(y);
  write(z); write(y)
END.
```

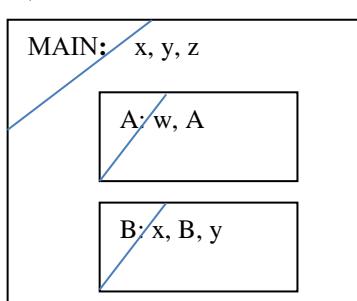
i)

- a) Θεωρώντας ότι ισχύει ο **στατικός κανόνας εμβέλειας** ποια τα περιβάλλοντα του προγράμματος (τοπικά, μη τοπικά και καθολικά);
 - b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (AR) μόλις έχει γίνει η κλήση της A
 - c) Ποιες τιμές θα τυπωθούν στο τέλος για τα z και y
- ii) Απαντήστε στα παραπάνω ερωτήματα θεωρώντας ότι ισχύει ο **δυναμικός κανόνας εμβέλειας**.

Απάντηση

i)

a)



Περιβάλλοντα Αναφοράς

Main: Τοπικό: x, y, z, A (κλήση), B (κλήση)

Μη-τοπικό και Καθολικό:-

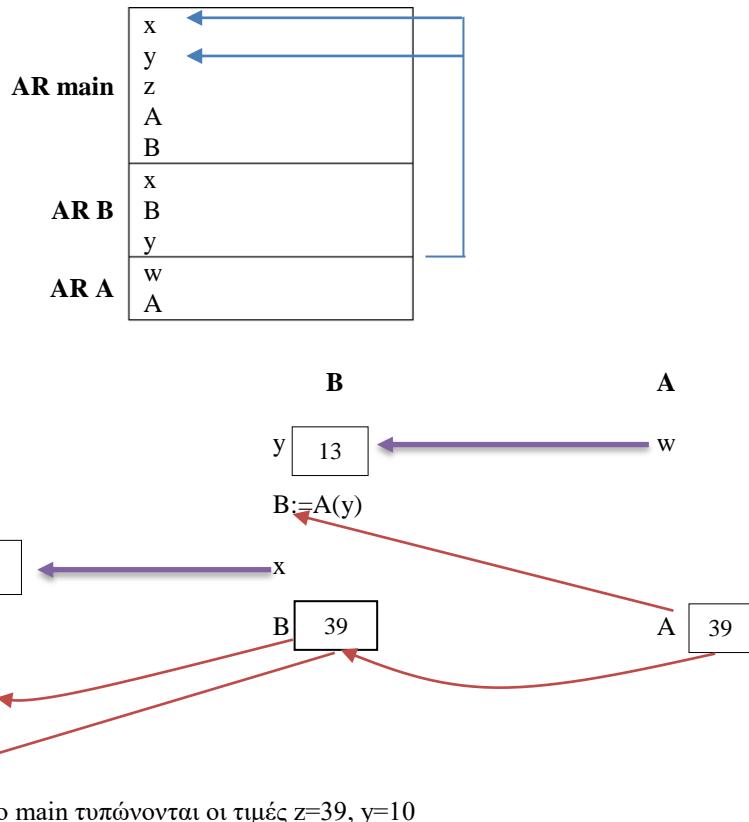
Function A: Τοπικό: w, A (τιμή)

Μη-τοπικό και Καθολικό: x, y, z, A (κλήση), B (κλήση)

Function B: Τοπικό: x, y, B (τιμή)

Μη-τοπικό και Καθολικό: z, A (κλήση), B (κλήση)

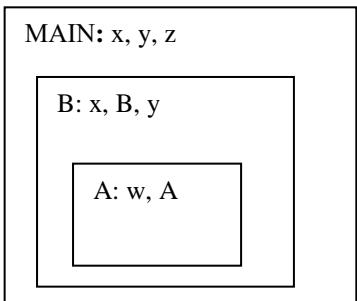
β) Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)



Τελικά στο main τυπώνονται οι τιμές z=39, y=10

ii)

α)



To Block της function B μπαίνει μέσα στη main διότι η B καλείται από το main.

To Block της function A μπαίνει μέσα στη B διότι η A καλείται από το B.

Περιβάλλοντα Αναφοράς

Main: Τοπικό: x, y, z, A B

Μη-τοπικό και καθολικό: –

Function A: Τοπικό: w, A

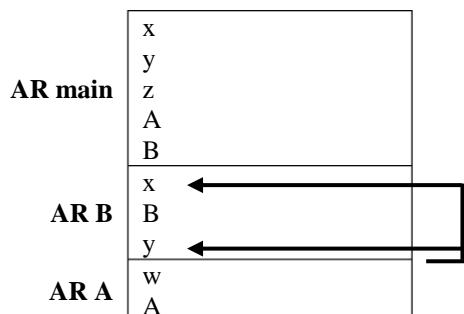
Μη-τοπικό: x, y, B (από B), z, A, B (κλήση) (από main)

Καθολικό: A (κλήση), B (κλήση)

Function B: Τοπικό: x, y, B

Μη-τοπικό και Καθολικό: z, A, B (υποπρογράμματα)

β) Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)



γ)

main

x 3

B

y 10

A

A 20

y 2

x

B 20

z 20

Tελικά στο main τυπώνονται οι τιμές z=20, y=2

1.10 Θέμα 3- Ιούνιος 2019 -Ομάδα Α

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας;

program MAIN;

var i, a: **integer**;

function F(j, b, c, d: **integer**): **integer**;

var e: **integer**;

begin

 e:= 1;

for j:= b **to** c **do** e:= e*b;

 F:= e

end;

BEGIN

 i:= 1;

 c:= i;

 a:= F(i, 1, 10, i);

write (a)

END.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη – τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

(b) Στον παραπάνω κώδικα υπάρχει **μία εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error**. Ποια είναι αυτή και γιατί; Σβήστε την εντολή αυτή από τον κώδικα.

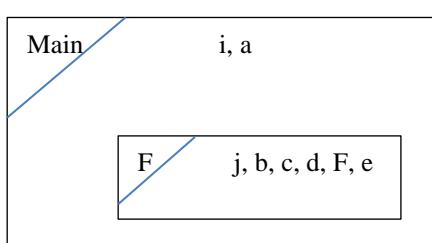
(c) Τι υπολογίζει η συνάρτηση F και τυπώνεται στο τέλος με την εντολή write (a), στις περιπτώσεις μεταβίβασης παραμέτρων (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών υπολογισμού);

1. Κλήση με τιμή (call by value)
2. Κλήση με αναφορά (call by reference)
3. Κλήση με τιμή – αποτέλεσμα (call by value – result)
4. Κλήση με όνομα (call by name)

Δώστε το αποτέλεσμά σας είτε με αριθμό είτε με μαθηματικό τύπο.

Απάντηση

(a)



Main: Τοπικό: i, a, F (**κλήση**)

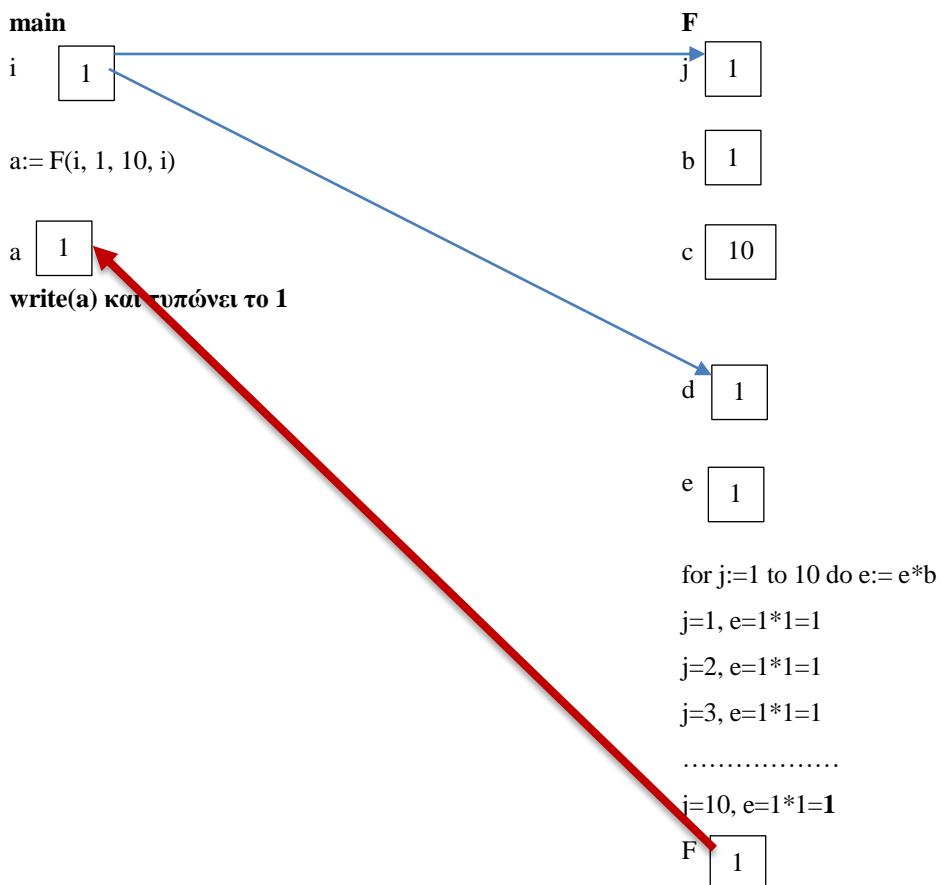
Μη Τοπικό και Καθολικό: –

F: Τοπικό: j, b, c, d, F (**τιμή**), e

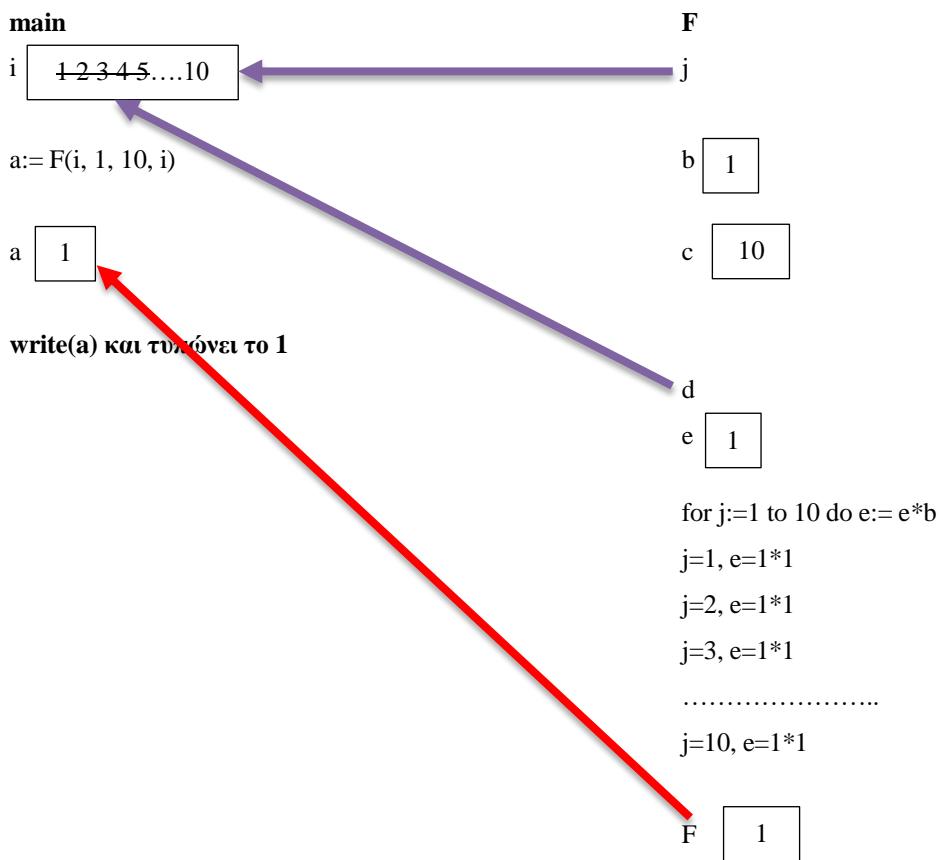
Μη Τοπικό και Καθολικό: i, a, F (**κλήση**)

(b) Η εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error είναι η $c := i$; διότι η μεταβλητή c δεν δηλώνεται στο main

c) 1. Call by value

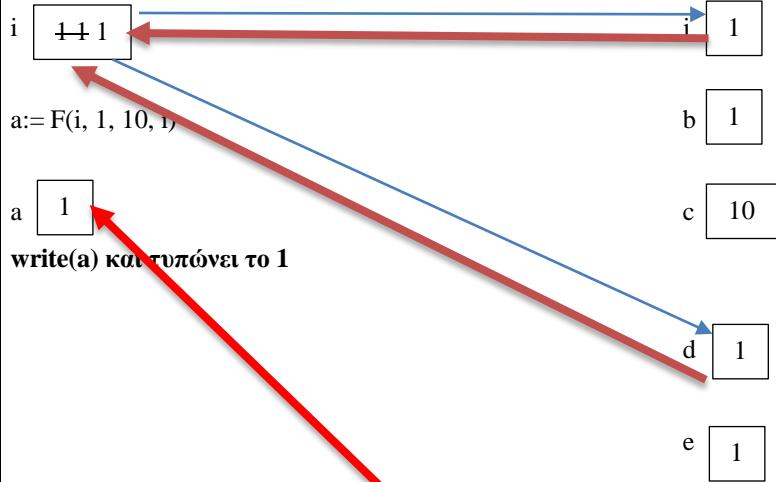


c) 2. Call by reference



c) 3. Call by value-result

main



for $j := 1$ to 10 do $e := e * b$

$j = 1, e = 1 * 1 = 1$

$j = 2, e = 1 * 1 = 1$

$j = 3, e = 1 * 1 = 1$

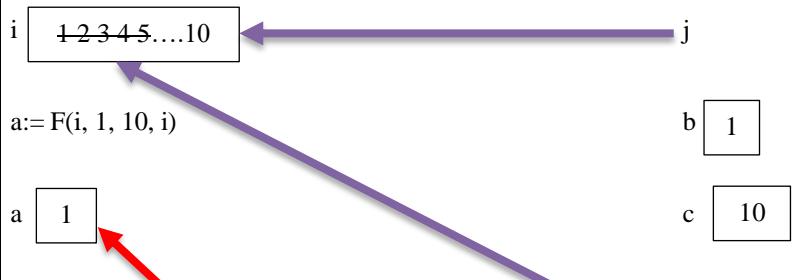
.....

$j = 10, e = 1 * 1 = 1$



d) 3. Call by name

main



for $j := 1$ to 10 do $e := e * b$

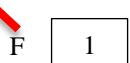
$j = 1, e = 1 * 1$

$j = 2, e = 1 * 1$

$j = 3, e = 1 * 1$

.....

$j = 10, e = 1 * 1$



1.11 Θέμα 1 Σεπτέμβριος 2010 και Θέμα 1 Σεπτέμβριος 2020

Λύστε το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

program MAIN;

 procedure X;

 a: integer;

 procedure Y;

 begin

 a:=a*2;

 write("In Y:", a)

 end;

 procedure Z;

 a: integer;

 begin

 a:=2;

 Y;

 write("In Z:", a)

 end;

 begin

 a:=3;

 Z

 end;

BEGIN

 X;

END.

I. Θεωρώντας ότι ισχύει ο **στατικός κανόνας εμβέλειας**:

(α) Ποια είναι τα **περιβάλλοντα αναφοράς** (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

(β) Παρουσιάστε τη **στοίβα εκτέλεσης** (run-time stack) των εγγραφών ενεργοποίησης (Activation Records – AR) μόλις έχει γίνει η κλήση της Y.

(γ) **Ποια τιμή θα τυπωθεί** με την εντολή **write{"In Y:" , a}** της procedure Y και **ποια τιμή** με την εντολή **write {"In Z:", a}** της procedure Z;

II. Απαντήστε στα παραπάνω ερωτήματα, θεωρώντας ότι ισχύει ο **δυναμικός κανόνας εμβέλειας**. Εξηγείστε σύντομα τις διαφορές που παρατηρείτε.

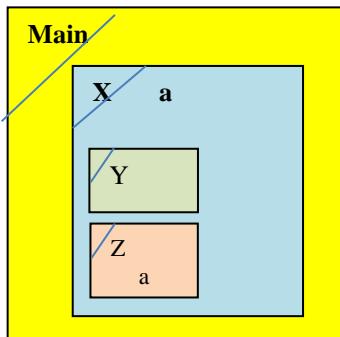
Απάντηση

I.

(a)

Περιβάλλοντα Αναφοράς με στατικό κανόνα Εμβέλειας

Στο **στατικό κανόνα** εμβέλειας βάζουμε τα blocks των υποπρογραμμάτων σύμφωνα με τη σειρά που τα συναντάμε στον κώδικα από πάνω προς τα κάτω



Main Τοπικό: X (κλήση)

Μη-Τοπικό και Καθολικό: —

X Τοπικό: a, Y (κλήση), Z (κλήση)

Μη-Τοπικό και Καθολικό: X (κλήση)

Y Τοπικό: —

Μη-Τοπικό: a, Y (κλήση), Z (κλήση) (*από X*) X (κλήση) (*από main*)

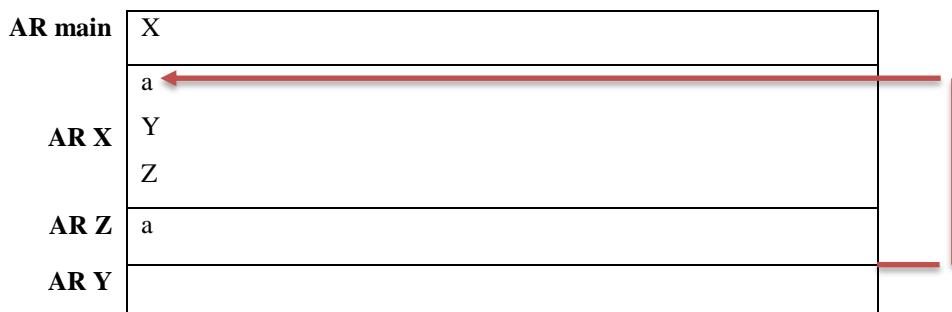
Καθολικό: X (κλήση)

Z Τοπικό: a

Μη-Τοπικό: Y (κλήση), Z (κλήση) (*από X*) X (κλήση) (*από main*)

Καθολικό: X (κλήση)

(b) Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)



(c) Στατικός Κανόνας Εμβέλειας

Main X Z Y

a [3 6] a [2]

Τυπώνει

In Z: 2

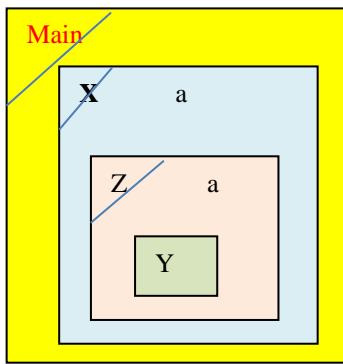
In Y: 6

II.

(a)

Περιβάλλοντα Αναφοράς με Δυναμικό κανόνα Εμβέλειας

Στο **δυναμικό κανόνα** εμβέλειας βάζουμε τα blocks των υποπρογραμμάτων σύμφωνά με τη σειρά εκτέλεσης. Εδώ π.χ. το main καλεί την procedure X. Η procedure X καθώς εκτελείται καλεί την procedure Z. Η procedure Z καθώς εκτελείται καλεί την procedure Y.



Για να φτιάξουμε τα περιβάλλοντα αναφοράς στο δυναμικό κανόνα εμβέλειας γράφουμε τις δηλώσεις μέσα στο κάθε υποπρόγραμμα από το στατικό κανόνα εμβέλειας. Το σχήμα του δυναμικού κανόνα το κοιτάμε μόνο για τα περιβάλλοντα blocks

Main Τοπικό: X (κλήση)

Μη-Τοπικό και Καθολικό: —

X Τοπικό: a, Y (κλήση), Z (κλήση)

Μη-Τοπικό και Καθολικό: X (κλήση)

Z Τοπικό: a

Μη-Τοπικό: Y (κλήση), Z (κλήση) (*από X*), X (κλήση) (*από main*)

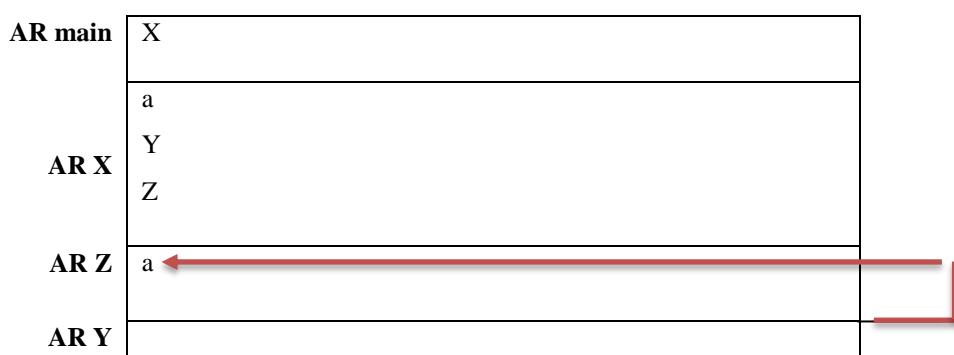
Καθολικό: X (κλήση)

Y Τοπικό: —

Μη-Τοπικό: a (*από Z*), Y (κλήση), Z (κλήση) (*από X*), X (κλήση) (*από main*)

Καθολικό: X (κλήση)

(b) Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)



(c) Δυναμικός Κανόνας Εμβέλειας

Main	X	Z	Y
	a [3]	a [2 4]	
Τυπώνει	In Z: 4	In Y: 4	

1.12 Θέμα 2 Σεπτέμβριος 2010

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί το στατικό κανόνα εμβέλειας:

program MAIN;

var y, z: **integer**;

function f (a: **integer**): **integer**;

begin

 y:= a+1;

 f:= y+a;

end;

function g (x: **integer**): **integer**;

begin

 y:= f(x+1)+1;

 z:= f(y-x);

 g:= z+1;

end;

BEGIN

 y:= 5 ;

 z:= g(2*y);

write (y, z);

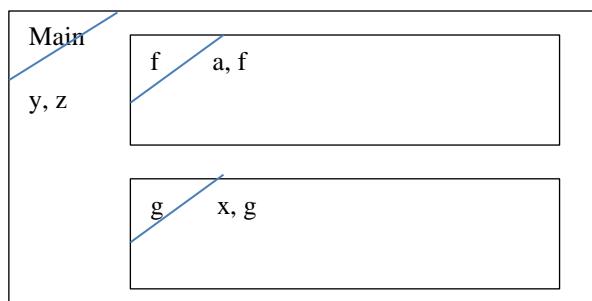
END.

Ποιες τιμές θα τυπωθούν στο τέλος για τα y και z, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων; (παρουσιάστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού, στις δύο περιπτώσεις)

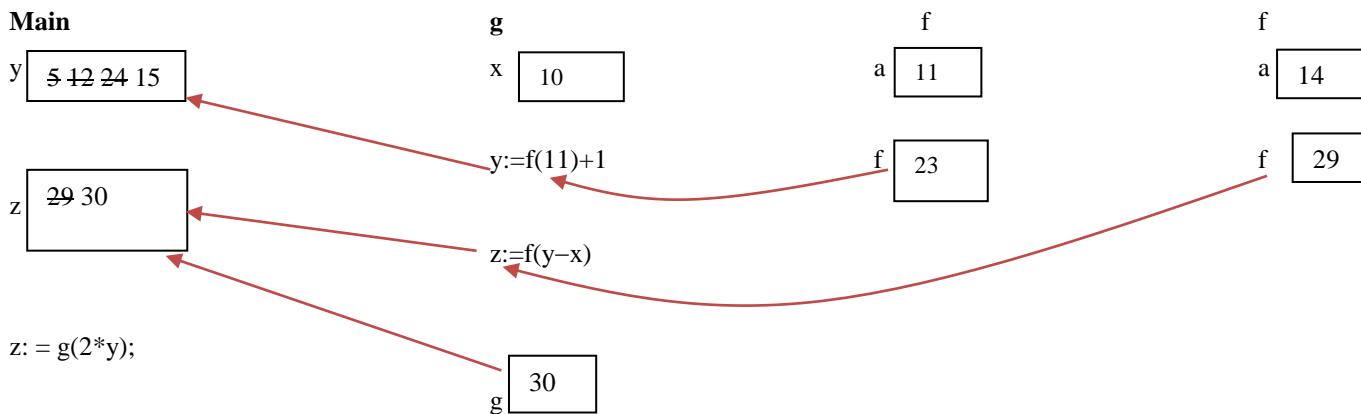
a) Κλήση με τιμή (call by value)

β) Κλήση με όνομα (call by name)

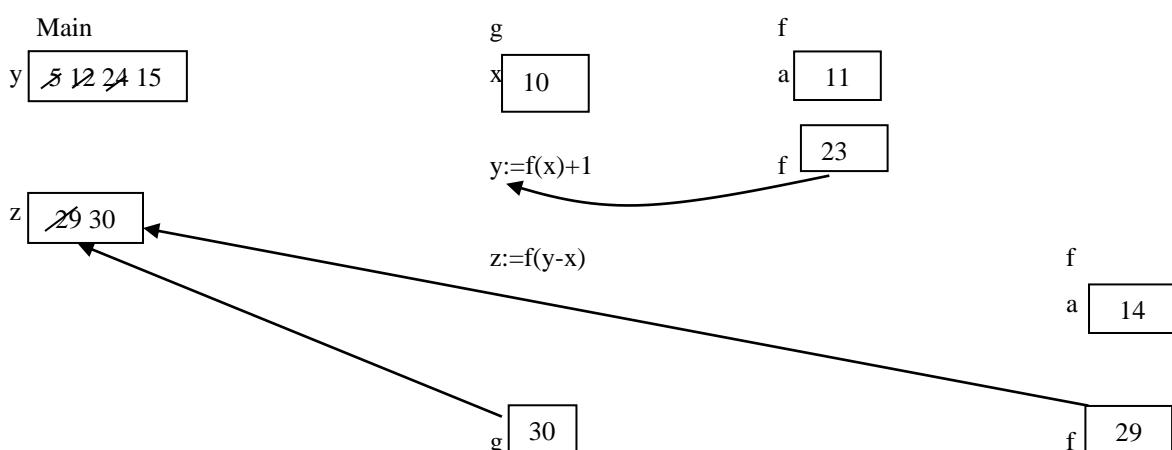
Απάντηση



(α) Κλήση με Τιμή (call by value)



(β) Κλήση με Όνομα (call by name)



Παρατήρηση

Παρατηρούμε ότι και στις 3 περιπτώσεις που καλούνται υποπρογράμματα δηλ:

- ✓ g(2*y)
- ✓ y := f(x+1)+1
- ✓ z := f(y-x)

έχουμε εκφράσεις μέσα στις παρενθέσεις που πρώτα υπολογίζονται και μετά μεταβιβάζεται ως τιμή το αποτέλεσμα τους, άρα το call by name ταυτίζεται στην περίπτωση αυτή με το call by value.

1.13 Θέμα 2 Σεπτέμβριος 2011 και Σεπτέμβριος 2017 και Σεπτέμβριος 2022

program MAIN;

var i, a:integer;

function P(j: integer):integer;

function Q(k: integer):integer;

begin

if k<4

then i:=P(k);

Q:=k;

else write(k);

Q:=0;

end;

begin

j:=j+j;

P:=Q(j);

end;

begin

i:=0;

a:=P(i+1);

write(i, a);

end.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

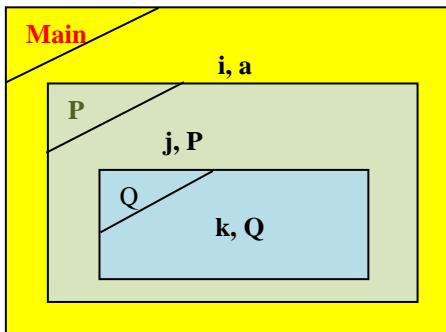
(b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (Activation Records-AR) μόλις έχει γίνει η τελευταία κλήση της Q

(c) Ποιες τιμές τυπώνονται από το πρόγραμμα;

(d) Τι διαφοροποιήσεις θα υπάρχουν στις απαντήσεις σας στα (a) –(c) αν χρησιμοποιείται δυναμικός κανόνας εμβέλειας;

Απάντηση

Στατικός Κανόνας Εμβέλειας



Main

Τοπικό Περιβάλλον: i, a, P (κλήση)

Μη τοπικό και Καθολικό: –

P

Τοπικό Περιβάλλον: j, P (τιμή), Q (κλήση)

Μη τοπικό και Καθολικό: i, a, P (κλήση)

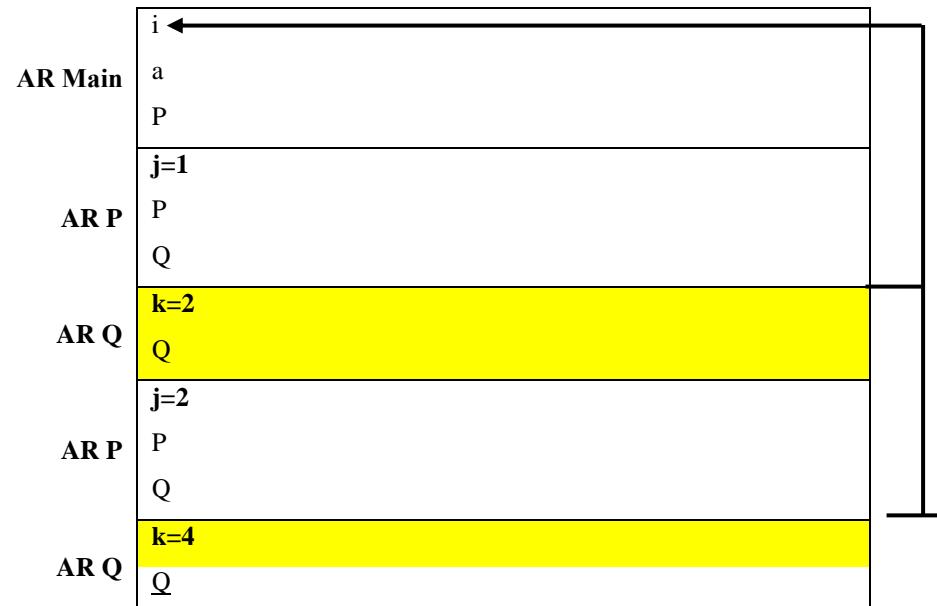
Q

Τοπικό Περιβάλλον: k, Q (τιμή)

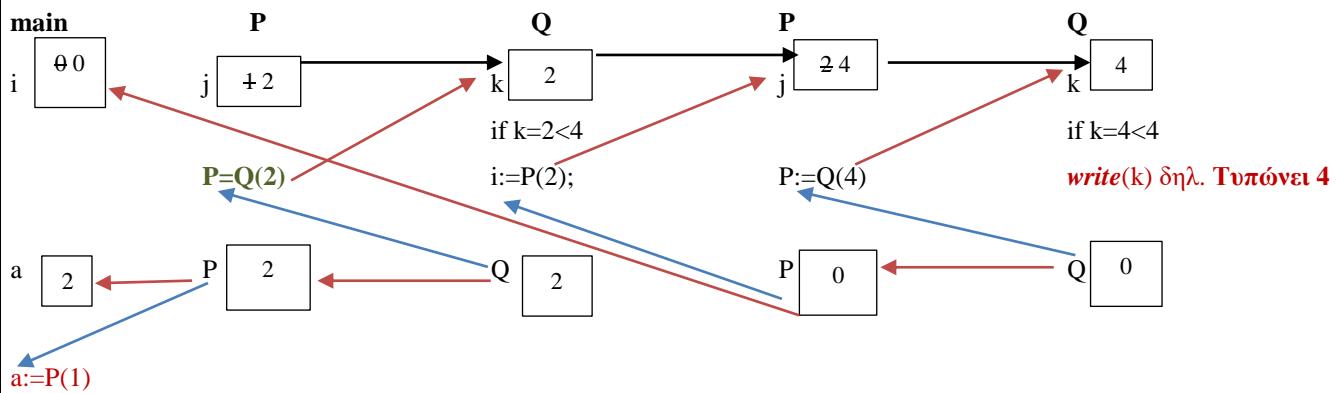
Μη τοπικό j, P (τιμή), Q (κλήση) (από P) και i, a, P (κλήση) (από Main)

Καθολικό: P (κλήση)

(b) Στοίβα Εκτέλεσης με Στατικό Κανόνα Εμβέλειας



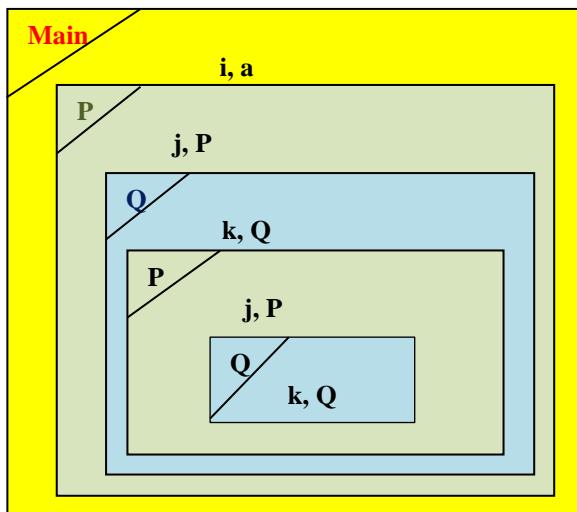
(c)



Το πρόγραμμα στο main τυπώνει 0, 2

Συνολικά το πρόγραμμα τυπώνει 4, 0, 2

Δυναμικός Κανόνας Εμβέλειας



Main

Τοπικό: i, a, P (κλήση)

Μη τοπικό και Καθολικό: –

P

Τοπικό: j, P (τιμή), Q (κλήση)

Μη τοπικό και Καθολικό: i, a, P (κλήση)

Q

Τοπικό: k, Q (τιμή)

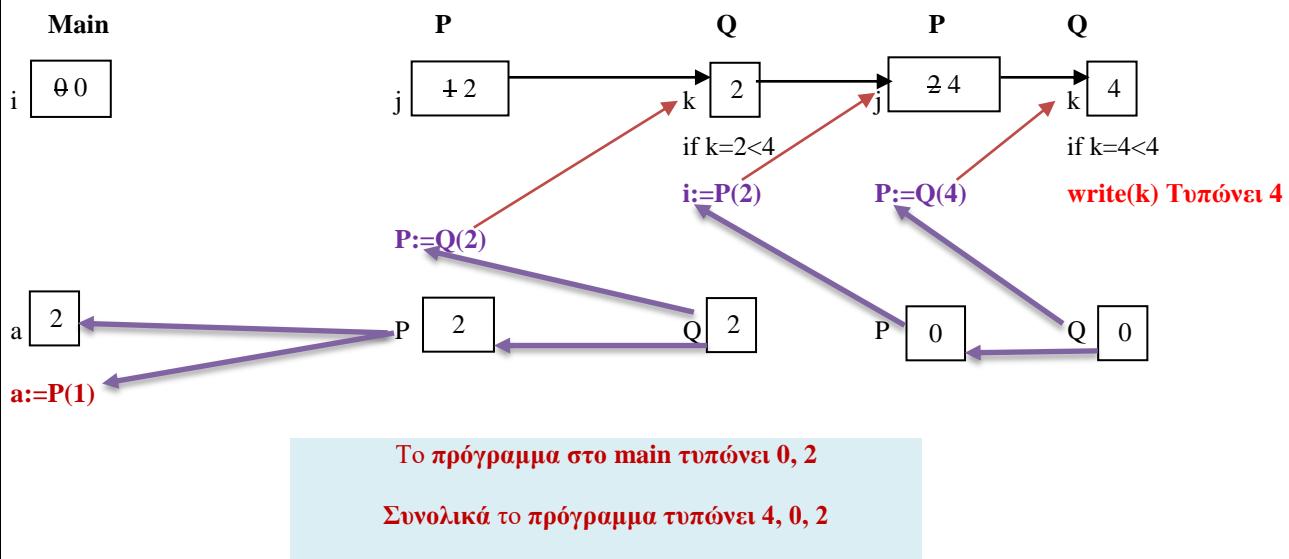
Μη τοπικό: j, P (τιμή), Q (κλήση) (από P) και i, a, P (κλήση) (από Main)

Καθολικό: P (κλήση)

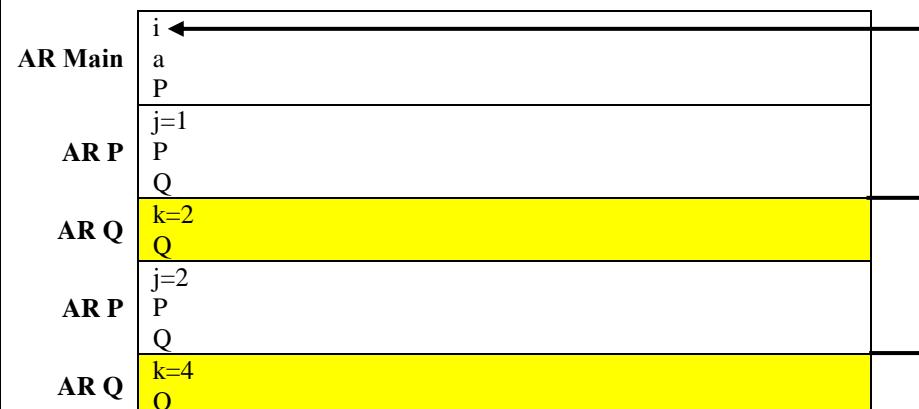
Παρατήρηση

Όταν στο δυναμικό κανόνα εμβέλειας ένα υποπρόγραμμα καλείται πολλές φορές και γράφεται το block του αντίστοιχα πολλές φορές, καθορίζουμε τα περιβάλλοντα αναφοράς του μόνο από την πρώτη φορά που τα συναντάμε.

β)



(b) Στοίβα Εκτέλεσης με Δυναμικό Κανόνα Εμβέλειας



1.14 Θέμα 2 Σεπτέμβριος 2014

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

```
program MAIN;  
var i: integer;  
    A: array [0..1] of integer;  
  
procedure P(x, y: integer);  
var z:integer;  
begin  
    z:=x;  
    x:=y;  
    i:=0;  
    y:=z ;  
end;  
  
BEGIN  
    i:=1;  
    A[0]:=0;  
    A[1]:=2;  
    P(i, A[i]);  
    write(i, A[0], A[1]);  
END.
```

(a) Ποια είναι τα περιβάλλοντα (τοπικό, μη-τοπικό και καθολικό) όλων των τμημάτων του προγράμματος;

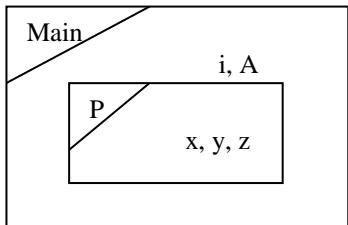
(b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (AR) μόλις έχει γίνει η κλήση της P

(c) Ποια είναι η έξοδος του προγράμματος κατά την εκτέλεση του στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγήστε τις αλλαγές 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)

Απάντηση

(a)



Περιβάλλοντα Αναφοράς

Main

Τοπικό: i, A, P (κλήση)

Μη-Τοπικό και Καθολικό: —

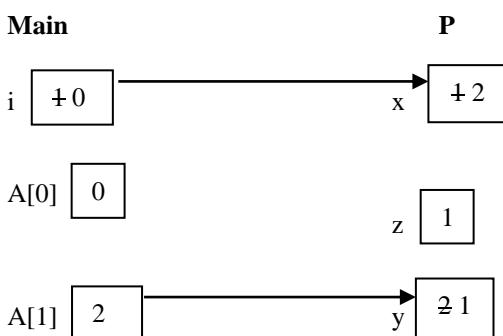
P

Τοπικό: x, y, z

Μη-Τοπικό και Καθολικό: i, A, P (κλήση)

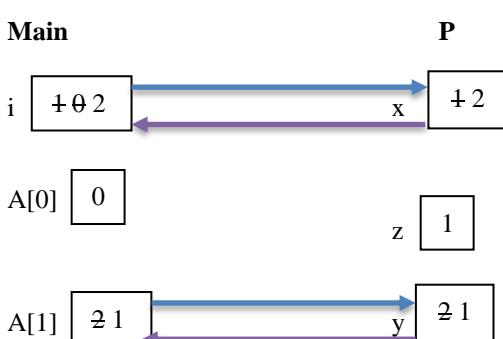
(c)

Call by value



Στο Call by value ΔΕΝ επιστρέφονται οι παράμετροι της P άρα τυπώνονται οι τιμές 0, 0, 2 στο main

Call by value-result



Στο Call by value-result επιστρέφονται οι παράμετροι της P άρα τυπώνονται οι τιμές 2, 0, 1 στο main

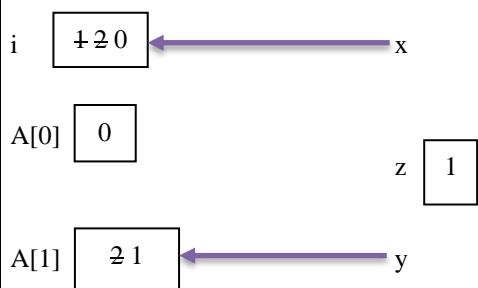
Παρατήρηση

Πρώτα επιστρέφεται το y και μετά το x

Call by reference

Main

P

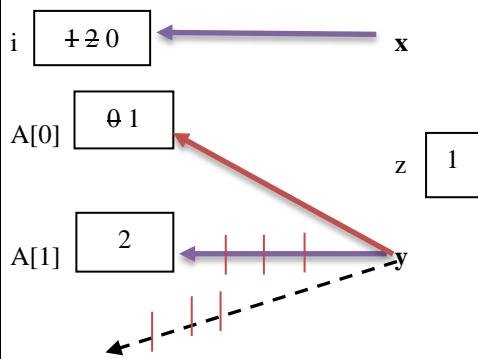


Στο Call by reference τυπώνονται οι τιμές 0, 0, 1 στο main

Call by name

Main

P



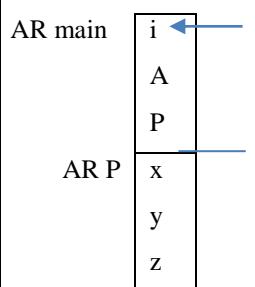
Αρχικά το y δείχνει στη θέση A[1] αλλά μετά όταν το i γίνεται 2 το y δείχνει στο A[2] (εκτός πίνακα) και μετά όταν το i γίνεται 0 το y δείχνει στο A[0]

Στο Call by name τυπώνονται οι τιμές 0, 1, 2 στο main

(b)

Στοίβα εκτέλεσης

Προσθέτουμε τα μπλοκ των υποπρογραμμάτων με τη σειρά κατά την οποία εκτελούνται. Επίσης βάζουμε στατικούς δείκτες στα μπλοκ των υποπρογραμμάτων που χρησιμοποιούν ξένες μεταβλητές και δείχνουν προς τις μεταβλητές που χρησιμοποιούν.



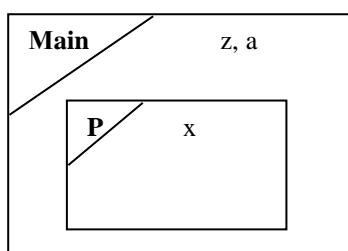
1.15 Θέμα 1 Σεπτέμβριος 2013

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί στατικό κανόνα εμβέλειας:

```
program MAIN;
var
  z: integer;
  a: array [1..2] of integer;
procedure P(x: integer);
begin
  a[1]:=5;
  z:=2;
  x:=x+7;
end;
begin
  a[1]:=2; a[2]:=3;
  z:=1;
  P(a[z]);
  write(a[1], a[2], z);
end.
```

- (a) Ποια είναι τα περιβάλλοντα (τοπικό, μη-τοπικό και καθολικό) όλων των τμημάτων του προγράμματος;
- (b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (AR) μόλις έχει γίνει η κλήση της P
- (c) Ποια είναι η έξοδος του προγράμματος κατά την εκτέλεση του στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγήστε τις αλλαγές 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)

Απάντηση



Main

Τοπικό: z, a, P

Μη Τοπικό και Καθολικό: -

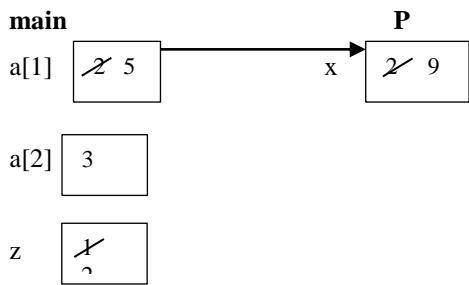
P

Τοπικό: x

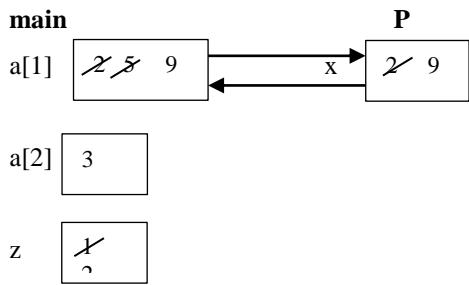
Μη Τοπικό και Καθολικό: z, a, P

c)

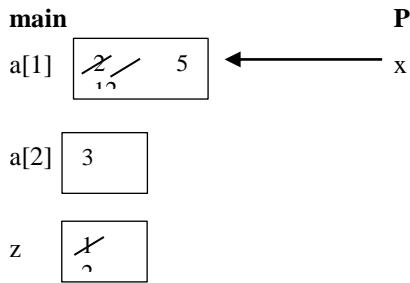
call by value → Όταν τερματίσει η συνάρτηση P η παράμετρος x δεν επιστρέφεται πίσω στο main. Οι τιμές στο main είναι: 5, 3, 2



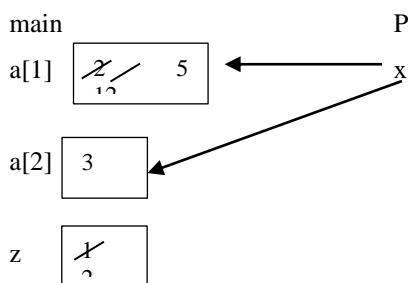
call by value result → Όταν τερματίσει η συνάρτηση P η παράμετρος x επιστρέφεται πίσω στο main στο στοιχείο a[1]. Οι τιμές που τυπώνονται είναι: 9, 3, 2



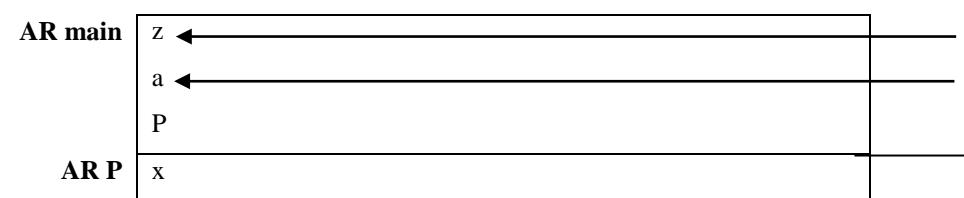
call by reference → Όταν τερματίσει η συνάρτηση P δεν επιστρέφεται τίποτα. Οι τιμές που τυπώνονται είναι: 12, 3, 2



call by name → Ο δείκτης δείχνει αρχικά στο a[1]. Όταν το z γίνει 2 τότε ο x δείχνει στο a[2]. Οι τιμές που τυπώνονται είναι: 5, 10, 2



b) Στοίβα Εκτέλεσης (Run-time Stack) με Εγγραφές Ενεργοποίησης (AR)



1.16 Θέμα 5 Ιούνιος 2008 και Φεβρουάριος 2017

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal

program MAIN;

var z:integer;

procedure p(x: integer)

begin

<BODY>

end;

BEGIN

z:=1;

p(z);

write(z);

END.

Γράψτε μέχρι 2 εντολές για το <BODY> τέτοιες ώστε να τυπωθούν διαφορετικές τιμές για το z (να τις αναφέρετε) αν η μεταβί-βαση παραμέτρων γίνεται με:

(i) Κλήση με τιμή (call by value)

(ii) Κλήση με αναφορά (call by reference)

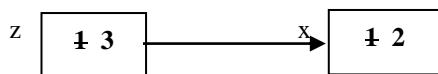
(iii) Κλήση με τιμή αποτέλεσμα (call by value-result)

Απάντηση

Οι 2 εντολές που βάζουμε στο <BODY> είναι οι εξής:

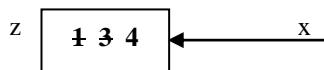
- z:=z+2;
- x:=x+1;

(i) Κλήση με τιμή (call by value)



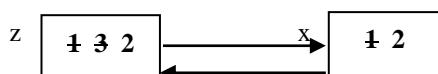
Στο main η εντολή **write(z);** **τυπώνει 3**

(ii) Κλήση με αναφορά (call by reference)



Στο main η εντολή **write(z);** **τυπώνει 4**

(iii) Κλήση με τιμή-αποτέλεσμα (call by value-result)



Στο main η εντολή **write(z);** **τυπώνει 2**

1.17 Φροντιστήριο 2007

```

program MAIN;
var g:integer;

procedure B(a: integer);
var x:integer;

procedure A(n: integer);
begin
    g:=n;
end;

procedure R(m: integer);
begin
    write(x);
    x:=x/2;
    if x>=1 then R(m+1) else A(m);
end;

begin
    x:=a*a;
    R(1);
end;

```

```

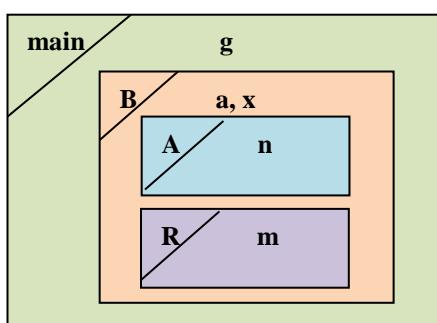
BEGIN
    B(3);
    write(g);
END.

```

- (α) Να βρεθούν τα περιβάλλοντα Αναφοράς όλων τμημάτων
- (β) Ποιες τιμές τυπώνονται;
- (γ) Η στοίβα εκτέλεσης (run-time stack) με τις εγγραφές ενεργοποίησης (AR) μόλις αρχίσει η εκτέλεση της A (και στατικοί δείκτες)

Απάντηση

(α)



Τα περιβάλλοντα Αναφοράς είναι τα ακόλουθα:

Main

Τοπικό: g, B (κλήση)

Μη-Τοπικό και Καθολικό: –

B

Τοπικό: a, x, A (κλήση), R (κλήση)

Μη-Τοπικό και Καθολικό: g, B (κλήση)

A

Τοπικό: n

Μη-Τοπικό: a, x, A (κλήση), R (κλήση) (από B), g, B (κλήση) (από MAIN)

Καθολικό: B (κλήση)

R

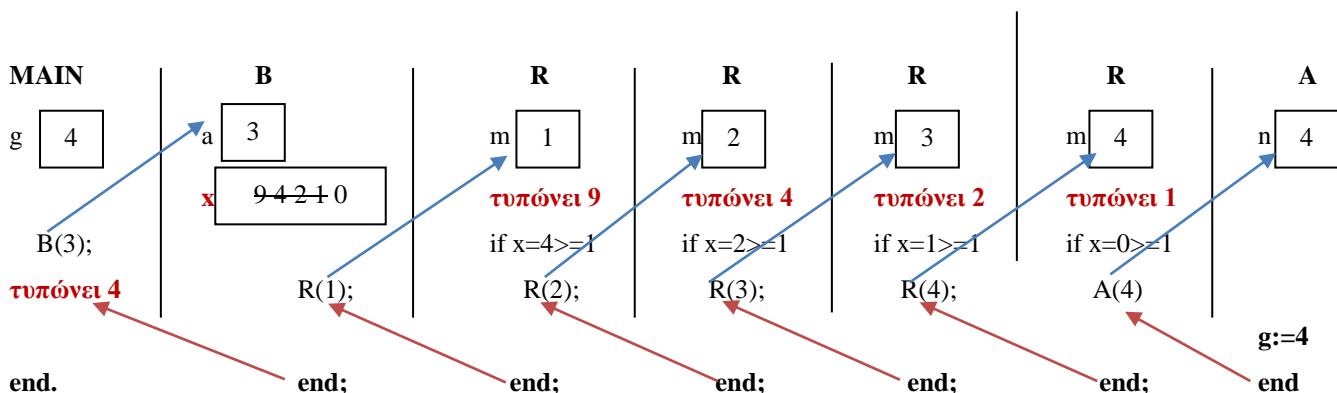
Τοπικό: m

Μη-Τοπικό: a, x, A (κλήση), R (κλήση) (από B), g, B (κλήση) (από MAIN)

Καθολικό: B (κλήση)

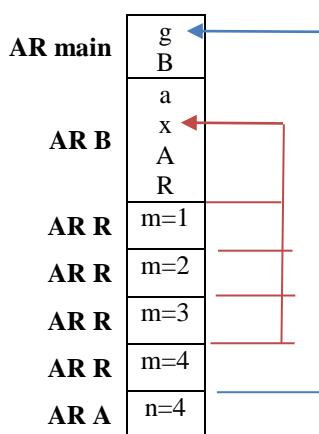
(β)

Οι τιμές που τυπώνονται είναι οι εξής:



Τυπώνει κατά σειρά 9, 4, 2, 1 και 4

(γ) Η στοίβα εκτέλεσης με τις εγγραφές ενεργοποίησης είναι η ακόλουθη:



Παρατηρήσεις για τη στοίβα εκτέλεσης

Οι στατικοί δείκτες χρησιμοποιούνται όταν μέσα σε ένα υποπρόγραμμα χρησιμοποιούνται μεταβλητές που δεν έχουν δηλωθεί σε αυτό. Σε αυτή την περίπτωση μόλις βάλουμε το block του υποπρογράμματος μέσα στη στοίβα δημιουργούμε ένα στατικό δείκτη προς όλες τις μεταβλητές που χρησιμοποιεί το υποπρόγραμμα αυτό και δεν είναι δικές του (δηλ. δηλωμένες μέσα σε αυτό).

1.18 Θέμα 1 Ατυπη 2009

Δίνεται ο παρακάτω κώδικας σε C:

```
int x, y, z;

main()
{
    x=15;

    y=10;

    z=A(x, y);

    print(z);
}

int A(int u, int v)
{
    if (v==0)
        return u;
    else
        return A(v, u%v);
}
```

(1) Να γράψετε τα περιβάλλοντα αναφοράς όλων των τμημάτων του προγράμματος

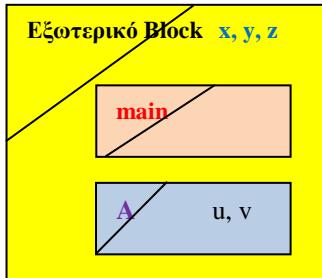
(2) Ποια είναι η λειτουργία της function A;

(3) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (activation records-AR) κατά την εκτέλεση της τελευταίας αναδρομής της A (δηλ. όταν είναι σε ισχύ το μέγιστο μέγεθος της στοίβας). Ποια είναι η τιμή της z θα τυπωθεί τελικά στο main;

(4) Στο συγκεκριμένο πρόγραμμα ποια η χρησιμότητα (αν υπάρχει) των στατικών δεικτών (static pointers) της στοίβας εκτέλεσης;

Απάντηση

(1)



Περιβάλλοντα Αναφοράς

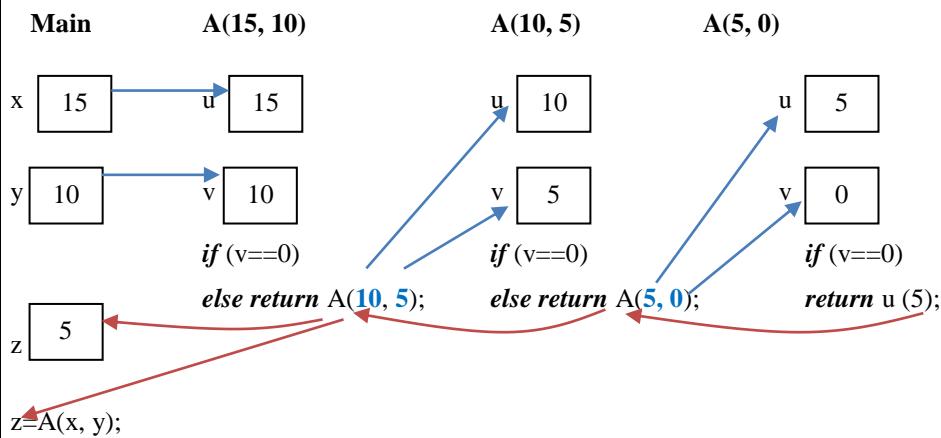
Main

Τοπικό: – και Μη-Τοπικό και Καθολικό: x, y, z, main (κλήση), A (κλήση)

A

Τοπικό: u, v και Μη-Τοπικό και Καθολικό: x, y, z, main (κλήση), A (κλήση)

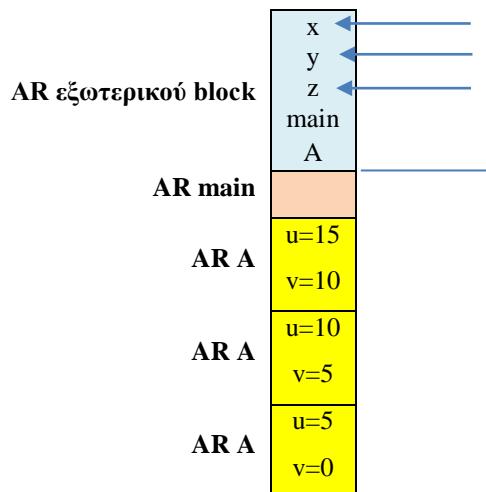
(2)



Το πρόγραμμα τυπώνει την τιμή 5 στο main

Η συνάρτηση A υπολογίζει το ΜΚΔ των 2 ορισμάτων που λαμβάνει εφαρμόζοντας τον αλγόριθμο του Ευκλείδη

(3) Η στοίβα εκτέλεσης με τις εγγραφές ενεργοποίησης είναι η ακόλουθη:



(4) Οι στατικοί δείκτες χρειάζονται στο main γιατί χρησιμοποιεί τις μεταβλητές x, y, z που δεν δηλώνονται μέσα σε αυτό.

Γενικά οι στατικοί δείκτες πηγαίνουν στο AR κάθε συνάρτησης (και του main αν χρειάζεται) που χρησιμοποιεί μεταβλητές που δεν δηλώνει η ίδια.

1.19 Θέμα 2 -Ιούνιος 2015

Δίνονται τα παρακάτω δύο προγράμματα C τα οποία εκτελούνται σε ένα compiler ο οποίος στην εντολή ανάθεσης προσδιορίζει πρώτα την l-value (αν χρειάζεται να προσδιοριστεί).

Πρόγραμμα A	Πρόγραμμα B
<pre> int n=1; int A[10]; int P(int x) { printf("%d\n", x); n=n+1; printf("%d\n", x); x=x+5; return x; } main() { A[1]=10; A[2]=20; A[3]=30; A[4]=40; A[5]=50; A[6]=x; A[n+3]=P(A[n+2]); printf("A[1]=%d\n", A[1]); printf("A[2]=%d\n", A[2]); printf("A[3]=%d\n", A[3]); printf("A[4]=%d\n", A[4]); printf("A[5]=%d\n", A[5]); return 0; } </pre>	<pre> int n=1; int A[10]; int P(int *x) { printf("%d\n", *x); n=n+1; printf("%d\n", *x); *x=*x+5; return *x; } main() { A[1]=10; A[2]=20; A[3]=30; A[4]=40; A[5]=50; A[6]=x; A[n+3]=P(&A[n+2]); printf("A[1]=%d\n", A[1]); printf("A[2]=%d\n", A[2]); printf("A[3]=%d\n", A[3]); printf("A[4]=%d\n", A[4]); printf("A[5]=%d\n", A[5]); return 0; } </pre>

a) Ποιες είναι οι **ομοιότητες και οι διαφορές** των δύο προγραμμάτων ιδιαίτερα όσον αφορά τον τρόπο μεταβίβασης παραμέτρων, τις μεταβλητές και τις σχέσεις τους; **Ποια είναι η έξοδος των δύο προγραμμάτων** κατά την εκτέλεση τους;

b) Ποια είναι **τα περιβάλλοντα αναφοράς** των P και main στο πρόγραμμα A;

c) Στα παραπάνω προγράμματα υπάρχει μια εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει error. Ποια είναι και γιατί; **Σβήστε την εντολή από τον κώδικα.**

d) Ποια θα ήταν η έξοδος του προγράμματος A κατά την εκτέλεση του, αν υποθέταμε ότι η μεταβίβαση παραμέτρων γινόταν με τους παρακάτω δύο τρόπους (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού):

(i) κλήση με τιμή-αποτέλεσμα (call by value-result)

(ii) κλήση με όνομα (call by name)

Απάντηση

(a)

Ομοιότητα

Το επιστρεφόμενο αποτέλεσμα της συνάρτησης καταχωρείται και στα 2 προγράμματα στο στοιχείο A[n+3]

Διαφορά

- Στο πρόγραμμα A η συνάρτηση P καλείται με call by value και μεταβιβάζεται η τιμή του στοιχείου A[n+2] του πίνακα A στην παράμετρο x της συνάρτησης A
- Στο πρόγραμμα B η συνάρτηση P καλείται με call by reference και μεταβιβάζεται η διεύθυνση του στοιχείου A[n+2] του πίνακα A με το σύμβολο & στην παράμετρο x της συνάρτησης A που είναι δείκτης

Η έξοδος των Προγραμμάτων είναι:

Πρόγραμμα A

0	1	2	3	4	5	6	7	8	9
	10	20	30	40	50				

A[4]=P(A[3]);

x

30 35

n

+ 2

Αποτελέσματα Εκτέλεσης

Στη συνάρτηση A τυπώνονται οι τιμές:

30

30

Στο main τυπώνεται ο πίνακας A με τις ακόλουθες τιμές:

10

20

30

35

50

Πρόγραμμα B

0	1	2	3	4	5	6	7	8	9
	10	20	30	40	50				

A[4]=P(&A[3]);

x

+ 2

Αποτελέσματα Εκτέλεσης

Στη συνάρτηση A τυπώνονται οι τιμές:

30

30

Στο main τυπώνεται ο πίνακας A με τις ακόλουθες τιμές

10

20

30

35

50

Σχολιασμός Προγράμματος Α

```

1.int n=1;
2.int A[10];

3.int P(int x)
4.{  

    5.printf("%d\n", x);
    6.n=n+1;
    7.printf("%d\n", x);
    8.x=x+5;
    9.return x;
10.}

11. main()
12.{  

13.    A[1]=10;
14.    A[2]=20;
15.A[3]=30;
16.A[4]=40;
17.A[5]=50;
18.//A[6]=x; H εντολή αυτή είναι συντακτικά λάθος γιατί η μεταβλητή x δεν έχει δηλωθεί ούτε στη main ούτε καθολικά και την αφαιρούμε από το πρόγραμμα
19.A[n+3]=P(A[n+2]);//καταχωρείται στο A[n+3] το επιστρεφόμενο αποτέλεσμα της συνάρτησης
20.printf("A[1]=%d\n", A[1]);
21.printf("A[2]=%d\n", A[2]);
22.printf("A[3]=%d\n", A[3]);
23.printf("A[4]=%d\n", A[4]);
24.printf("A[5]=%d\n", A[5]);
25.return 0;
}

```

Επεξήγηση Αποτελεσμάτων

- Στη γραμμή 19 του main που έχει την εντολή $A[n+3]=P(A[n+2]);$ μεταβιβάζεται στη συνάρτηση P η τιμή του στοιχείου $A[3]=30$ και καταχωρείται στην παράμετρο x της P.
- Στη γραμμή 5 της P τυπώνεται η τιμή 30
- Στη γραμμή 6 της P το n γίνεται ίσο με 2
- Στη γραμμή 7 της P τυπώνεται η τιμή 30
- Στη γραμμή 8 της P το x γίνεται ίσο με 35
- Στη γραμμή 9 της P η συνάρτηση τερματίζεται και επιστρέφεται η τιμή 35

Σχολιασμός Προγράμματος Β

```

1.int n=1;
2.int A[10];

3.int P(int *x) //To * στη δήλωση δηλώνει ότι το x είναι δείκτης σε int
4.{  

    5.printf("%d\n", *x); //To * στην εντολή αναφέρεται στο περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x
    6.n=n+1;
    7.printf("%d\n", *x);
    8.*x=*x+5; //το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x ανζάνεται κατά 5
    9.return *x; //επιστρέφεται το περιεχόμενο της θέσης στην οποία δείχνει ο δείκτης x
10.}

11. main()
12.{  

13.A[1]=10;
14.A[2]=20;
15.A[3]=30;
16.A[4]=40;
17.A[5]=50;
18.//A[6]=x; Oμοίως η εντολή αυτή είναι συντακτικά λάθος και την αφαιρούμε από το πρόγραμμα
19.A[n+3]=P(&A[n+2]); //καταχωρείται στο A[n+3] το επιστρεφόμενο αποτέλεσμα της συνάρτησης
20.printf("A[1]=%d\n", A[1]);
21.printf("A[2]=%d\n", A[2]);
22.printf("A[3]=%d\n", A[3]);
23.printf("A[4]=%d\n", A[4]);
24.printf("A[5]=%d\n", A[5]);
25.return 0;
}

```

Επεξήγηση Αποτελεσμάτων

- Στη γραμμή 19 του main που έχει την εντολή $A[n+3]=P(&A[n+2]);$ μεταβιβάζεται στη συνάρτηση P η διεύθυνση του στοιχείου $A[3]$ και καταχωρείται στο δείκτη x της P.
- Στη γραμμή 5 της P τυπώνεται το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x δηλ. το περιεχόμενο του στοιχείου $A[n+3]$ που είναι 30
- Στη γραμμή 6 της P το n γίνεται ίσο με 2

Στη γραμμή 19 του main στο στοιχείο A[4] καταχωρείται η τιμή 35. Πρόσοχη: Παρόλο που το n είναι τώρα 2, όταν κλήθηκε αρχικά η συνάρτηση P στη γραμμή 19 το n ήταν 1 και λόγω του l-value το n κρατά αυτή την τιμή. Άρα η τιμή 35 θα καταχωρηθεί στο A[4].

- Στη γραμμή 20 του main τυπώνεται η τιμή του A[1] δηλ. 10
- Στη γραμμή 21 του main τυπώνεται η τιμή του A[2] δηλ 20
- Στη γραμμή 22 του main τυπώνεται η τιμή του A[3] δηλ 30
- Στη γραμμή 23 του main τυπώνεται η τιμή του A[4] δηλ 35
- Στη γραμμή 24 του main τυπώνεται η τιμή του A[5] δηλ 50

Αυτός ο τρόπος εκτέλεσης είναι η κλήση με πέρασμα τιμών (call by value) όσον αφορά τον τρόπο που καλείται η συνάρτηση P από το main(). Εδώ συγκεκριμένα μεταβιβάζουμε στη συνάρτηση P την τιμή του στοιχείου A[n+2] και η αντίστοιχη παράμετρος της P που λαμβάνει την τιμή αυτή είναι η μεταβλητή x.

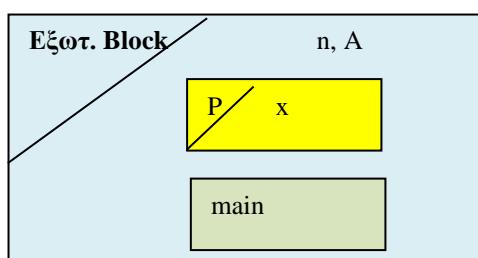
- Στη γραμμή 7 της P τυπώνεται η τιμή 30
- Στη γραμμή 8 της P το αυξάνεται κατά 5 το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x δηλ. το περιεχόμενο του στοιχείου A[3] γίνεται ίσο με 35
- Στη γραμμή 9 της P η συνάρτηση τερματίζεται και επιστρέφεται το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x δηλ. το περιεχόμενο του στοιχείου A[3] που είναι ίσο με 35. Το περιεχόμενο αυτό καταχωρείται στο στοιχείο A[4] λόγω του l-value και το κάνει 35.

Στη γραμμή 19 του main στο στοιχείο A[4] καταχωρείται η τιμή 35. Πρόσοχη: Παρόλο που το n είναι τώρα 2, όταν κλήθηκε αρχικά η συνάρτηση P στη γραμμή 19 το n ήταν 1 και λόγω του l-value το n κρατά αυτή την τιμή. Άρα η τιμή 35 θα καταχωρηθεί στο A[4].

- Στη γραμμή 20 του main τυπώνεται η τιμή του A[1] δηλ 10
- Στη γραμμή 21 του main τυπώνεται η τιμή του A[2] δηλ 20
- Στη γραμμή 22 του main τυπώνεται η τιμή του A[3] δηλ 35
- Στη γραμμή 23 του main τυπώνεται η τιμή του A[4] δηλ 35
- Στη γραμμή 24 του main τυπώνεται η τιμή του A[5] δηλ 50

Αυτός ο τρόπος εκτέλεσης είναι η κλήση με πέρασμα αναφορών (call by reference) όσον αφορά τον τρόπο που καλείται η συνάρτηση P από το main(). Εδώ συγκεκριμένα μεταβιβάζουμε στη συνάρτηση P τη διεύθυνση του στοιχείου A[n+2]

(b) Περιβάλλοντα Αναφοράς των P και main στο Πρόγραμμα A



Main

Τοπικό: —

Μη Τοπικό και Καθολικό: n, A, P, main

P

Τοπικό: x

Μη Τοπικό και Καθολικό: n, A, P, main

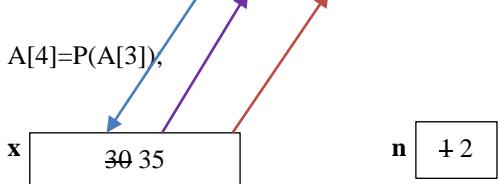
(c) Και στα δύο προγράμματα η εντολή A[6]=x; είναι συντακτικά λάθος διότι η μεταβλητή x ΔΕΝ έχει δηλωθεί στο main. Άρα πρέπει να τη διαγράψουμε για να εκτελεστεί σωστά το main.

(d)

Call by value result

Ο τρόπος εκτέλεσης που έχουμε περιγράψει προηγουμένως για το πρόγραμμα A είναι εξορισμού το call by value και οι τιμές που εκτυπώνονται έχουν ήδη περιγραφεί.

0	1	2	3	4	5	6	7	8	9
	10	20	30	40	50				



Οι τιμές που τυπώνονται στη συνάρτηση P είναι:

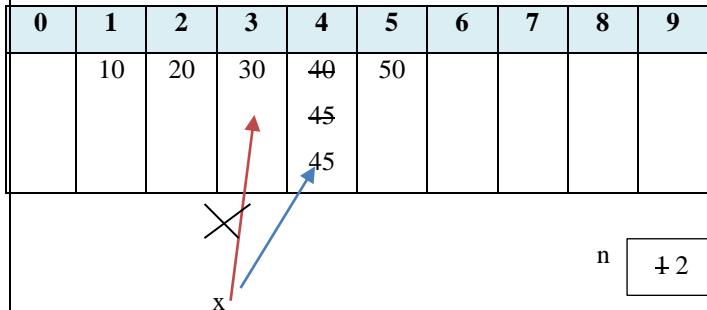
- 30
- 30

Οι τιμές που τυπώνονται στο main για τον πίνακα A είναι:

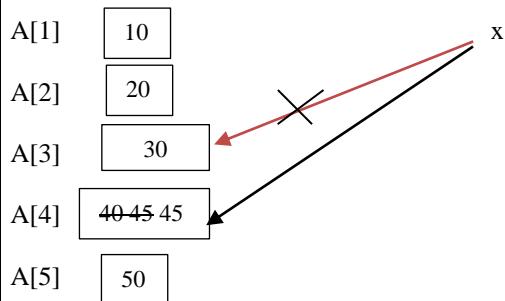
- A[1]=10
- A[2]=20
- A[3]=35
- A[4]=35
- A[5]=50

Call by name

Στην εντολή $A[4]=P(A[3])$ το όρισμα της συνάρτησης P είναι η παράμετρος $A[3]$. Το στοιχείο $A[3]$ του πίνακα A είναι μια απλή μεταβλητή. Όταν στο call by name μεταβιβάζουμε μεταβλητές τότε αυτό ταυτίζεται με το call by reference, άρα μεταβιβάζεται η διεύθυνση του στοιχείου $A[3]$ στο δείκτη x .



$A[4]=P(&A[1+2]);$ άρα $A[3]\leftarrow x.$ Μετά $A[2+2]\leftarrow x$



Οι τιμές που τυπώνονται στη συνάρτηση P είναι:

- 30
- 40

Οι τιμές που τυπώνονται στο main για τον πίνακα A είναι:

- $A[1]=10$
- $A[2]=20$
- $A[3]=30$
- $A[4]=45$
- $A[5]=50$

Προσοχή: Οι πίνακες στη C, C++ και στη Java αρχίζουν από τη θέση 0, ενώ στην Pascal αρχίζουν από τη θέση 1 εκτός και αν ο πίνακας δηλώνεται διαφορετικά.

Παρατήρηση

Η αναφορά στην εκφώνηση ότι ο compiler στην εντολή ανάθεσης προσδιορίζει πρώτα την l-value σημαίνει ότι σε όλα τα ερωτήματα κατά την κλήση της συνάρτησης P με την εντολή $A[n+3]=P(A[n+2]);$ **το στοιχείο του πίνακα A αριστερά από την καταχώριση** δεν αλλάζει ποτέ έστω και αν τροποποιείται η τιμή του $n.$

1.20 Θέμα 3 Ιούνιος 2021

Δίνονται τα παρακάτω δύο προγράμματα μιας γλώσσα προγραμματισμού η οποία όσο αφορά το συντακτικό και τη γενική δομή των προγραμμάτων της είναι τύπου Pascal, ενώ υιοθετεί τις τεχνικές μεταβίβασης παραμέτρων καθώς και τη σημειογραφία και τη λειτουργικότητα δεικτών (pointers) της C. Τα προγράμματα εκτελούνται σε ένα compiler ο οποίος στην εντολή ανάθεσης προσδιορίζει **πρώτα τη l-value** (αν χρειάζεται να προσδιοριστεί)

- a) Ποιες είναι οι **ομοιότητες και οι διαφορές των δύο προγραμμάτων** ιδιαίτερα όσον αφορά τον τρόπο μεταβίβασης παραμέτρων, τις μεταβλητές και τις σχέσεις τους;
- b) Ποια είναι τα **περιβάλλοντα αναφοράς** των **P** και **main** στο πρόγραμμα **A**;
- c) Στα παραπάνω προγράμματα υπάρχει μια εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει error. Ποια είναι και γιατί; Σβήστε την εντολή από τον κώδικα.
- d) **Ποια είναι η έξοδος των δύο προγραμμάτων κατά την εκτέλεση τους;** Εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού
- e) Ποια θα ήταν η έξοδος του προγράμματος **A** κατά την εκτέλεση του, αν υποθέταμε ότι η μεταβίβαση παραμέτρων γινόταν με τους παρακάτω δύο τρόπους (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού):
- (i) κλήση με τιμή-αποτέλεσμα (call by value-result)
 - (ii) κλήση με όνομα (call by name)

Πρόγραμμα A

```
program MAIN;
  var n: integer;
      a: array [0..10] of integer;

  function P(x: integer): integer;
  begin
    write(x);
    n:= n + 1;
    write(x);
    x:= x + 5;
    P:= x;
  end;

  BEGIN
    n:= 1;
    a[0]:= x; a[1]:= 10; a[2]:= 20;
    a[3]:= 30; a[4]:= 40; a[5]:= 50;
    a[6]:= 60;
    a[n+3]:= P(a[n+2]);
    write(a[1]);
    write(a[2]);
    write(a[3]);
    write(a[4]);
    write(a[5]);
  END.
```

Πρόγραμμα B

```
program MAIN;
  var n: integer;
      a: array [0..10] of integer;

  function P(*x: integer): integer;
  begin
    write(*x);
    n:= n + 1;
    write(*x);
    *x:= *x + 5;
    P:= *x;
  end;

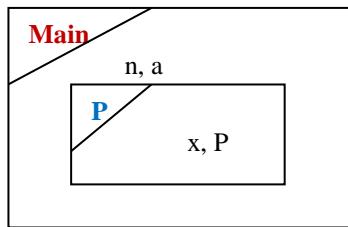
  BEGIN
    n:= 1;
    a[0]:= x; a[1]:= 10; a[2]:= 20;
    a[3]:= 30; a[4]:= 40; a[5]:= 50;
    a[6]:= 60;
    a[n+3]:= P(&a[n+2]);
    write(a[1]);
    write(a[2]);
    write(a[3]);
    write(a[4]);
    write(a[5]);
  END.
```

Απάντηση

a)

- Η παράμετρος **x** στη **function P** στο πρόγραμμα **A** είναι ακέραια μεταβλητή, ενώ η παράμετρος **x** στη **function P** στο πρόγραμμα **B** είναι δείκτης σε ακέραια μεταβλητή. **Συνεπώς η μεταβίβαση παραμέτρων στο πρόγραμμα A γίνεται με call by value ενώ η μεταβίβαση παραμέτρων στο πρόγραμμα B γίνεται με call by reference**
- Η ομοιότητα των δύο προγραμμάτων **A** και **B** ότι αρχικά θέτουν τιμές στον του πίνακα **a** ο οποίος και στα δύο προγράμματα δηλώνεται στο **main**. Η **function P** και στα δύο προγράμματα ενημερώνει στοιχεία του πίνακα **a** και στο τέλος και τα δύο προγράμματα τυπώνουν τις ενημερωμένες τιμές του πίνακα **a**

b)



Περιβάλλοντα Αναφοράς

Main: Τοπικό: n, a, P

Mη-Τοπικό και Καθολικό: —

P: Τοπικό: x, P (τιμή)

Mη-Τοπικό και Καθολικό: n, a, P (κλήση)

c)

Η εντολή που είναι λάθος από σημασιολογική άποψη και θα προκαλέσει error και στα δύο προγράμματα είναι η **a[0]:=x;** διότι δεν υπάρχει δήλωση της x στο main ούτε στο πρόγραμμα A ούτε στο πρόγραμμα B

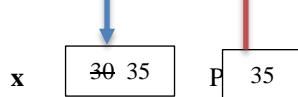
d)

Εκτέλεση Προγράμματος A

n + 2

0	1	2	3	4	5	6	7	8	9	10
	10	20	30	40 35	50	60				

a[4]:=P(a[3]);



Δειτούργια Προγράμματος A

- Αρχικά $a[n+3]=P(a[n+2])$; δηλ. $a[4]:=P(a[3])$; δηλ. καλείται η συνάρτηση P με όρισμα την τιμή του στοιχείου $a[3]$ διότι $n=1$ και το επιστρεφόμενο αποτέλεσμα της P θα καταχωρηθεί στο στοιχείο $a[4]$
- Λόγω του l-value η τιμή $a[n+3]$ που καταχωρείται το αποτέλεσμα της function P δεν επηρεάζεται από την αλλαγή του n
- Η function P τυπώνει την τιμή 30 στην 1^η εντολή $write(x)$; διότι το x έλαβε την τιμή 30 από το $a[3]$
- Η function P τυπώνει πάλι την τιμή 30 και στη 2^η εντολή $write(x)$; διότι το x δεν έχει αλλάξει τιμή
- Στην εντολή $x:=x+5$; το x αλλάζει τιμή και γίνεται 35
- Στην εντολή $P:=x$; καταχωρείται στη μεταβλητή P η τιμή 35 δηλ. η τιμή της x. Η τελευταία εντολή μιας Function στην Pascal είναι η καταχώριση του τελικού αποτελέσματος της function στο όνομα της που είναι ταυτόχρονα και τοπική μεταβλητή της function και το οποίο επιστρέφεται στο MAIN όταν η function τερματίσει
- Όταν τελειώσει η function P επιστρέφει μέσω του ονόματος της το τελικό αποτέλεσμα δηλ. την τιμή 35 και την καταχωρεί στο $a[4]$
- Στο MAIN τυπώνονται οι τελικές τιμές του πίνακα A δηλ.

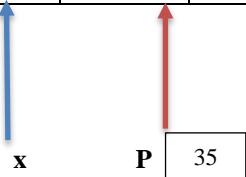
10 20 30 35 50 60

Εκτέλεση Προγράμματος B

n x2

0	1	2	3	4	5	6	7	8	9	10
	10	20	30 35	40 35	50	60				

A[4]:=P(&a[3]);



Λειτουργία Προγράμματος B

- Αρχικά $a[n+3]=P(&a[n+2]);$ δηλ. $a[4]:=P(&a[3]);$ Δηλ. καλείται η συνάρτηση P με όρισμα τη διεύθυνση του a[3] διότι $n=1$ και το επιστρεφόμενο αποτέλεσμα της P θα καταχωρηθεί στο στοιχείο a[4]
- Λόγω του l-value η τιμή a[n+3] αριστερά της καταχώρισης δεν επηρεάζεται από την αλλαγή του n
- Η function P τυπώνει την τιμή 30 στην 1^η εντολή write(*x); διότι τυπώνεται το περιεχόμενο της μεταβλητής a[3] στην οποία δείχνει ο δείκτης x
- Η function P τυπώνει πάλι την τιμή 30 στην 2^η εντολή write(*x); διότι το περιεχόμενο της μεταβλητής a[3] στην οποία δείχνει ο δείκτης x δεν έχει αλλάξει
- Με την εντολή $*x:=*x+5;$ ενημερώνεται το περιεχόμενο της μεταβλητής a[3] στην οποία δείχνει ο δείκτης x σε 35
- Στην εντολή $P:=*x;$ καταχωρείται στην τοπική μεταβλητή P που είναι το όνομα της function το περιεχόμενο της μεταβλητής a[3] στην οποία δείχνει ο δείκτης x δηλ. η τιμή 35
- Όταν τελειώσει η function P επιστρέφει μέσω του ονόματος της που είναι μεταβλητή το τελικό αποτέλεσμα δηλ. την τιμή 35 και την καταχωρεί στο a[4]
- Στο MAIN τυπώνονται οι τελικές τιμές του πίνακα A δηλ.

10 20 35 35 50 60

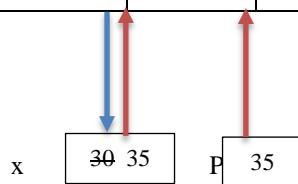
e)

Call by value – result στην Εκτέλεση Προγράμματος A

n ✓2

0	1	2	3	4	5	6	7	8	9	10
	10	20	30 35	40 35	50	60				

a[4]:=P(a[3]);



Δειτουργία Function P

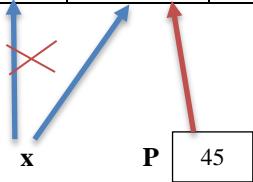
- Αρχικά `a[n+3]:=P(a[n+2]);` δηλ. `a[4]:=P(a[3]);` δηλ. καλείται η συνάρτηση `P` με όρισμα την τιμή του στοιχείου `a[3]` διότι `n=1` και το επιστρεφόμενο αποτέλεσμα της `P` θα καταχωρηθεί στο στοιχείο `a[4]`
- Λόγω του l-value η τιμή `a[n+3]` που καταχωρείται το αποτέλεσμα της function `P` δεν επηρεάζεται από την αλλαγή του `n`
- H function `P` τυπώνει την τιμή 30 στην 1^η εντολή `write(x);` διότι το `x` έλαβε την τιμή 30 από το `a[3]`
- H function `P` τυπώνει πάλι την τιμή 30 και στη 2^η εντολή `write(x);` διότι το `x` δεν έχει αλλάξει τιμή
- Στην εντολή `x:=x+5;` το `x` αλλάζει τιμή και γίνεται 35
- Στην εντολή `P:=x;` καταχωρείται στη μεταβλητή `P` η τιμή 35 δηλ. η τιμή της `x`. Η τελευταία εντολή μιας Function στην Pascal είναι η καταχώριση του τελικού αποτελέσματος της function στο όνομα της που είναι ταυτόχρονα και τοπική μεταβλητή της function και το οποίο επιστρέφεται στο MAIN όταν η function τερματίσει
- Όταν τελειώσει η function `P` επιστρέφει μέσω του ονόματος της το τελικό αποτέλεσμα δηλ. την τιμή 35 και την καταχωρεί στο `a[4]`
- Επιπλέον επειδή έχουμε call-by-value-result η παράμετρος `x` επιστρέφει αντίστροφα πίσω στο `a[3]` από το οποίο έλαβε τιμή και το κάνει 35
- Όταν τελειώσει η function `P` επιστρέφουμε στο MAIN και τυπώνονται οι τιμές του πίνακα `A` δηλ.

10 20 **35** **35** 50 60

Call by name στην Εκτέλεση Προγράμματος A

n + 2

0	1	2	3	4	5	6	7	8	9	10
	10	20	30 35	40 45 45	50	60				



Λειτουργία Function P

- Επειδή ο τρόπος κλήσης είναι call-by-name και η παράμετρος a[3] που μεταβιβάζουμε είναι μια ακέραια μεταβλητή το call-by-name ταυτίζεται με το call-by-reference και μεταβιβάζεται η διεύθυνση του a[3] στο δείκτη x
- Η function P τυπώνει 30 στην 1^η εντολή write(x); διότι τυπώνεται το περιεχόμενο της μεταβλητής a[3] στην οποία δείχνει ο δείκτης x
- Με την αύξηση του n κατά 1 το n γίνεται 2. Ο δείκτης x δείχνει αρχικά στο a[n] δηλ. στο a[3]. Όταν όμως το n γίνει 2 ο δείκτης x ενημερώνεται αυτόματα και δείχνει πλέον στο a[4] δηλ.



- Η function P τυπώνει την τιμή 40 στη 2^η εντολή write(x); διότι ο δείκτης x δείχνει τώρα στο a[4] που έχει τιμή 40
- Με την εντολή x:=x+5; ενημερώνεται το περιεχόμενο της μεταβλητής a[4] στην οποία δείχνει ο δείκτης x σε 45
- Στην εντολή P:=x; καταχωρείται στην τοπική μεταβλητή P που είναι το όνομα της function, το περιεχόμενο της μεταβλητής a[4] στην οποία δείχνει ο δείκτης x δηλ. η τιμή 45
- Όταν τελειώσει η function P επιστρέφει μέσω του ονόματος της το τελικό αποτέλεσμα δηλ την τιμή 45 στο στοιχείο a[4] διότι λόγω του l-value το στοιχείο αριστερά της καταχώρισης δεν αλλάζει δηλ. το a[n+3] αριστερά της καταχώρισης δεν επηρεάζεται από την αλλαγή του n
- Στο MAIN και τυπώνονται οι τελικές τιμές του πίνακα A δηλ.

10 20 35 45 50 60

1.21 Θέμα 2 -Σεπτέμβριος 2015

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί το **στατικό κανόνα εμβέλειας**:

program MAIN;

var y, z:integer;

function f(a: integer): integer;

begin

y:=a+1;

f:=y+a;

end;

function g(x: integer): integer;

begin

y:=f(x+1)+1;

z:=f(y-x);

g:=z+2;

end;

BEGIN

y:=6;

z:=g(2*y);

write(y, z);

END.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη τοπικά καθολικά) όλων των τμημάτων του προγράμματος;

(b) Παρουσιάστε τη στοίβα εκτέλεσης (Run-time Stack) των εγγραφών ενεργοποίησης (Activation Records-AR) μόλις έχει γίνει η $2^{\text{η}}$ κλήση της **f**

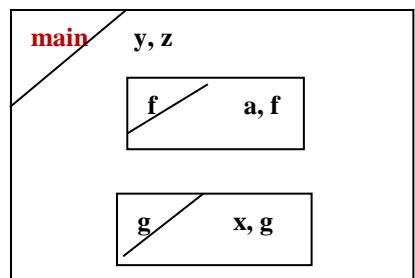
(c) Ποιες τιμές θα τυπωθεί στο τέλος για τα **y** και **z** στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων; Παρουσιάστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού στις δύο περιπτώσεις:

(i) Κλήση με τιμή (call by value)

(ii) Κλήση με όνομα (call by name)

Απάντηση

(a)



Περιβάλλοντα Αναφοράς

Main

Τοπικό: y, z, f (κλήση), g (κλήση)

Μη-Τοπικό και Καθολικό: –

f

Τοπικό: a, f (τιμή)

Μη-Τοπικό και Καθολικό: y, z, f (κλήση), g (κλήση)

g

Τοπικό: x, g (τιμή)

Μη-Τοπικό και Καθολικό y, z, f (κλήση), g (κλήση)

(c)

call by value

Main

g

f

f

y ↙ 14 ↘ 17

x 12

a 13

a 16

y:=f(x+1)+1

f 27

f 33

z:=f(y-x)

g 35

z ↙ 33 ↘ 35

z:=g(2*y);

Τυπώνονται οι τιμές 17, 35 στο main

call by name

Main

g

f

f

y ↙ 14 ↘ 17

x 12

a 13

a 16

y:=f(x+1)+1

f 27

f 33

z:=f(y-x)

g 35

z ↙ 35 ↘ 35

z:=g(2*y);

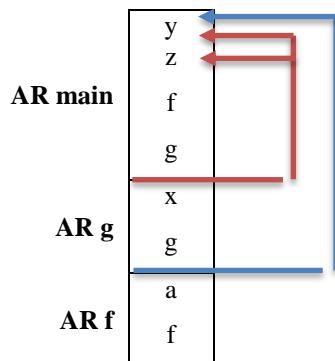
Τυπώνονται οι τιμές 17, 35 στο main

Παρατηρήσεις

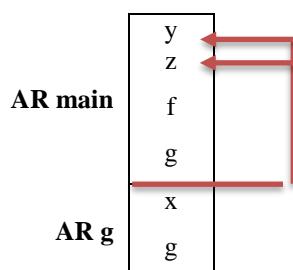
- Η κλήση $g(2*y)$ είναι call by value διότι μεταβιβάζουμε έκφραση-παράσταση. Γίνεται η πράξη 2*6 και μεταβιβάζεται η σταθερά 12
- Η κλήση $f(x+1)$ είναι call by value διότι μεταβιβάζουμε έκφραση-παράσταση. Γίνεται η πράξη $x+1$ και μεταβιβάζεται η σταθερά 13
- Η κλήση $f(y-x)$ είναι call by value διότι μεταβιβάζουμε έκφραση-παράσταση. Γίνεται η πράξη $x+1$ και μεταβιβάζεται η σταθερά 16
- Σε όλες τις συναρτήσεις που καλούμε τα ορίσματα που μεταβιβάζουμε είναι εκφράσεις και όχι μεταβλητές. Άρα το call by name ταυτίζεται με το call by value σε όλες τις περιπτώσεις

(b)

Η στοίβα εκτέλεσης με τις εγγραφές ενεργοποίησης μέχρι και την 1^η κλήση της f είναι η ακόλουθη:

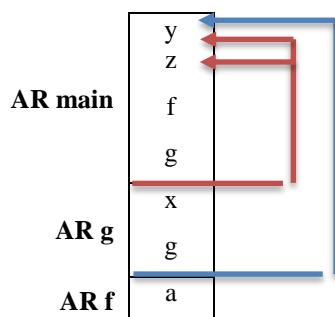


Όταν τελειώσει η εκτέλεση της f, το AR της f αφαιρείται από τη στοίβα όπως φαίνεται στο ακόλουθο σχήμα:



Αυτή είναι η στοίβα εκτέλεσης όταν το AR της f αφαιρεθεί από τη στοίβα

Η στοίβα εκτέλεσης με τις εγγραφές ενεργοποίησης στη 2^η κλήση της f είναι η ακόλουθη:



| f |

1.22 Ασκηση 1 Φροντιστήριο 2015

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας και η μεταβίβαση παραμέτρων γίνεται **με κλήση με τιμή** (call by value):

program MAIN;

var x, y, z:integer;

function A(w: integer): integer;

begin

y:=10;

*A:=x*w;*

*y:=4*k;* █

end;

function B(x: integer): integer;

var k:integer;

begin

k:=12;

B:=A(k);

end;

BEGIN

x:=3; y:=2;

z:=B(y);

*x:=2*w;* █

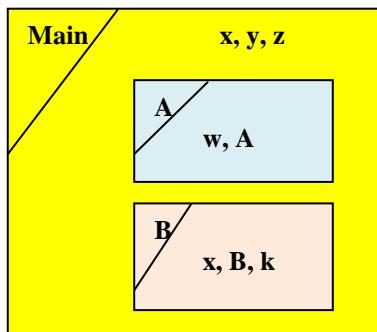
writeln(z); writeln(y);

END.

- Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- Στον παραπάνω κώδικα υπάρχουν **δυο εντολές οι οποίες δεν είναι σωστές από σημασιολογική άποψη** και θα προκαλέσουν run-time error. Ποιες είναι αυτές και γιατί; Σβήστε τις δύο εντολές από τον κώδικα
- Παρουσιάστε τη **στοίβα εκτέλεσης** (run-time stack) των εγγραφών ενεργοποίησης (Activation Records-AR) μόλις έχει γίνει η κλήση της A
- Ποιες **τιμές θα τυπωθούν στο τέλος για τα z και y**;
- Απαντήστε στα (b), (d) θωρώντας ότι ισχύει ο **δυναμικός κανόνας εμβέλειας**. Τώρα μόνο η μια από τις δύο εντολές δεν είναι **σωστή από σημασιολογική άποψη**. Ποια είναι η εντολή που είναι τώρα σωστή; Εξηγείστε τη διαφοροποίηση από την αρχική απάντηση σας

Απάντηση

(a)



Τα περιβάλλοντα Αναφοράς είναι τα ακόλουθα:

Main

Τοπικό: x, y, z, A (κλήση), B (κλήση)

Μη-Τοπικό και Καθολικό: –

B

Τοπικό: x, B (τιμή), k

Μη-Τοπικό και Καθολικό: y, z, A (κλήση), B (κλήση)

A

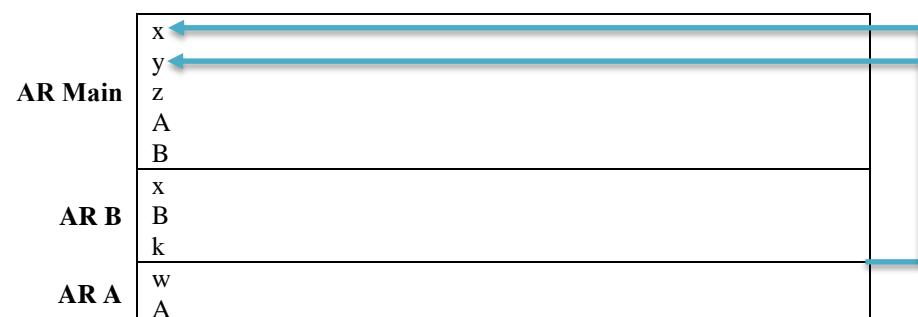
Τοπικό: w, A (τιμή)

Μη-Τοπικό και Καθολικό: , y, z, A (κλήση), B (κλήση)

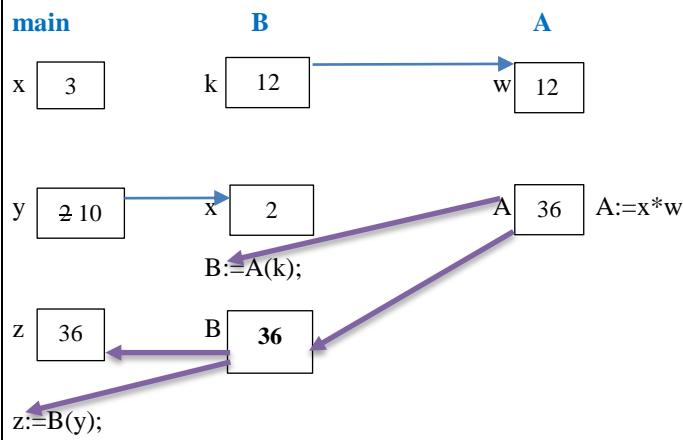
(b)

- Η εντολή $y:=4*k$ είναι λάθος και πρέπει να διαγραφεί διότι η k δεν δηλώνεται ούτε στη function A ούτε στο main.
- Η εντολή $x:=2*w$ στο main είναι λάθος διότι το main δεν έχει δηλωμένη μεταβλητή w. Άρα και η εντολή αυτή διαγράφεται

(c) Στοίβα Εκτέλεσης



(d)

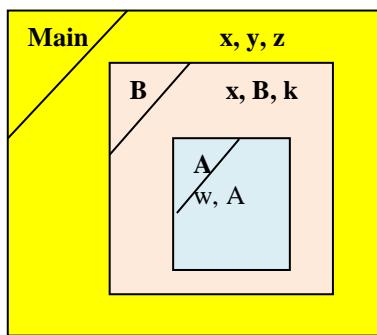


Στο main τυπώνονται οι τιμές $y=10$, $z=36$

Παρατήρηση

Η εντολή `A:=x*w` που καταχωρεί το τελικό αποτέλεσμα στο όνομα της function πρέπει να είναι η τελευταία εντολή της function.

(e)



Τα περιβάλλοντα Αναφοράς είναι τα ακόλουθα:

Main

Τοπικό: `x, y, z, A` (κλήση), `B` (κλήση)

Μη-Τοπικό και Καθολικό: –

B

Τοπικό: `x, B` (τιμή), `k`

Μη-Τοπικό και Καθολικό: `y, z, A` (κλήση), `B` (κλήση)

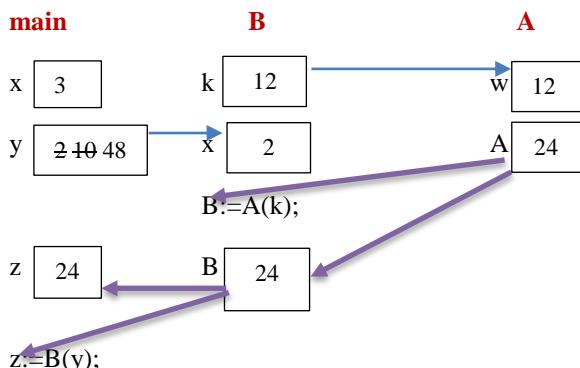
A

Τοπικό: `w, A` (τιμή)

Μη-Τοπικό: `x, B` (τιμή), `k` (από B), `y, z, A, B` (κλήση) (από main)

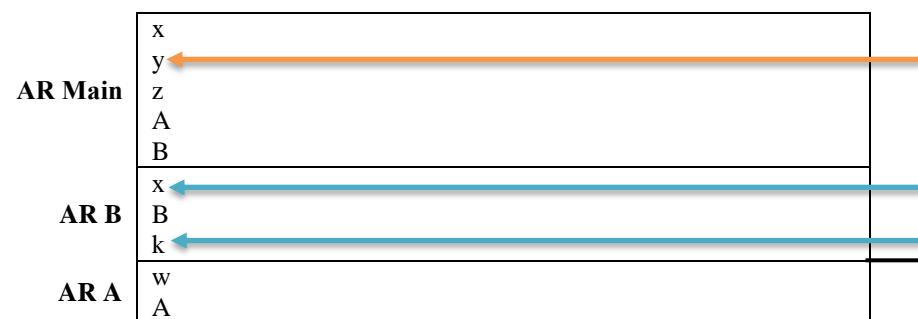
Καθολικό: `A` (κλήση), `B` (κλήση)

- Η εντολή $y:=4*k$ είναι σωστή. Κανονικά πρέπει να τοποθετηθεί πριν την εντολή $A:=x*w$ που καταχωρεί το τελικό αποτέλεσμα στο όνομα της function αλλά και πάλι είναι σωστή
- Η εντολή $x=2*w$ στο main είναι λάθος διότι το main δεν έχει δηλωμένη μεταβλητή w. Άρα η εντολή αυτή είναι η μόνη που διαγράφεται τώρα



Στο main τυπώνονται οι τιμές $y=48$, $z=24$

Στοίβα Εκτέλεσης



1.23 Ασκηση 2 Φεβρουάριος 2018

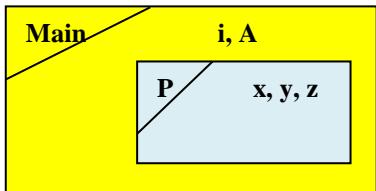
Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

```
program Main;
var i: integer;
A: array [0..1] of integer;
procedure P(x, y:integer);
var z:integer;
begin
z:=x;
x:=y;
i:=0;
y:=z;
end;
BEGIN
i:=1;
A[0]:=0;
A[1]:=2;
P(i, A[i]);
write(i, A[0], A[1]);
END.
```

- (a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- (b) Παρουσιάστε τη στοίβα εκτέλεσης (run-time stack) των εγγραφών ενεργοποίησης (Activation Records-AR) μόλις έχει γίνει η κλήση της P
- (c) Ποια είναι η έξοδος του προγράμματος κατά την εκτέλεση 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)

Απάντηση

(a)



Τα περιβάλλοντα Αναφοράς είναι τα ακόλουθα:

Main

Τοπικό: i, A, P

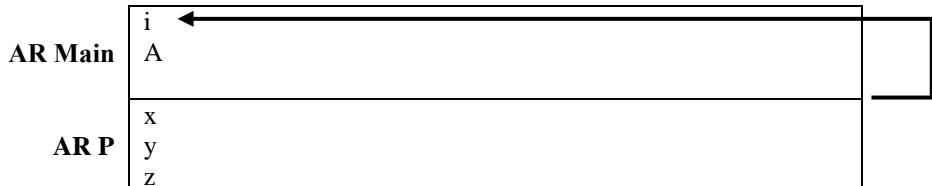
Μη-Τοπικό και Καθολικό: -

P

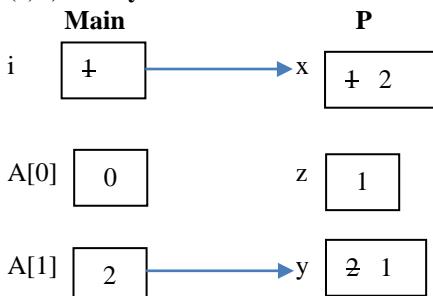
Τοπικό: x, y, z

Μη-Τοπικό και Καθολικό: i, A, P

(b) Στοίβα Εκτέλεσης

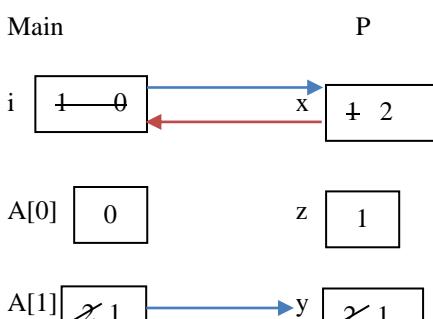


(c) i) call by value



Στο main τυπώνονται οι τιμές 0, 0, 2

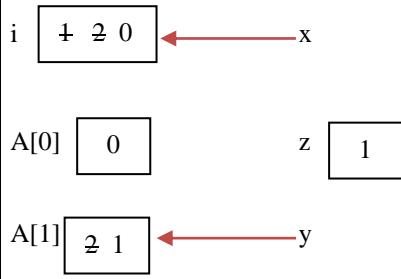
iii) call by value-result



Στο main τυπώνονται οι τιμές 0, 0, 2

ii) call by reference

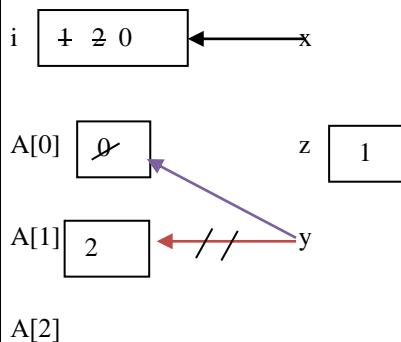
Main P



Στο main τυπώνονται οι τιμές 0, 0, 1

iv) call by name

Main P



Στο main τυπώνονται οι τιμές 0, 0, 1

Παρατήρηση

Στο call by name και αποκλειστικά στους πίνακες όταν αλλάζει η τιμή του i μέσα στις αγκύλες, ο δείκτης που έδειχνε στο στοιχείο αυτό αυτομάτως δείχνει σε μια νέα θέση ανάλογα μ την τιμή του i.

1.24 Θέμα 4 Σεπτέμβριος 2016

Δίνεται ο παρακάτω ορισμός σε μια γλώσσα τύπου Pascal:

```
procedure P;  
x:integer;  
procedure Q;  
begin  
    x:=x+1;  
end;  
procedure R;  
x:integer;  
begin  
    x:=1;  
    Q;  
    write(x);  
end;  
begin  
    x:=2;  
    R;  
    write(x);  
end;
```

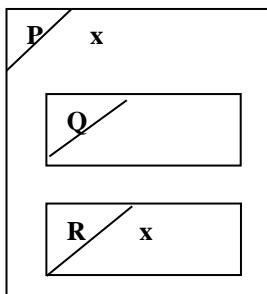
(a) Αν κληθεί η P, τι θα τυπωθεί αν η γλώσσα χρησιμοποιεί (i) το στατικό δυναμικό ή (ii) το δυναμικό κανόνα εμβέλειας. Εξηγείστε τε σύντομα τη λειτουργία της P και στις δύο περιπτώσεις

(b) Στις δύο περιπτώσεις (στατικού και δυναμικού κανόνα εμβέλειας) ποιο είναι το τοπικό περιβάλλον αναφοράς της P και ποια τα τοπικά και μη-τοπικά περιβάλλοντα αναφοράς των διαδικασιών Q και R;

Απάντηση

(b)

Στατικός Κανόνας Εμβέλειας



Περιβάλλοντα Αναφοράς

P

Τοπικό: x, Q, R

Μη-Τοπικό και Καθολικό: —

Q

Τοπικό: —

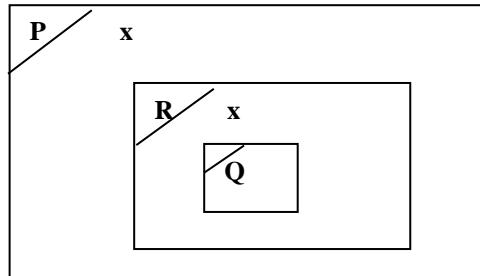
Μη-Τοπικό και Καθολικό: x, Q, R

R

Τοπικό: x

Μη-Τοπικό και Καθολικό: Q, R

Δυναμικός Κανόνας Εμβέλειας



Περιβάλλοντα Αναφοράς

P

Τοπικό: x, Q, R

Μη-Τοπικό και Καθολικό: —

Q

Τοπικό: —

Μη-Τοπικό: x (από R), Q, R(από P)

Καθολικό: Q, R

R

Τοπικό: x

Μη-Τοπικό και Καθολικό: Q, R

(α)

Στατικός Κανόνας Εμβέλειας

P	R	Q	Εκτύπωση
x ✓ 3	x 1		Εκτύπωση τοπικής μεταβλητής x της R με τιμή 1

H write τυπώνει 1 στην R και 3 στην P στο Στατικό Κανόνα Εμβέλειας

Δυναμικός Κανόνας Εμβέλειας

P	R	Q	Εκτύπωση
x 2	x + 2		Εκτύπωση τοπικής μεταβλητής x της R με τιμή 2

H write τυπώνει 2 στην R και 2 στην P στο Δυναμικό Κανόνα Εμβέλειας

1.25 Θέμα 2 Ιούνιος 2017

Δίνεται το ακόλουθο πρόγραμμα σε μια γλώσσα τύπου Pascal:

```
program MAIN;  
  
var x, y: integer;  
  
procedure F(a, b, c: integer);  
  
begin  
    b:=b+5;  
    b:=a+c+4;  
    write(a, b, c);  
  
end;  
  
BEGIN  
  
    x:=1;  
    y:=2;  
    F(x, x, x+y);  
    write(x, y);  
  
END.
```

Τι τυπώνεται με την εκτέλεση του προγράμματος στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων; Εξηγήστε σύντομα τις διαδικασίες υπολογισμού σε όλες τις περιπτώσεις:

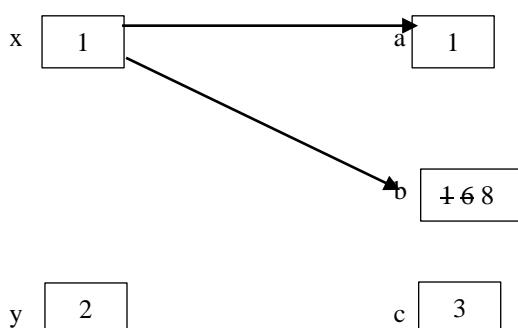
- (a) Όλες οι παράμετροι μεταβιβάζονται με κλήση με τιμή (call by value)
- (b) Τα a και b μεταβιβάζονται με κλήση με αναφορά (call by reference) και το c με κλήση με τιμή
- (c) Τα a και b μεταβιβάζονται με κλήση με τιμή-αποτέλεσμα (call by value-result) και το c με κλήση με τιμή
- (d) Όλες οι παράμετροι μεταβιβάζονται με κλήση με όνομα (call by name)

Απάντηση

- (a) Όλες οι παράμετροι κλήση με τιμή (call by value)

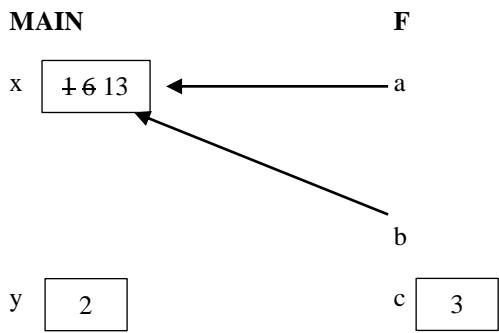
MAIN

F



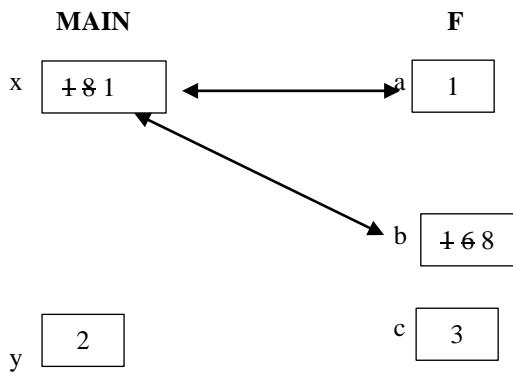
Τυπώνονται οι τιμές a=1, b=8, c=3 μέσα στην procedure F και οι τιμές x=1, y=2 στο MAIN

(b) Τα a και b μεταβιβάζονται με κλήση με αναφορά (call by reference) και το c με κλήση με τιμή



Τυπώνονται οι τιμές a=13, b=13, c=3 στην procedure F και οι τιμές x=13, y=2 στο MAIN

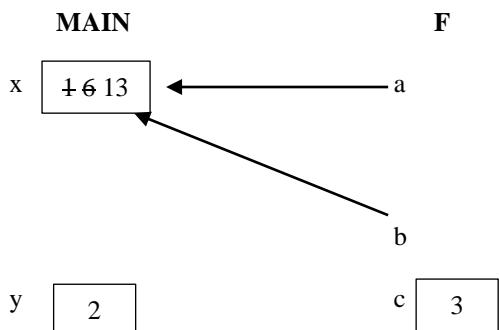
(c) Τα a και b μεταβιβάζονται με κλήση με τιμή-αποτέλεσμα (call by value-result) και το c με κλήση με τιμή



Τυπώνονται οι τιμές a=1, b=8, c=3 μέσα στην procedure F και οι τιμές x=1, y=2 στο MAIN

(πρώτα επιστρέφει πίσω το b στη x και μετά επιστρέφει πίσω το a στη x)

(d) Όλες οι παράμετροι μεταβιβάζονται με κλήση με όνομα (call by name)



Τυπώνονται οι τιμές a=13, b=13, c=3 στην procedure F και οι τιμές x=13, y=2 στο MAIN

(Στο call by name η παράμετρος x μεταβιβάζεται με call by reference ενώ η παράμετρος x+y που είναι έκφραση μεταβιβάζεται με call by value)

1.26 Θέμα 3 Φεβρουάριος 2019

Δίνεται ο παρακάτω ορισμός σε μια γλώσσα που χρησιμοποιεί στατικό κανόνα εμβέλειας:

```

int A[3];
int i;
void main()
{
    i= 1;
    A[1] = 2;
    A[2] = 4;
    Q(A[i]);
    write ("A[1] = ", A[1]);
    write ("A[2] = ", A[2]);
}
void Q (int B)
{
    A[1] = 3;
    i= 2;
    write ("B=", B);
    B = 5;
}

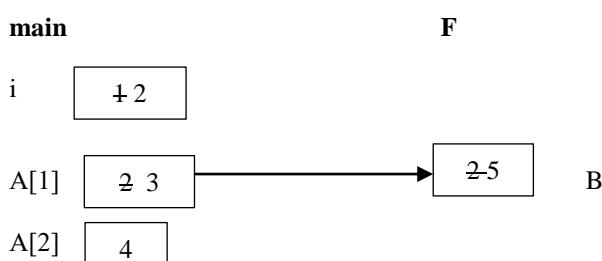
```

Να παρουσιάσετε την έξοδο του προγράμματος κατά την εκτέλεσή του, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων, παρουσιάζοντας και τις αλλαγές μεταβλητών σε κάθε περίπτωση

1. Κλήση με τιμή (call by value)
2. Κλήση με αναφορά (call by reference)
3. Κλήση με τιμή – αποτέλεσμα (call by value – result)
4. Κλήση με όνομα (call by name)

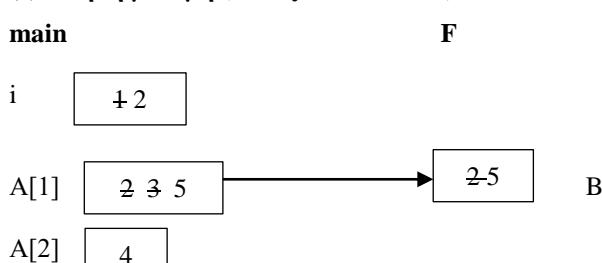
Απάντηση

(a) Κλήση με τιμή (call by value)



Τυπώνονται οι τιμές $B=2$, $A[1]=3$ και $A[2]=4$

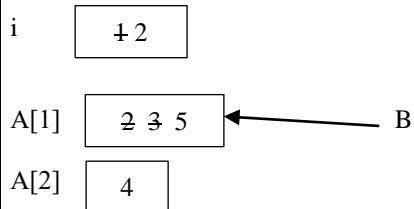
(c) Κλήση με τιμή (call by value-result)



Τυπώνονται οι τιμές $B=2$, $A[1]=5$ και $A[2]=4$

(b) Κλήση με αναφορά (call by reference)

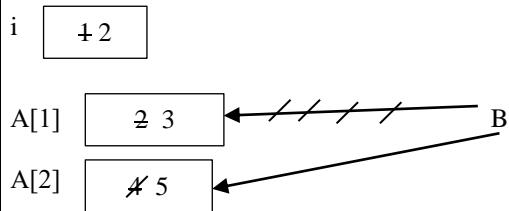
main F



Τυπώνονται οι τιμές $B=3$, $A[1]=5$ και $A[2]=4$

(d) Κλήση με όνομα (call by name)

main F



Τυπώνονται οι τιμές $B=4$, $A[1]=3$ και $A[2]=5$

1.27 Φροντιστήριο 2019

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας;

program MAIN;

var y, z: **integer**;

function f(a: **integer**): **integer**;

begin

 y:= a+2;

 a:=x;

 f:=y+a;

end;

function g(x: **integer**): **integer**;

begin

 y:= f(x+1);

 z:= f(y-x);

 x:= a+1;

 g:= z+3

end;

BEGIN

 y:= 4;

 z:= g (2*y);

write (y, z)

END.

- (a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη – τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- (b) Στον παραπάνω κώδικα υπάρχουν δύο εντολές οι οποίες δεν είναι σωστές από σημασιολογική άποψη και θα προκαλέσουν run-time error. Ποιες είναι αυτές και γιατί; Σβήστε τις δύο αυτές εντολές από τον κώδικα.
- (c) Παρουσιάστε τη στοίβα εκτέλεσης (run – time stack) των εγγραφών ενεργοποίησης (Activation Records – AR) μόλις έχει γίνει 2^n κλάση της f.
- (d) Ποιες τιμές θα τυπωθούν στο τέλος για τα y και z στις παρακάτω δύο περιπτώσεις μεταβίβασης παραμέτρων; Παρουσιάστε τις μεταβολές των μεταβλητών και παραμέτρων.
1. Κλήση με τιμή (call by value)
 2. Κλήση με όνομα (call by name)

- (e) Απαντήστε στα (b) και (d) θεωρώντας ότι ισχύει ο δυναμικός κανόνας εμβέλειας. Τώρα, μόνο η μία από τις δύο εντολές δεν είναι σωστή από σημασιολογική άποψη και σβήνετε μόνο αυτήν. Ποια είναι η εντολή που είναι τώρα σωστή και γιατί;

1.28 Θέμα 2 Σεπτέμβριος 2018

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας;

program MAIN;

var x, y, z: **integer**;

function A(w: **integer**): **integer**;

begin

w:=10;

y:= 4*k;

A:= x*w

end;

function B(x: **integer**): **integer**;

var k: **integer**;

begin

k:= 10;

x:= 12;

k:= 2*w;

B:= A(x)

end;

BEGIN

x:= 4; y:= 2;

z:= B(y);

write (z) ; **write** (y)

END.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη – τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

(b) Στον παραπάνω κώδικα υπάρχουν δύο εντολές οι οποίες δεν είναι σωστές από σημασιολογική άποψη και θα προκαλέσουν run-time error. Ποιες είναι αυτές και γιατί; Σβήστε τις δύο αυτές εντολές από τον κώδικα.

(c) Παρουσιάστε τη στοίβα εκτέλεσης (run – time stack) των εγγραφών ενεργοποίησης (Activation Records – AR) μόλις έχει γίνει η κλάση της A.

(d) Ποιες τιμές θα τυπωθούν στο τέλος για τα y και z στις παρακάτω δύο περιπτώσεις μεταβίβασης παραμέτρων; Παρουσιάστε τις μεταβολές των μεταβλητών και παραμέτρων.

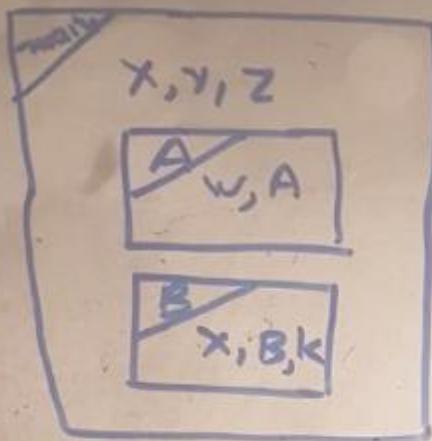
1. Κλήση με τιμή (call by value)
2. Κλήση με αναφορά (call by reference)

Απαντήστε στα παραπάνω ερωτήματα θεωρώντας ότι ισχύει ο δυναμικός κανόνας εμβέλειας. Τώρα, μόνο η μία από τις δύο εντολές δεν είναι σωστή από σημασιολογική άποψη και σβήνετε μόνο αυτήν. Ποια είναι η εντολή που είναι τώρα σωστή και γιατί;

Απάντηση

oefeningen vanwaar een vergelijking

a)



maar

Tonud: x, y, z, A, B
Nu-Tonud u'keDgud: —

A

Tonud: w, A (prees)
Nu-Tonud u'keDgud:
 x, y, z, A (couppen),
B

B

Tonud: x, y, B (preaguer)

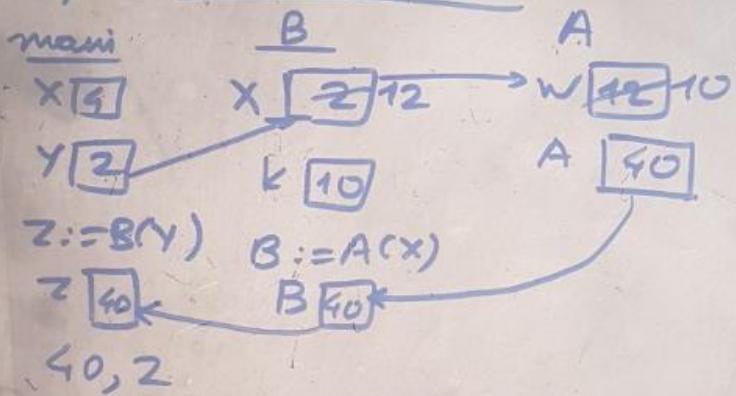
Nu-Tonud u'keDgud: y, z, A, B (and
s)

b)

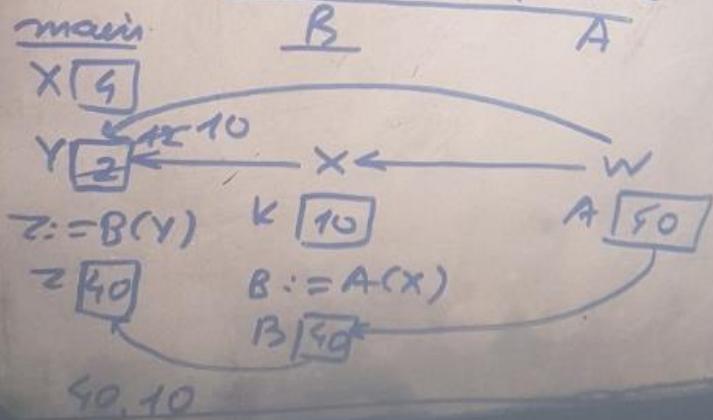
It vraagt $y = 4 \times k$ van function A
en dat doet de ene voorbeeld
dus juist dan k van A

It vraagt $k = 2 \times w$ van function
B en dat doet de ene voorbeeld
van B dus juist dan w

d) call by value

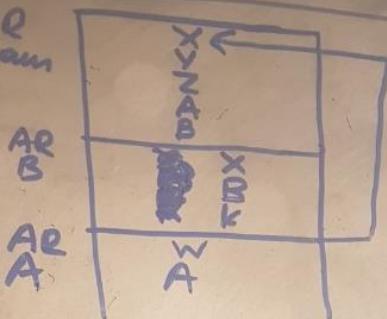


call by reference



5

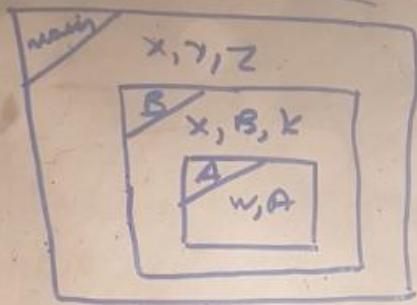
AR
man



020189
EXTERIORS

Διαδικούσα Καρέκλα Γενετής

a)



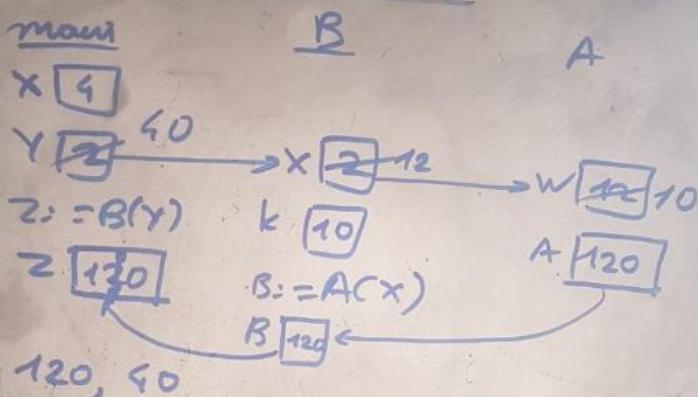
Main: Τοπίο: x, y, z, A, B
Μη-Τοπίο που χρησιμοποιείται: —

B: Τοπίο: x, k, B (μη τοπίο)
Μη-Τοπίο που χρησιμοποιείται: y, z, A, B (καθορισμένο)

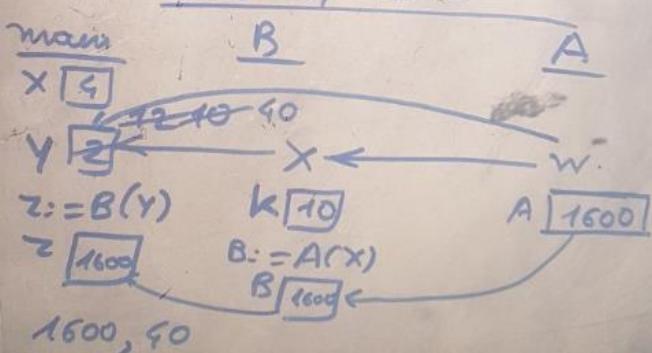
A: Τοπίο: w, A (μη τοπίο)
Μη-Τοπίο: x, k, B (από B)
 y, z, A, B (από μάκιο)
Καθορισμένο: A, B

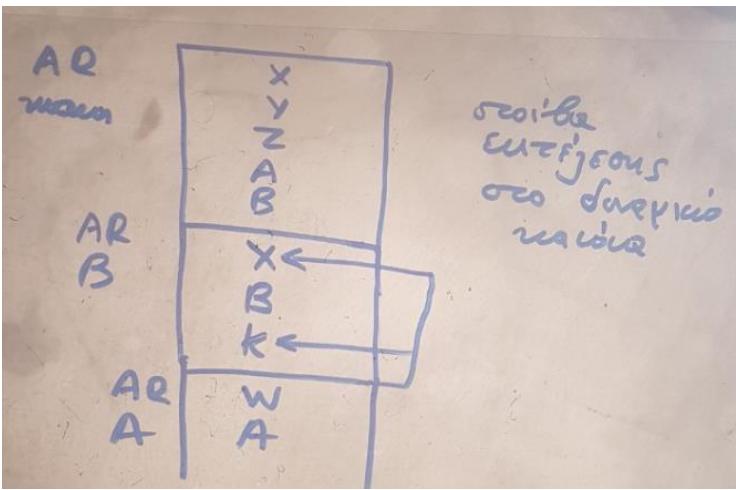
B) Μάλιστα στην γραφή $k := 2 = w$ σημαίνει
function B επιστρέφει 200 , δηλαδή
διανομή $= k$ στην function B

c) call by value



call by reference





1.29 Θέμα 3- Ιούνιος 2019 -Ομάδα Β

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας;

program MAIN;

var i: integer;

A: array [1..2] of **integer**;

procedure F (x, y, z: **integer**);

begin

x:= x+1;

y:= z;

z:= z+1;

end;

BEGIN

i:= 1;

A [1]:= 10; A[2]:= 11;

x:= i;

F(i, A[i], i);

write (i, A[1], A[2]);

END.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη – τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

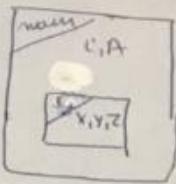
(b) Στον παραπάνω κώδικα υπάρχει μια εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error. Ποια είναι αυτή και γιατί; "Σβήστε" την εντολή αυτή από τον κώδικα.

(c) Τι τυπώνεται στο τέλος με την εντολή write, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία αυτή);

- 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)

Απάντηση

Opfer 3 - OMADA B



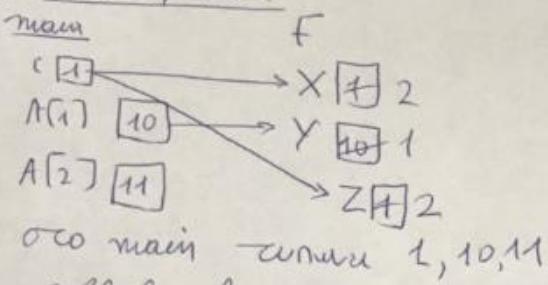
a) Перевіссажа Аарре'

Main: Томас i,A,F
Ми-Томас вар. val. : —

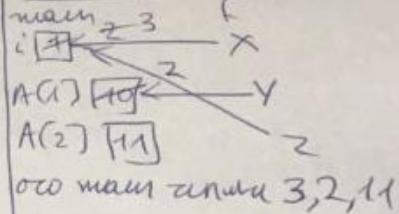
F Томас x,y,z
Ми-Томас вар. val. : i,A,F

b) If assign $x := i$ dual $\neq D_0$, oco MAIN var
дивергент

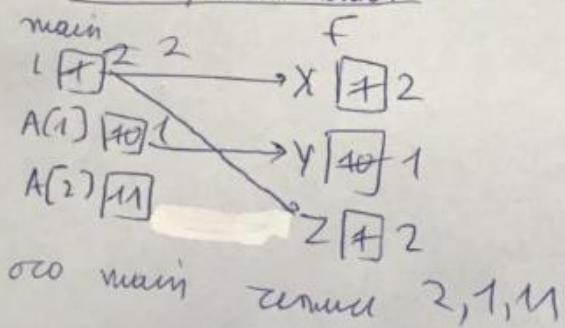
8) call by value



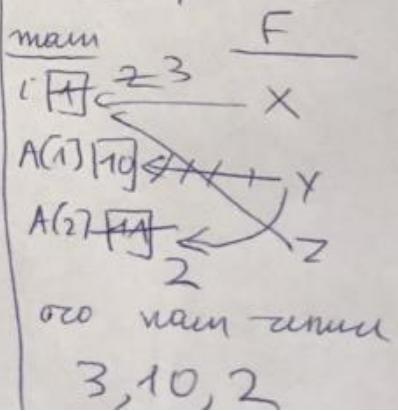
call by reference



call by value-result



call by name



1.30 Θέμα 3- Σεπτέμβριος 2019 -Ομάδα Α

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας:

program MAIN;

var z: integer;

 a: array [1..2] of integer;

procedure P(x: integer);

begin

 a[1]:= 6;

 z:= 2;

 x:= x + 5

end;

BEGIN

 a[1]:= 2; a[2]:= 3;

 z:= 1; x:= 2;

 P(a[z]);

write(a[1], a[2], z)

END.

(a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

(b) Στον παραπάνω κώδικα υπάρχει μία εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error. Ποια είναι αυτή και γιατί; "Σβήστε" την εντολή αυτή από τον κώδικα.

(c) Τι τυπώνεται στο τέλος με την εντολή write, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού);

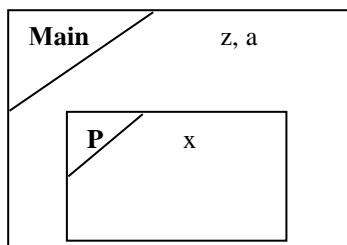
i) Κλήση με τιμή (call by value)

ii) Κλήση με αναφορά (call by reference)

iii) Κλήση με τιμή – αποτέλεσμα (call by value – result)

iv) Κλήση με όνομα (call by name)

Απάντηση



Main

Τοπικό: z, a, P

Μη Τοπικό και Καθολικό: -

P

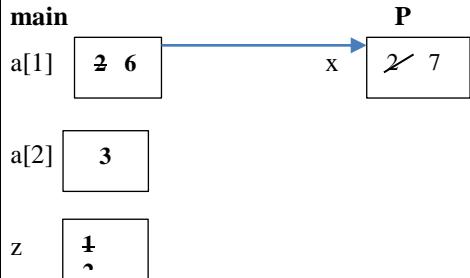
Τοπικό: x

Μη Τοπικό και Καθολικό: z, a, P

(b) Η λάθος εντολή είναι η x:=2;

c)

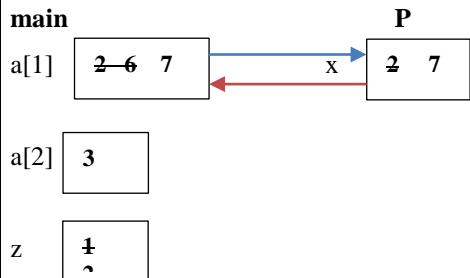
call by value



call by value : Όταν τερματίσει η συνάρτηση P η παράμετρος x δεν επιστρέφεται στο main.

Οι τιμές που τυπώνονται στο main είναι: 6, 3, 2

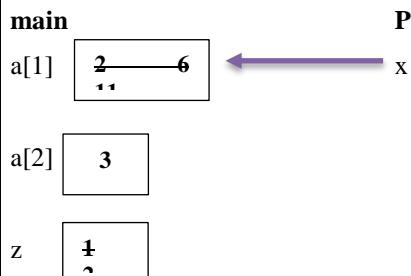
call by value result



call by value result: Όταν τερματίσει η συνάρτηση P η παράμετρος x επιστρέφεται πίσω στο main στο στοιχείο a[1].

Οι τιμές που τυπώνονται στο main είναι: 7, 3, 2

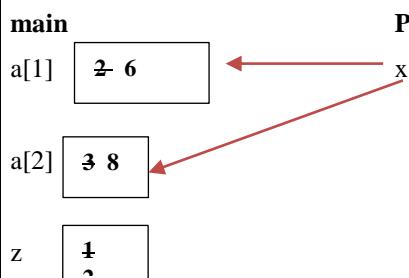
call by reference



call by reference: Όταν τερματίσει η συνάρτηση P δεν επιστρέφεται τίποτα.

Οι τιμές που τυπώνονται στο main είναι: 11, 3, 2

call by name → Ο δείκτης δείχνει αρχικά στο a[1]. Όταν το z γίνει 2 τότε ο x δείχνει στο a[2]. Οι τιμές στο main είναι: 6, 8, 2



call by name: Ο δείκτης δείχνει αρχικά στο a[1]. Όταν το z γίνει 2 τότε ο x δείχνει στο a[2].

Οι τιμές που τυπώνονται στο main είναι: 6, 8, 2

1.31 Θέμα 1- Φεβρουάριος 2020

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί **στατικό κανόνα εμβέλειας**:

program MAIN;

var x, y: **integer**;

function f(x: **integer**): **integer**;

begin

 x := x+1;

 y := x;

 x := x+1;

 f := y;

end;

function g(x: **integer**): **integer**;

begin

 y := f(x) + 1;

 x := f(y) + 3;

 g := x;

end;

BEGIN

 y := 7;

 z := g(y);

write (y, z);

END.

(α) Ποια είναι τα **περιβάλλοντα αναφοράς** (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;

(β) Παρουσιάστε τη **στοίβα εκτέλεσης** (run-time stack) των εγγραφών ενεργοποίησης (Activation Records – AR) μόλις έχει γίνει η δεύτερη κλήση της f.

(γ) Ποια είναι η **έξοδος του προγράμματος** κατά την εκτέλεσή του, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού):

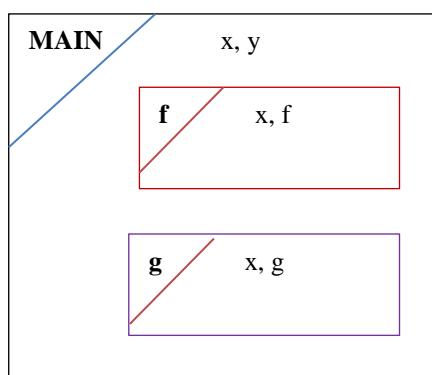
i) Κλήση με τιμή (call by value)

ii) Κλήση με αναφορά (call by reference)

iii) Κλήση με τιμή – αποτέλεσμα (call by value – result)

Απάντηση

(α)



Περιβάλλοντα Αναφοράς

Main

Τοπικό: x, y, f (κλήση), g (κλήση)

Μη-Τοπικό και Καθολικό: –

f

Τοπικό: x, f (τιμή)

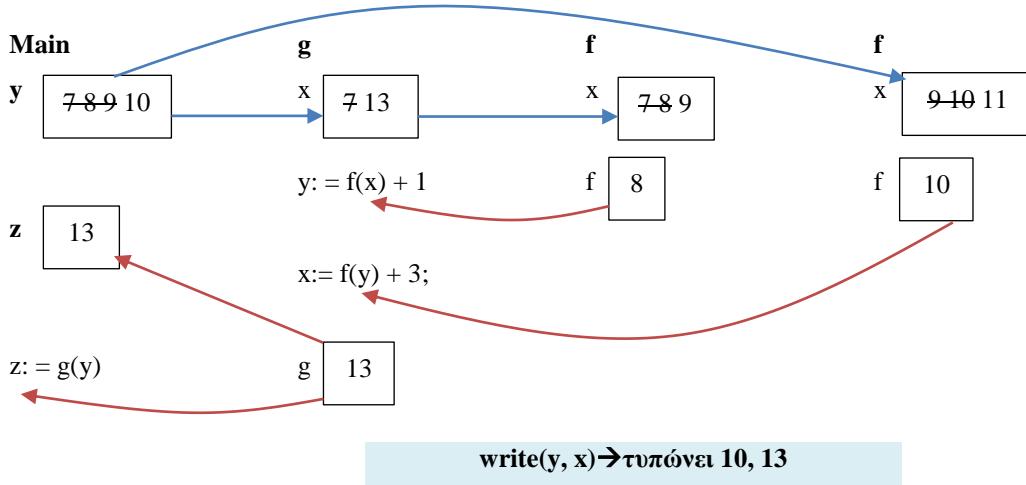
Μη-Τοπικό και Καθολικό: y, f (κλήση), g (κλήση)

g

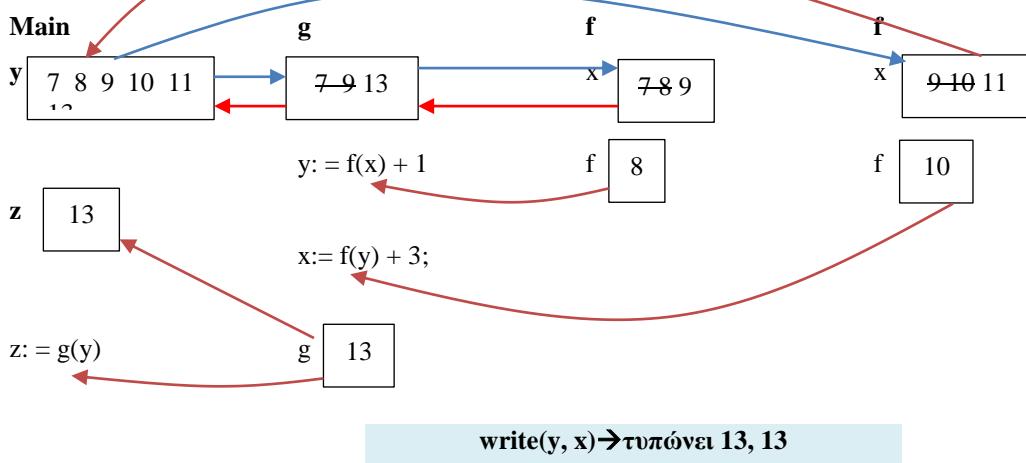
Τοπικό: x, g (τιμή)

Μη-Τοπικό και Καθολικό: y, f (κλήση), g (κλήση)

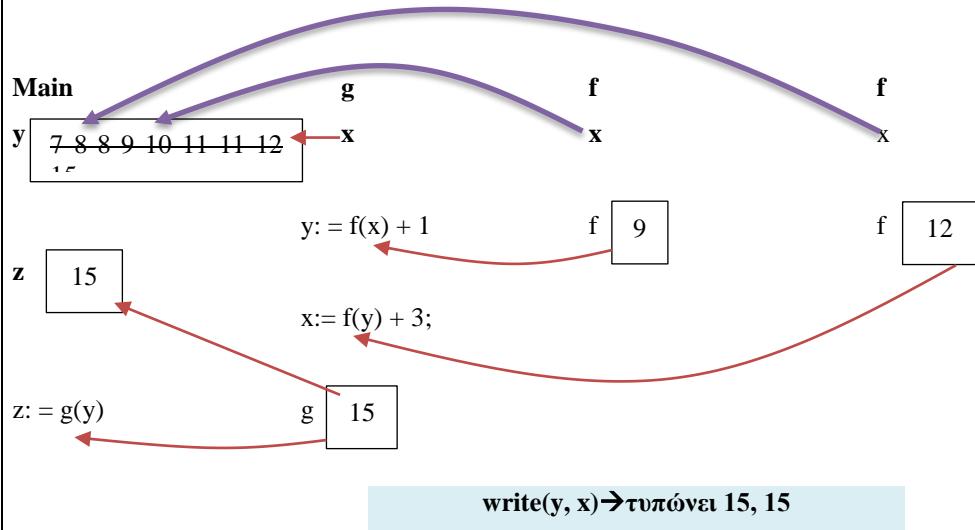
Έξοδος με call by value



Έξοδος με call by value-result



Έξοδος με call by reference



Η στοίβα εκτέλεσης στην 1^η κλήση της f

AR main	x y f g
AR g	x g
AR f	x f

Όταν τελειώσει η f το AR της f αφαιρείται από τη στοίβα εκτέλεσης

AR main	x y f g
AR g	x g

Η στοίβα εκτέλεσης στη 2^η κλήση της f

AR main	x y f g
AR g	x g
AR f	x f

1.32 Θέμα 2- Ιούνιος 2020

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal **στην οποία η μεταβίβαση παραμέτρων γίνεται με αναφορά** (call by reference).

program A

var K, L: integer;

procedure SUB1 (var J: integer);

begin

 J:= J+1;

print("J=", J);

end;

procedure SUB2;

begin

 SUB1(K);

 SUB1(L);

end;

procedure SUB3;

var L: integer;

begin

 L:= 8;

 SUB2;

end;

BEGIN

 K:= 3;

 L:= 4;

 SUB3;

print("K=", K);

print("L=", L);

END.

Για την περίπτωση που ισχύει στατικός κανόνας εμβέλειας, ζητούνται:

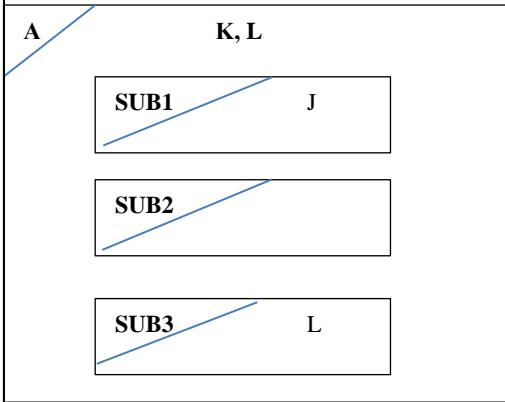
a.Να ορίσετε τα **περιβάλλοντα αναφοράς** (τοπικά, μη- τοπικά και καθολικά), όλων των τμημάτων του προγράμματος.

b.Να **παρουσιάσετε τα αποτελέσματα που θα εμφανιστούν στην οθόνη** του υπολογιστή κατά την εκτέλεση του προγράμματος (με τη σωστή, χρονικά, σειρά). **Παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών** κατά τη διαδικασία υπολογισμού.

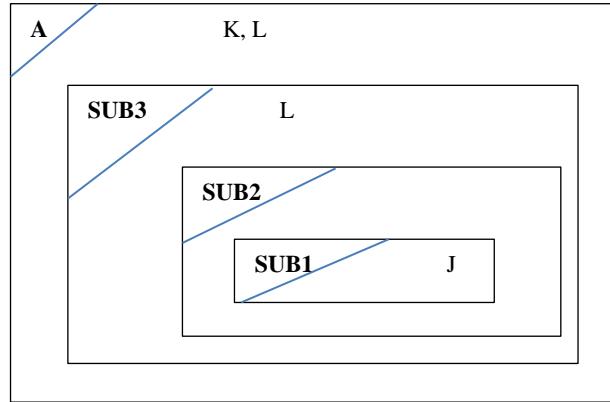
Να απαντήσετε στα παραπάνω ερωτήματα (a) και (b), υποθέτοντας ότι η γλώσσα ακολουθεί δυναμικό κανόνα εμβέλειας

Απάντηση

Στατικός Κανόνας Εμβέλειας



Δυναμικός Κανόνας Εμβέλειας



A

Τοπικό: K, L, SUB1, SUB2, SUB3

Μη-Τοπικό και Καθολικό: –

SUB1

Τοπικό: J

Μη-Τοπικό και Καθολικό: K, L, SUB1, SUB2, SUB3

SUB2

Τοπικό: –

Μη-Τοπικό και Καθολικό: K, L, SUB1, SUB2, SUB3

SUB3

Τοπικό: L

Μη-Τοπικό και Καθολικό: K, SUB1, SUB2, SUB3

A

Τοπικό: K, L, SUB1, SUB2, SUB3

Μη-Τοπικό και Καθολικό: –

SUB1

Τοπικό: J

Μη-Τοπικό: L (από SUB3), K, SUB1, SUB2, SUB3 (από A)

Καθολικό: SUB1, SUB2, SUB3

SUB2

Τοπικό: –

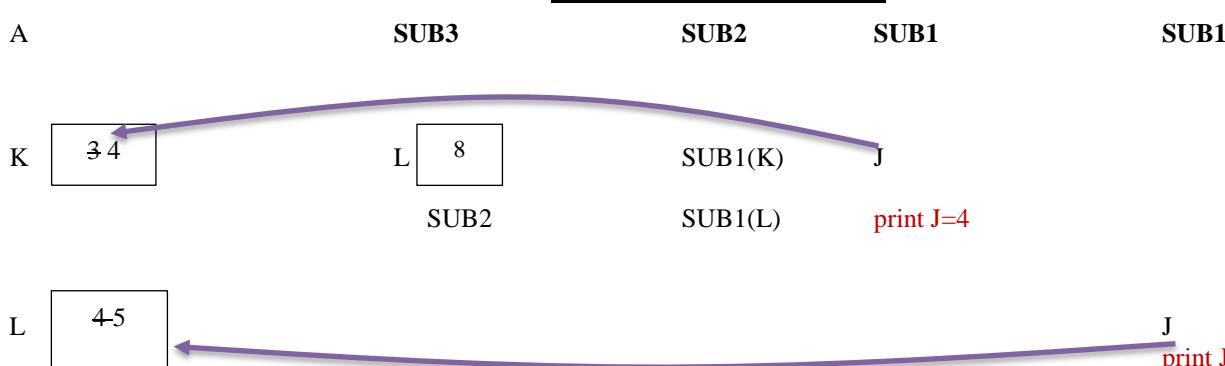
Μη-Τοπικό: L (από SUB3), K, SUB1, SUB2, SUB3 (από A)

Καθολικό: SUB1, SUB2, SUB3

SUB3

Τοπικό: L

Μη-Τοπικό και Καθολικό: K, SUB1, SUB2, SUB3



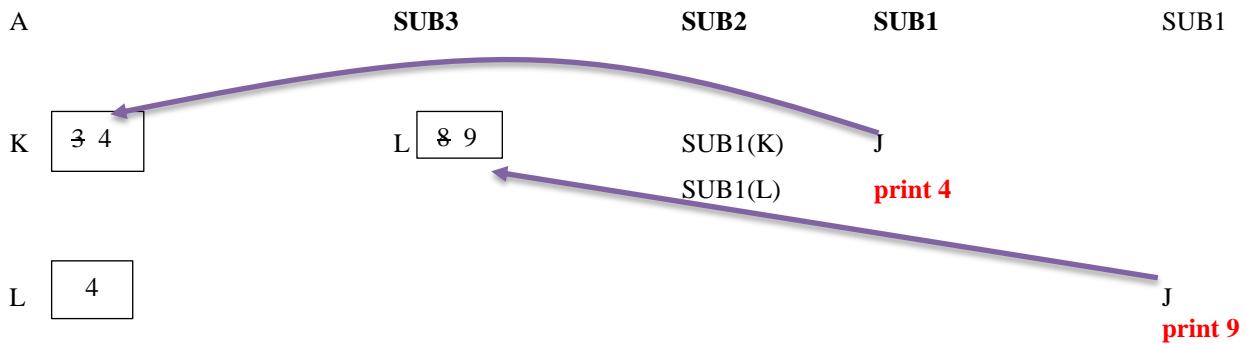
Στο main τυπώνονται οι τιμές K=4 και L=5

Η χρονική σειρά εμφάνισης των αποτελεσμάτων είναι:

J=4
J=5
K=4
L=5

Παρατήρηση: Στη sub2 τα K, L είναι από το main (A) διότι αυτό είναι το περιβάλλον μπλοκ.

Δυναμικός Κανόνας Εμβέλειας



Στο main τυπώνονται οι τιμές K=4 και L=4

Η χρονική σειρά εμφάνισης των αποτελεσμάτων είναι:

J=4
J=9
K=4
L=4

Παρατήρηση: Στη sub2 το K είναι από το main και το L από το sub3 διότι αυτά είναι τα περιβάλλοντα μπλοκ.

1.33 Θέμα 2 Φεβρουάριος 2021

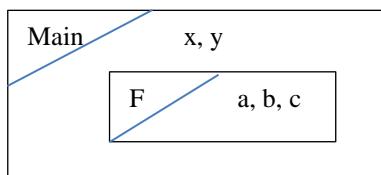
Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

- (a) Ποια είναι τα περιβάλλοντα αναφοράς (Τοπικά, Μη-τοπικά, Καθολικά) όλων των τμημάτων του προγράμματος;
- (b) Στο παραπάνω πρόγραμμα υπάρχει μια εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run time error. Ποια είναι αυτή και γιατί; Θεωρήστε για τη συνέχεια ότι δεν υπάρχει η εντολή αυτή
- Τι τυπώνεται με την εκτέλεση του προγράμματος στις παρακάτω περιπτώσεις μεταβίβασης παραμέτρων. Εξηγήστε σύντομα τις διαδικασίες υπολογισμού σε όλες τις περιπτώσεις
- (c) Όλες οι παράμετροι μεταβιβάζονται με **κλήση με τιμή** (call by value)
- (d) Τα a, b μεταβιβάζονται με **κλήση με αναφορά** (call by reference) και το c με κλήση με τιμή (call by value)
- (e) Τα a, b μεταβιβάζονται με κλήση με **τιμή-αποτέλεσμα** (call by value-result) και το c με κλήση με τιμή (call by value)
- (f) Όλες οι παράμετροι μεταβιβάζονται με κλήση με **όνομα** (call by name)

```
program MAIN;
  var x, y: integer;
  procedure F(a, b, c: integer);
    begin
      b:= b + 5;
      b:= a + c + 4;
      write(a, b, c)
    end;
  BEGIN
    x:= 4;
    y:= 6;
    c:= 8;
    F(x, x, x + y);
    write(x, y)
  END.
```

Απάντηση

- (a) Περιβάλλοντα Αναφοράς



Main

Τοπικό: x, y, F (κλήση)

Μη-Τοπικό και Καθολικό: –

F

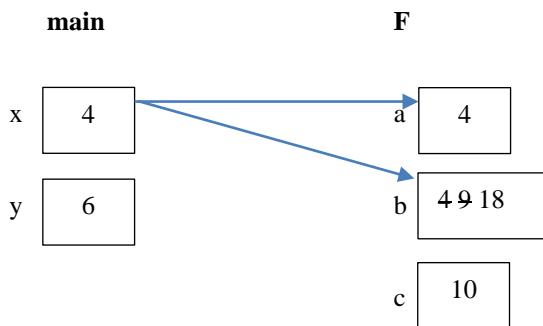
Τοπικό: a, b, c

Μη-Τοπικό και Καθολικό: x, y, F (κλήση)

(b)

Η εντολή c:=8; δεν είναι σωστή από σημασιολογική άποψη **διότι στο main δεν δηλώνεται μεταβλητή c**.

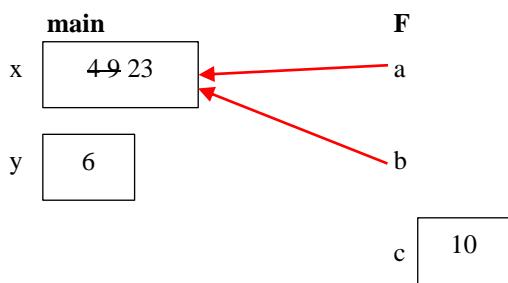
(c) Όλες οι παράμετροι μεταβιβάζονται με κλήση με τιμή (call by value)



Στην Procedure F τυπώνει 4, 18, 10

Στο main τυπώνει 4, 6

(d) Τα **a, b** μεταβιβάζονται με κλήση με αναφορά (call by reference) και το **c** με κλήση με τιμή (call by value)



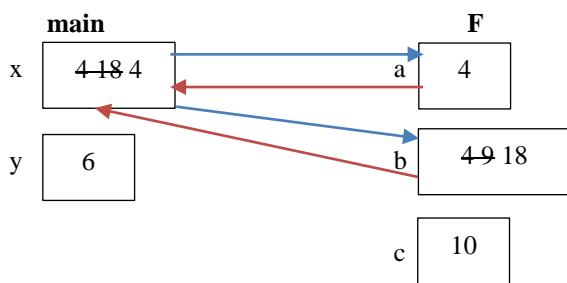
Στην Procedure F τυπώνει 23, 23, 10

Στο main τυπώνει 23, 6

Παρατήρηση

Επειδή μεταβιβάζεται η διεύθυνση της **x** και στο δείκτη **a** και στο δείκτη **b** και οι δύο δείκτες **a**, **b** δείχνουν στην ίδια θέση μνήμης δηλ. στο **x** του **main** και αυτό ονομάζεται **φαινόμενο ψευδωνυμίας**

(e) Τα **a, b** μεταβιβάζονται με κλήση με τιμή-αποτέλεσμα (call by value-result) και το **c** με κλήση με τιμή (call by value)

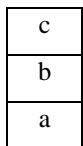


Στην Procedure F τυπώνεται 4, 18, 10

Στο main τυπώνεται 4, 6

Παρατήρηση

Στοίβα με παραμέτρους υποπρογράμματος

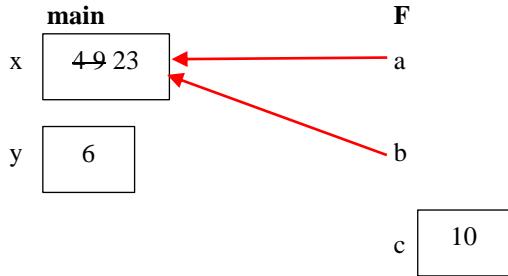


Κατά τον τερματισμό της συνάρτησης με τη μέθοδο LIFO πρώτα επιστρέφεται πίσω η παράμετρος **c** και δεν καταχωρείται κάπου

Μετά επιστρέφεται η παράμετρος b και καταχωρείται στο x και το κάνει 18

Μετά επιστρέφεται η παράμετρος a και καταχωρείται στο x και το κάνει 4

(f) Όλες οι παράμετροι μεταβιβάζονται με κλήση με όνομα (call by name)



Στην Procedure F τυπώνει 23, 23, 10

Στο main τυπώνει 23, 6

Παρατήρηση

Στο call-by-name τυπώνονται τα ίδια αποτελέσματα με το call by reference διότι η παράμετρος x μεταβιβάζεται και στο a και στο b με call by reference διότι είναι απλή μεταβλητή ενώ η έκφραση x+y μεταβιβάζεται στο c με call by-value διότι είναι παράσταση.

Γενικά στο call-by-name όταν μεταβιβάζουμε απλές μεταβλητές το call-by-name=call-by-reference

Γενικά στο call-by-name όταν μεταβιβάζουμε παραστάσεις ή σταθερές το call-by-name=call-by-value

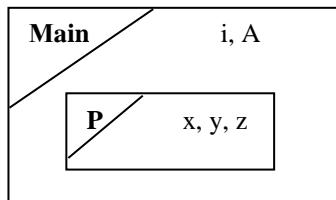
1.34 Θέμα 2 Σεπτέμβριος 2021

- a. Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη τοπικά και καθολικά) όλων των τμημάτων του προγράμματος;
- b. Στο πρόγραμμα **υπάρχει μια εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει error**. Ποια είναι και γιατί; Σβήστε την εντολή από τον κώδικα
- d. Ποια είναι η έξοδος του προγράμματος κατά την εκτέλεση του στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων;
Παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού.
- κλήση με τιμή (call by value)
 - κλήση με τιμή-αποτέλεσμα (call by value-result)
 - κλήση με αναφορά (call by reference)
 - κλήση με όνομα (call by value name)

```
program MAIN;
var i: integer;
    A: array [1..4] of integer;
procedure P(x, y: integer);
var z: integer;
begin
    z:= x;
    x:= y;
    i:= 4;
    y:= z;
end;
BEGIN
    i:= 2;
    y:= 3;
    A[1]:= 0;
    A[2]:= 1;
    A[3]:= 2;
    A[4]:= 3;
    P(i, A[i]);
    write(i, A[1], A[2], A[3], A[4]);
END.
```

Απάντηση

a. Περιβάλλοντα Αναφοράς



Main

Τοπικό: i, a, P

Μη Τοπικό και Καθολικό: -

P

Τοπικό: x, y, z

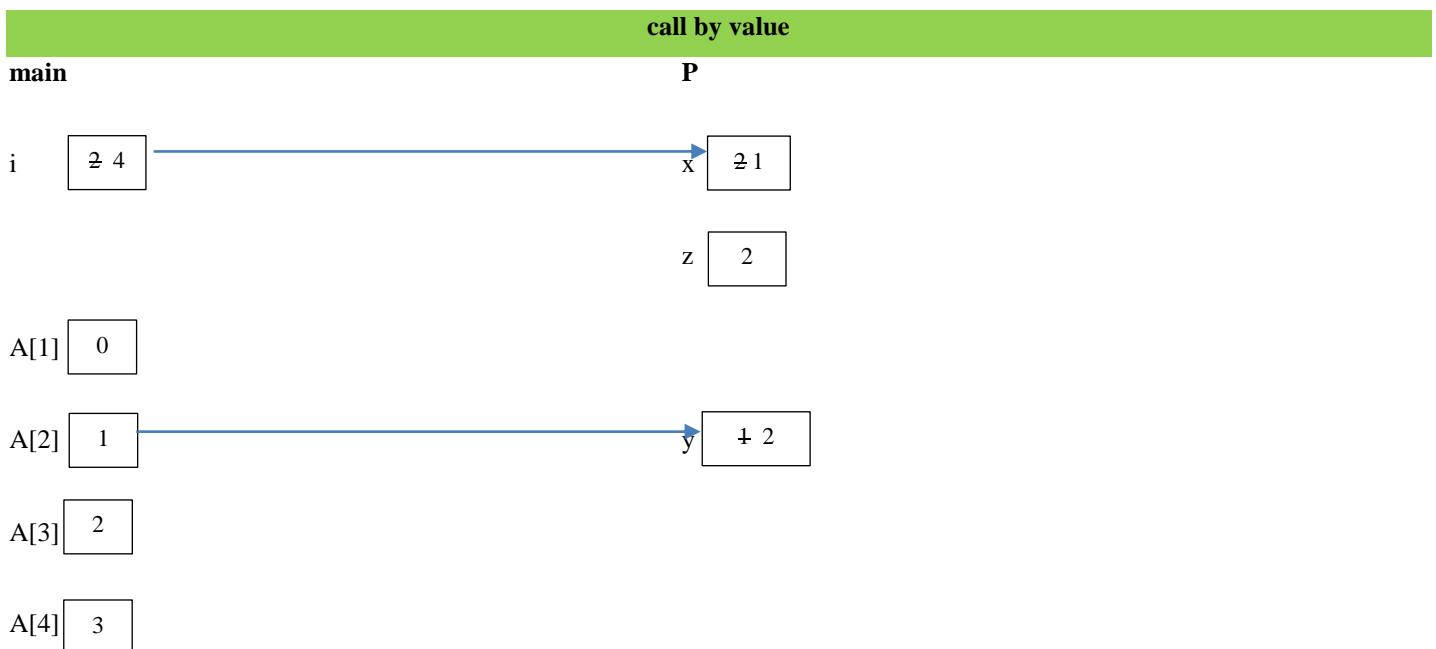
Μη Τοπικό και Καθολικό: i, a, P

b.

Στο MAIN η εντολή $y := 3$; είναι ΛΑΘΟΣ διότι δεν έχει δηλωθεί μεταβλητή γ στο MAIN. Η εντολή $y := 3$ διαγράφεται από το MAIN

c.

i.

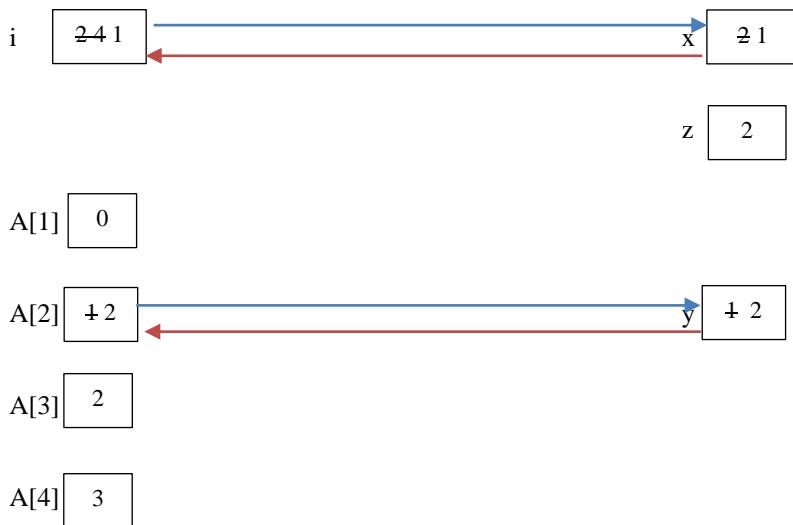


ii.

call by value-result

main

P



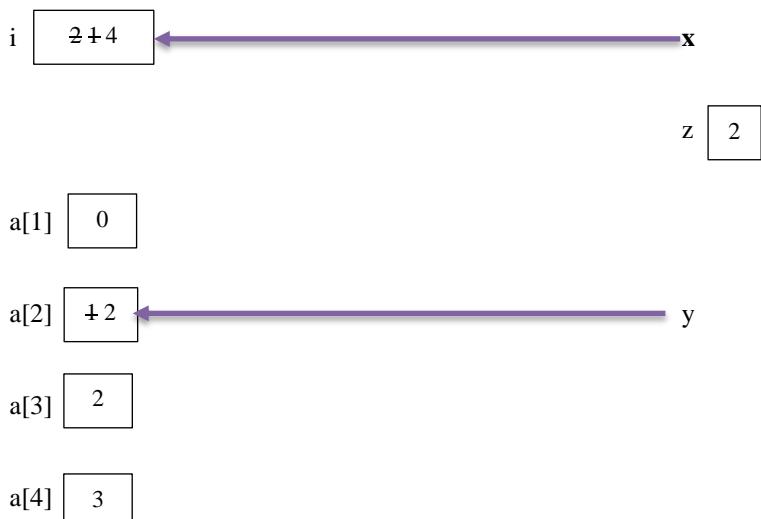
Η εντολή write στο main τυπώνει $i=1, A[1]=0, A[2]=2, A[3]=2, A[4]=3$

iii.

call by reference

main

P



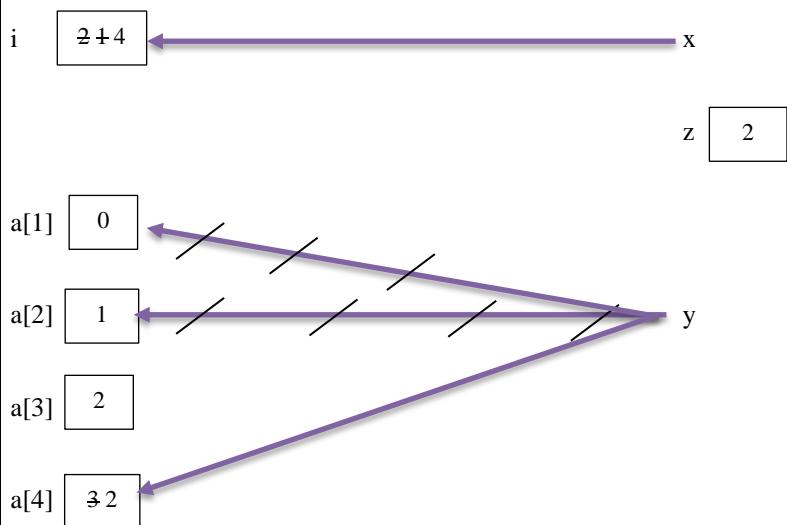
Η εντολή write στο main τυπώνει $i=4, A[1]=0, A[2]=2, A[3]=2, A[4]=3$

iii. $A[i] \leftarrow y$

call by name

main

P



Η εντολή write στο main τυπώνει $i=4$, $A[1]=0$, $A[2]=1$, $A[3]=2$, $A[4]=2$

Παρατηρήσεις για το call by name

- Καλώντας την P με ορίσματα i , $A[i]$ που είναι μεταβλητές το call by name ταυτίζεται με το call by reference
- Αρχικά ο δείκτης y δείχνει στο $A[2]$ διότι το i είναι 2. Ο δείκτης y δείχνει στο $A[i]$ γενικά
- Όταν το i αλλάζει τιμή και γίνει 4, ο δείκτης y θα δείξει αυτόματα στο $A[4]$
- Όταν καταχωρούμε δείκτη σε δείκτη βάζουμε τον αριστερό δείκτη να δείχνει όπου και ο δεξιός δείκτης π.χ. στην καταχώριση $x:=y$; βάζουμε το δείκτη x να δείχνει όπου και ο y

1.35 Θέμα 2 Σεπτέμβριος 2020

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal:

```
program MAIN;
var A:array [0..9] of integer;
var j, k:integer;
procedure f(x,y, Lo, Hi: integer);
var k: integer;
begin
  for k:=Lo to Hi do
    y:=k+1;
    x:=x+1;
end;
BEGIN
  for k:=0 to 9 do A[k]:=0;
  j:=3;
  f(j, A[j], 0, 5);
  for k:=0 to 9 do
    writeln(A[k]);
END.
```

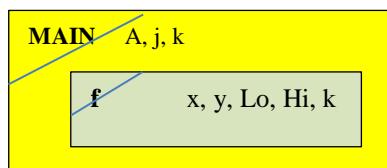
- a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τιμημάτων του προγράμματος;
- b) Τι τιμές του A τυπώνονται στο τέλος με την εντολή writeln, εφόσον η διαδικασία f καλείται στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού);

- Κλήση με τιμή (call by value)
- Κλήση με αναφορά (call by reference)
- Κλήση με όνομα (call by name)

Εξηγήστε σύντομα τη διαδικασία υπολογισμού στις τρεις περιπτώσεις

Απάντηση

a)



Περιβάλλοντα Αναφοράς

MAIN: Τοπικό: A, j, k, f (κλήση)

Μη Τοπικό και Καθολικό: –

f: Τοπικό: x, y, Lo, Hi, k

Μη Τοπικό και Καθολικό: A, j, f (κλήση)

b)

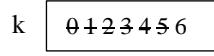
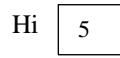
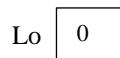
Call by value

MAIN

f

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

(Αρχικοπότηση Πίνακα)

*for k:=0 to 5 do*1^η επανάληψη για k=0: y=0+1=1, x=3+1=42^η επανάληψη για k=1: y=1+1=2, x=4+1=53^η επανάληψη για k=2: y=2+1=3, x=5+1=64^η επανάληψη για k=3: y=3+1=4, x=6+1=75^η επανάληψη για k=4: y=4+1=5, x=7+1=86^η επανάληψη για k=5: y=5+1=6, x=8+1=9Στο main με το *for k:=0 to 9 do* τυπώνεται ο πίνακας A με τις ακόλουθες τιμές:

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Παρατήρηση για το Call By Value

Μεταβιβάζονται οι τιμές των j και A[j] στις παραμέτρους x και y αντίστοιχα λόγω του call by value. Οι σταθερές 0 και 5 μεταβιβάζονται by value στις παραμέτρους Lo και Hi αντίστοιχα. **Οι παράμετροι της procedure δεν επιστρέφονται πίσω από την procedure όταν αυτή τελειώσει διότι έχουμε call by value.** Στο MAIN τυπώνεται ο πίνακας A με τις αρχικές του τιμές.

Call by reference

MAIN

f

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

(Αρχικοποίηση Πίνακα)



Lo 0

Hi 5

k 0 1 2 3 4 5 6

for k:=0 to 5 do

1^η επανάληψη για k=0: y=0+1=**1**, x=3+1=**4**

2^η επανάληψη για k=1: y=1+1=**2**, x=4+1=**5**

3^η επανάληψη για k=2: y=2+1=**3**, x=5+1=**6**

4^η επανάληψη για k=3: y=3+1=**4**, x=6+1=**7**

5^η επανάληψη για k=4: y=4+1=**5**, x=7+1=**8**

6^η επανάληψη για k=5: y=5+1=**6**, x=8+1=**9**

Στο main με το *for k:=0 to 9 do* τυπώνεται ο πίνακας Α με τις ακόλουθες τιμές:

0	1	2	3	4	5	6	7	8	9
0	0	0	6	0	0	0	0	0	0

Παρατήρηση για το Call By Reference

Οι διευθύνσεις των μεταβλητών **j** και **A[j]** μεταβιβάζονται στους δείκτες **x** και **y** αντίστοιχα ενώ οι σταθερές **0** και **5** μεταβιβάζονται **by value** στο **Lo** και **Hi** αντίστοιχα. Μέσω των δεικτών τροποποιούμε απευθείας τη μεταβλητή **j** και στο στοιχείο **A[3]**. Στο MAIN τυπώνεται ο πίνακας Α με αλλαγμένο μόνο το στοιχείο **A[3]**.

Call by name

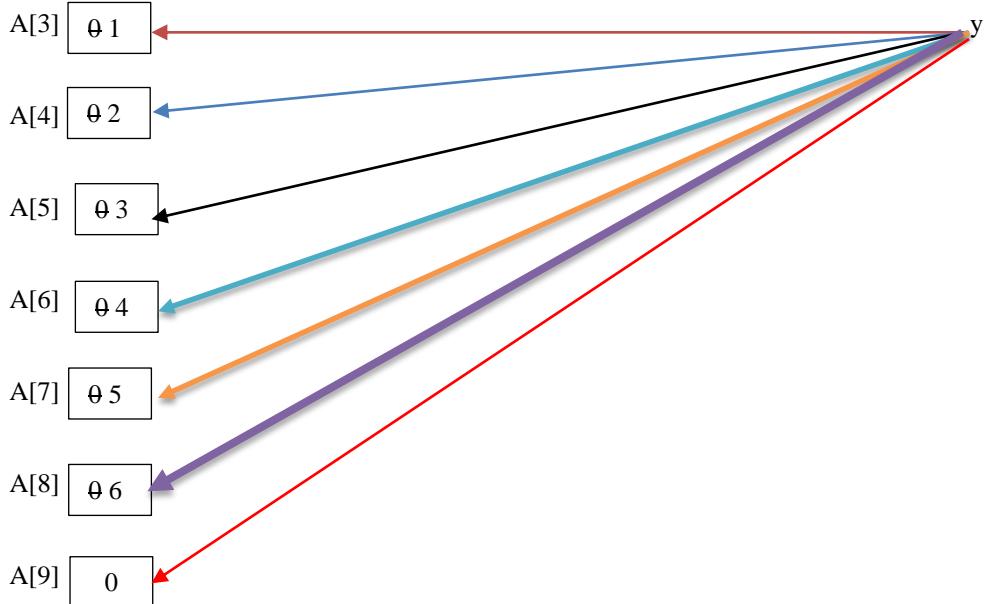
MAIN

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

(Αρχικοπόίηση Πίνακα)



$A[j] \leftarrow y$



Lo 0

Hi 5

k θ 4 2 3 4 5 6

for k:=0 to 5 do

1^η επανάληψη για k=0: y=0+1=1, x=3+1=4

2^η επανάληψη για k=1: y=1+1=2, x=4+1=5

3^η επανάληψη για k=2: y=2+1=3, x=5+1=6

4^η επανάληψη για k=3: y=3+1=4, x=6+1=7

5^η επανάληψη για k=4: y=4+1=5, x=7+1=8

6^η επανάληψη για k=5: y=5+1=6, x=8+1=9

Στο main με το **for k:=0 to 9 do** τυπώνεται ο πίνακας A με τις ακόλουθες τιμές:

0	1	2	3	4	5	6	7	8	9
0	0	0	1	2	3	4	5	6	0

Παρατήρηση για το Call By Name

Οι διευθύνσεις των μεταβλητών j και $A[j]$ μεταβιβάζονται by reference στους δείκτες x και y αντίστοιχα ενώ οι σταθερές 0 και 5 μεταβιβάζονται by value στις μεταβλητές Lo και Hi αντίστοιχα. Θυμίζουμε ότι στο call by name οι μεταβλητές μεταβιβάζονται με call by reference (το j και το $A[j]$ είναι μεταβλητές) ενώ οι σταθερές μεταβιβάζονται με call by value (το 0 και το 5 είναι σταθερές). Επίσης στο call by name επειδή έχουμε μεταβιβάζουμε τη διεύθυνση του στοιχείου πίνακα $A[j]$ στο δείκτη y δηλ. $A[j] \leftarrow y$ κάθε φορά που αλλάζει το j μέσα στις $[]$ ο δείκτης y αυτομάτως δείγνει στο νέο στοιχείο $A[j]$ του πίνακα A με βάση τη νέα τιμή του j . Αυτή η ιδιομορφία στους πίνακες ισχύει μόνο στο call by name.

1.36 Ασκηση 2 από Φροντιστήριο Ιούνιος 2022

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας:

program MAIN;

var z: integer;

a: array [1..2] of integer;

procedure P(x: integer);

begin

 a[1]:= 6;

 z:= 2;

 x:= x + 5;

end;

BEGIN

 a[1]:= 2; a[2]:=3;

 z:=1; x:= 2;

 P(a[z]);

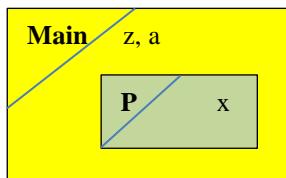
write(a[1], a[2], z)

END.

- a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- b) Στον παραπάνω κώδικα υπάρχει/ουν εντολή/ές που δεν είναι σωστή/ές από σημασιολογική άποψη και θα προκαλέσει/ουν **run-time error**. Ποια/ες είναι αυτή/ές και γιατί; «Σβήστε» την/τις εντολή/ές από τον κώδικα.
- c) Τι τυπώνεται στο τέλος με την εντολή **write**, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγείστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού);
- Κλήση με τιμή (call by value)
 - Κλήση με τιμή – αποτέλεσμα (call by value – result)
 - Κλήση με αναφορά (call by reference)
 - Κλήση με όνομα (call by name)

Απάντηση

a)



Main: Τοπικό: z, a, P (κλήση) Μη-Τοπικό και Καθολικό: -

P: Τοπικό: x Μη-Τοπικό και Καθολικό: z, a, P (κλήση)

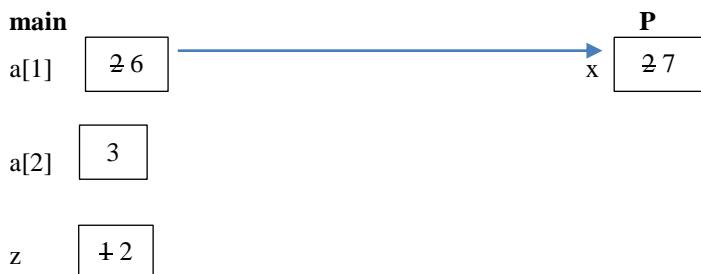
b)

Η μόνη εντολή που δεν είναι σημασιολογικά σωστή και θα προκαλέσει run-time error είναι η εντολή **x:= 2; στο main διότι δεν υπάρχει δήλωση μεταβλητής x στο main**.

- ✓ Προσοχή η παράμετρος x στην procedure P αποτελεί **τοπική μεταβλητή της procedure** και δεν είναι ορατή στο main
- ✓ Προσοχή η μεταβλητή z του main μπορεί να χρησιμοποιηθεί στην procedure P άρα η εντολή z:=2; της procedure P είναι σωστή διότι οι μεταβλητές του main είναι ορατές εντός των συναρτήσεων. Το αντίθετο ΔΕΝ ισχύει, δηλ. οι τοπικές μεταβλητές των συναρτήσεων ΔΕΝ είναι ορατές στο main

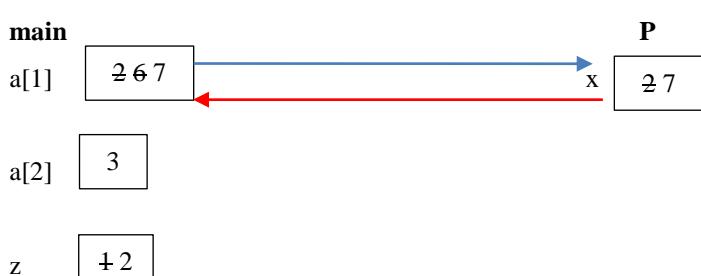
c)

call by value



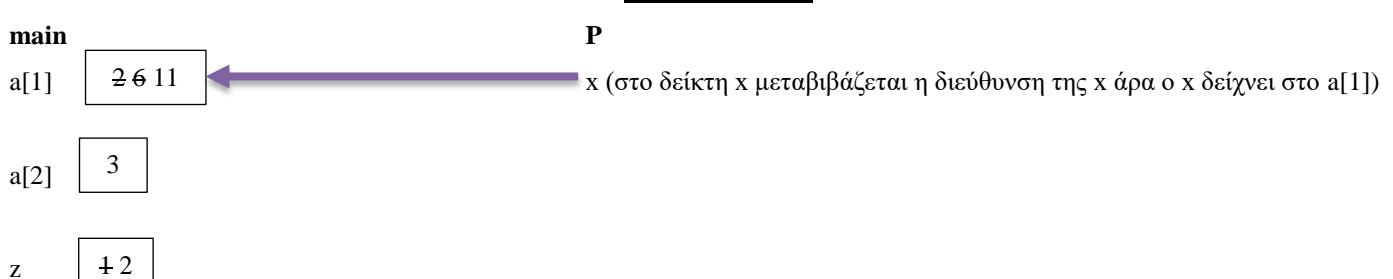
Η εντολή **write(a[1], a[2], z)** στο main τυπώνει τις τιμές **6, 3, 2**

call by value-result



Η εντολή **write(a[1], a[2], z)** στο main τυπώνει τις τιμές **7, 3, 2**

call by reference



Η εντολή **write(a[1], a[2], z)** στο main τυπώνει τις τιμές **11, 3, 2**

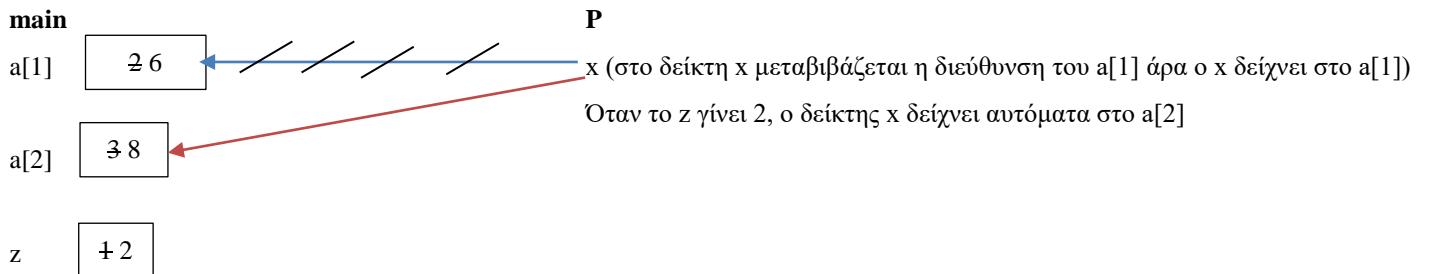
call by name

Παρατήρηση 1: Επειδή η παράμετρος που μεταβιβάζουμε στη συνάρτηση είναι απλή μεταβλητή το call by name ταυτίζεται με το call by reference.

Παρατήρηση 2: Επειδή μεταβιβάζουμε το $a[z]$ με $z=1$ ο δείκτης x δείχνει στο $A[1]$ αρχικά

Παρατήρηση 3: Με την εντολή $z:=2$; ο δείκτης x δείχνει πλέον στο $A[2]$. ΠΡΟΣΟΧΗ ΑΥΤΗ Η ΑΛΛΑΓΗ ΣΤΟ ΔΕΙΚΤΗ ΓΙΝΕΤΑΙ ΑΠΟΚΛΕΙΣΤΙΚΑ ΜΟΝΟ ΣΤΟ CALL BY NAME

Παρατήρηση 4: Η εντολή $x := x + 5$ εκτελείται στο $A[2]$ και όχι στο $A[1]$



1.37 Θέμα 1 Φεβρουάριος 2023

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί στατικό κανόνα εμβέλειας

program MAIN;

var a, b, c: **integer**;

function F1(x, y):**integer**;

var k, l: **integer**;

function F2(w: **integer**):**integer**;

begin

if (w>5) **then**

 w:=w-1;

else

 w:=w+1;

 k:=w*w;

 F2:=k;

end;

begin

 x:=x+z;

 k:=F2(x);

 l:=F2(x);

 b:=F2(y);

 F1:=k+l+b;

end;

function F3(x, y):**integer**;

var z: **integer**;

begin

 z:=1;

 F3:=F1(x, y);

end;

BEGIN

 a:=6;

 b:=4;

 k:=5;

 c:=F3(a, b);

writeln(a); **writeln**(b); **writeln**(c);

END.

- (a) Ποια είναι τα **περιβάλλοντα αναφοράς** (τοπικά, μη-τοπικά, καθολικά)όλων των τμημάτων του προγράμματος;
- (b) Στον κώδικα υπάρχουν δύο εντολές οι οποίες δεν είναι σωστές σημασιολογικά και θα προκαλέσουν run-time error. Ποιες είναι αυτές και γιατί; Σβήστε τις δύο εντολές από τον κώδικα
- (c) Παρουσιάστε τη **στοίβα εκτέλεσης** (run-time stack) των εγγραφών ενεργοποίησης (activation Records-AR) μόλις έχει γίνει η **πρώτη κλήση της F2**;
- (d) Ποιες τιμές **θα τυπωθούν στο τέλος για τα a, b, c** στις παρακάτω δύο περιπτώσεις μεταβίβασης παραμέτρων; Παρουσιάστε τις μεταβολές των μεταβλητών και παραμέτρων

1. Κλήση **με τιμή** (call by value)
2. Κλήση **με αναφορά** (call by reference)

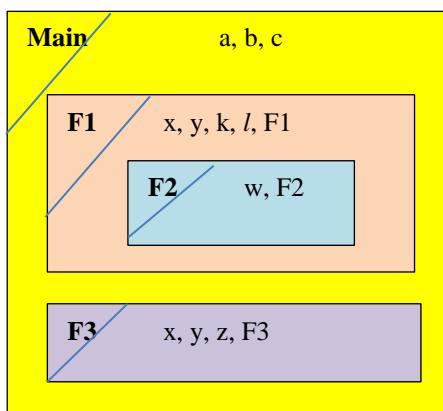
- (e) Απαντήστε στα (b) και (d) θεωρώντας ότι **ισχύει ο δυναμικός κανόνας εμβέλειας**. Τώρα μόνο η **μία από τις δύο εντολές δεν είναι σωστή** από σημασιολογική άποψη και σβήνετε μόνο αυτή. Ποια είναι η εντολή που είναι τώρα σωστή και γιατί;

Απάντηση

(a)

Στατικός Κανόνας Εμβέλειας

Τα blocks των υποπρογραμμάτων τοποθετούνται με τη σειρά γραφής τους στον κώδικα.



Περιβάλλοντα Αναφοράς

Main: Τοπικό: **a,b, c, F1 (κλήση), F3 (κλήση)**

Μη-Τοπικό και Καθολικό: –

F1: Τοπικό: **x, y, F1 (τιμή), k, l, F2 (κλήση)**

Μη-Τοπικό και Καθολικό: **a, b, c, F1 (κλήση), F3 (κλήση)**

F2: Τοπικό: **w, F2 (τιμή)**

Μη-Τοπικό: **x, y, F1 (τιμή), k, l, F2 (κλήση) (από F1), a, b, c, F1 (κλήση), F3 (κλήση) (από main)**

Καθολικό: **F1 (κλήση), F3 (κλήση)**

F3: Τοπικό: **x, y, z, F3 (τιμή)**

Μη-Τοπικό και Καθολικό: **a,b, c, F1 (κλήση), F3 (κλήση)**

(b) Λανθασμένες Εντολές

- Η 1^η εντολή που προκαλεί run-time error είναι η **k:=5**; στο main διότι δεν έχει δηλωθεί μεταβλητή k στο main
- Η 2^η εντολή που προκαλεί run-time error είναι η **x:=x+z**; στη function F1 διότι δεν έχει δηλωθεί μεταβλητή z ούτε στη function F1 αλλά ούτε και στο περιβάλλον μπλοκ της F1 δηλ. στο main

c) Στοίβα Εκτέλεσης στην 1^η κλήση της F2

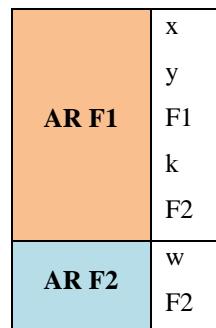
AR main	a b c F1 F3
AR F3	x y F3 z
AR F1	x y F1 k F2
AR F2	w F2

Αν ζητούσε τη Στοίβα Εκτέλεσης στη 1^η κλήση της F2 τότε όταν τερματίσει η F2 αφαιρείται από τη στοίβα

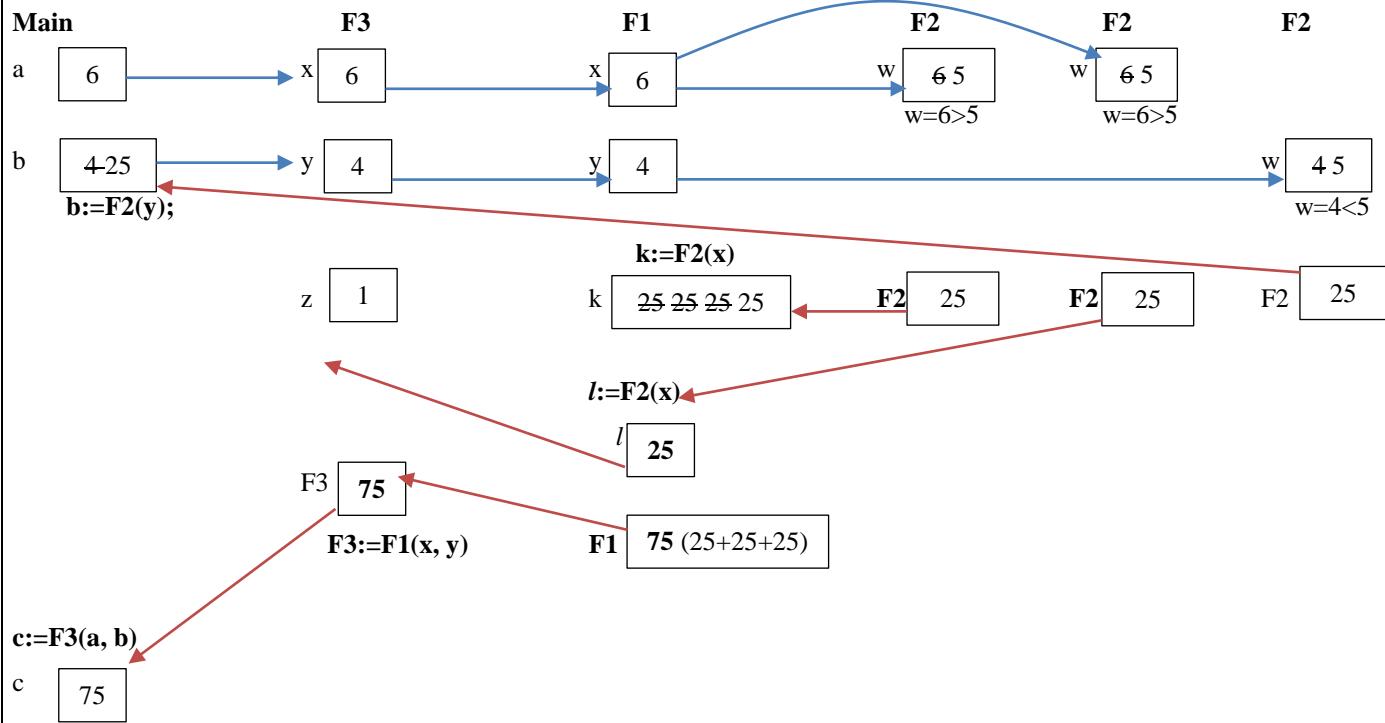
AR main	a b c F1 F3
AR F3	x y F3 z
AR F1	x y F1 k F2

και μετά όταν ξανακληθεί η F2 το μπλοκ της προστίθεται στη στοίβα

AR main	a b c F1 F3
AR F3	x y F3 z

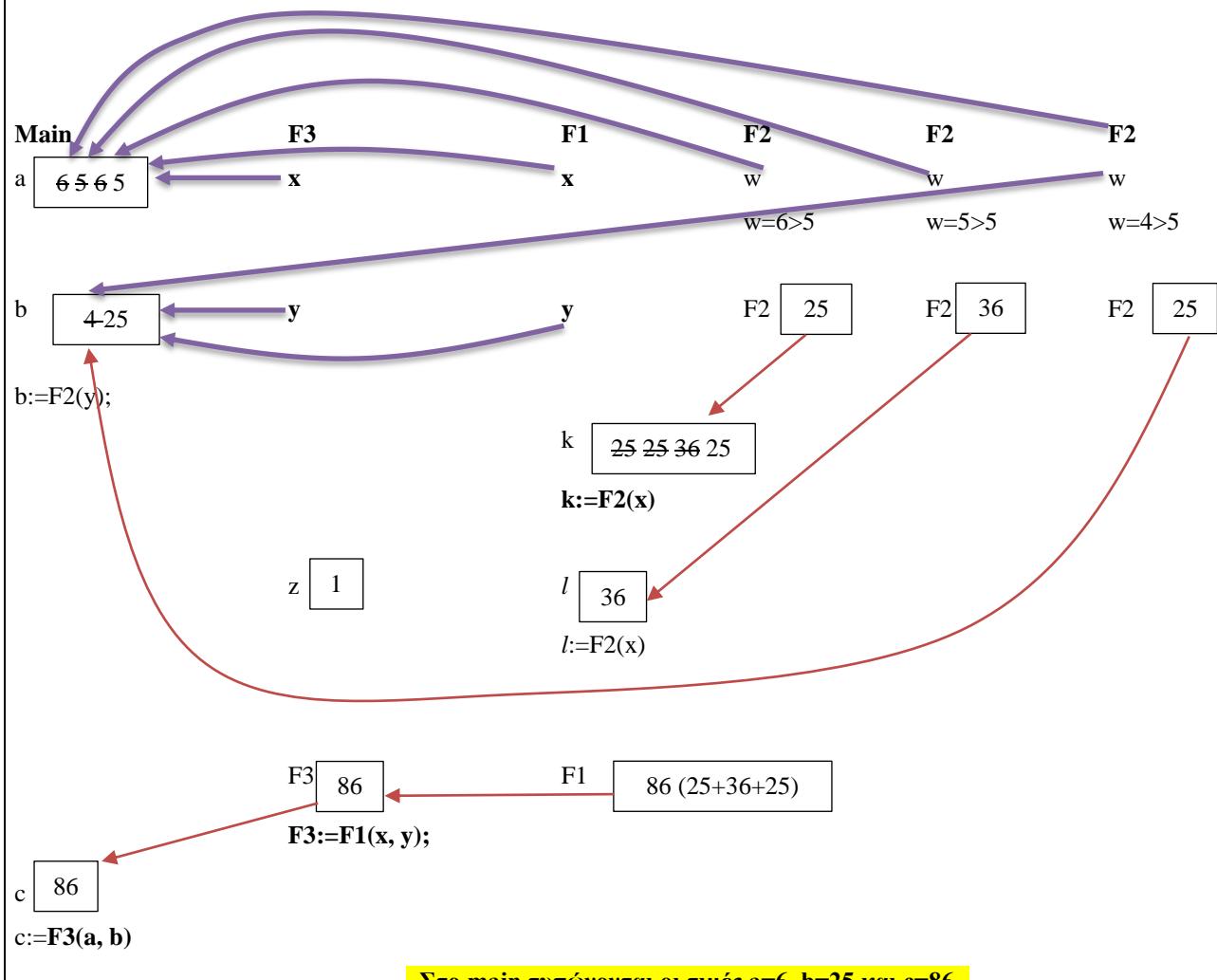


(d1) Call by value



Στο main τυπώνονται οι τιμές $a=6$, $b=25$ και $c=75$

(d2) Call by reference

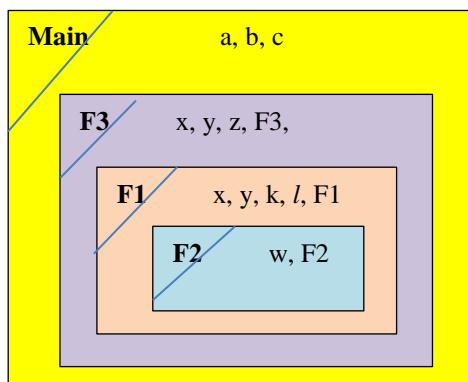


(e)

(e). Ερώτημα (a)

Δυναμικός Κανόνας Εμβέλειας

Τα blocks των υποπρογραμμάτων τοποθετούνται με τη σειρά εκτέλεσης στον κώδικα.



Περιβάλλοντα Αναφοράς

Main: Τοπικό: **a,b, c, F1 (κλήση), F3 (κλήση)**

Μη-Τοπικό και Καθολικό: –

F1: Τοπικό: **x, y, F1 (τιμή), k, l, F2 (κλήση)**

Μη-Τοπικό: **z, F3 (τιμή) (από F3), a, b, c, F1 (κλήση), F3 (κλήση) (από main)**

Καθολικό: **F1 (κλήση), F3 (κλήση)**

F2: Τοπικό: **w, F2 (τιμή)**

Μη-Τοπικό: **x, y, F1 (τιμή), k, l, F2 (κλήση) (από F1) z, F3 (τιμή) (από F3)**

Καθολικό: **a,b, c, F1 (κλήση), F3 (κλήση) (από main)**

Καθολικό: **F1 (κλήση), F3 (κλήση)**

F3: Τοπικό: **x, y, z, F3 (τιμή)**

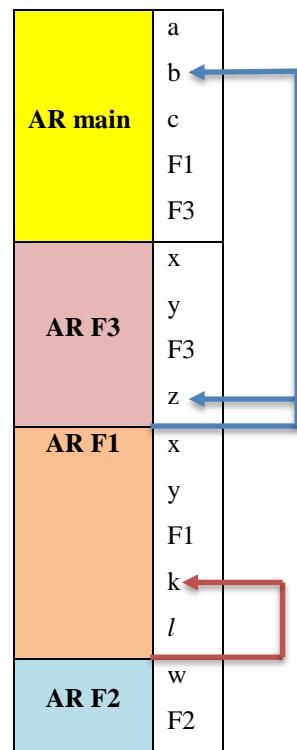
Μη-Τοπικό και Καθολικό: **a,b, c, F1 (κλήση), F3 (κλήση)**

(e) Ερώτημα (b)

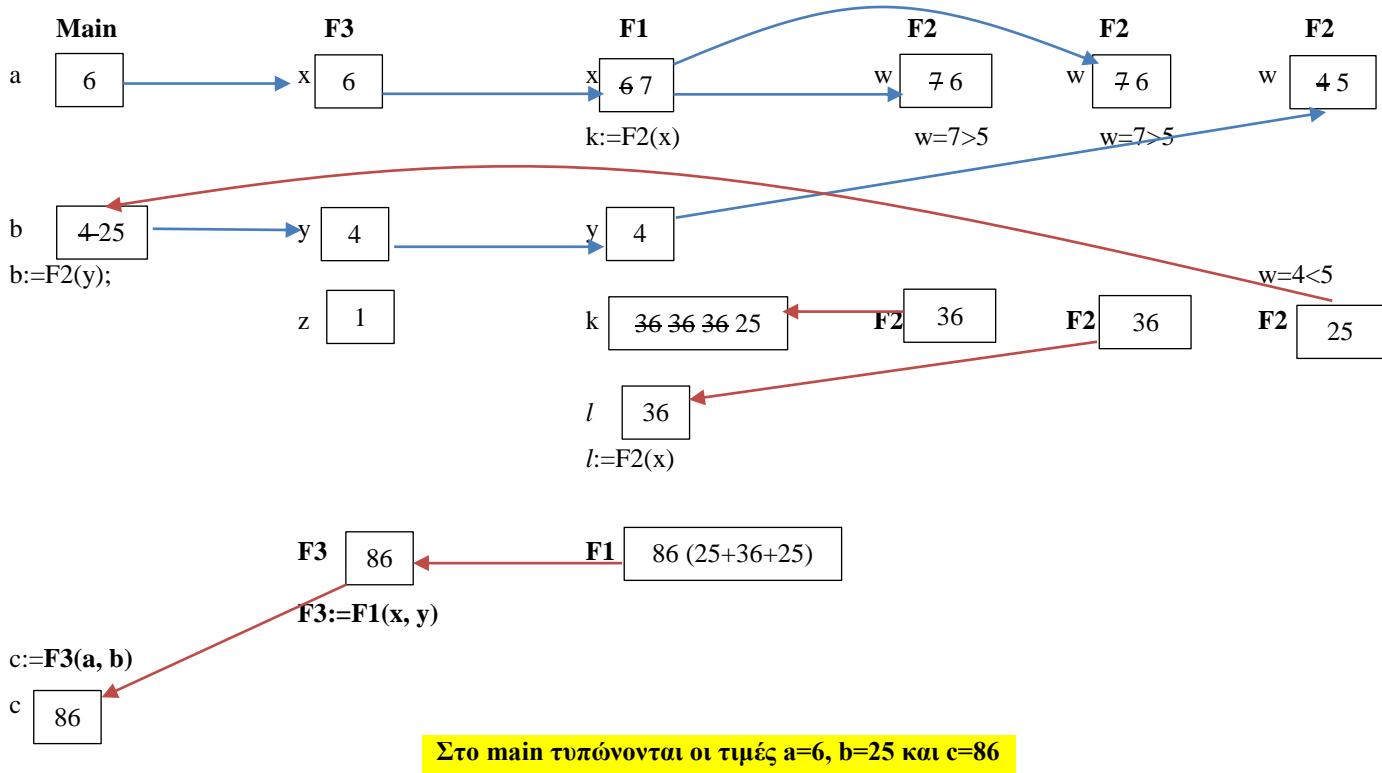
Η λανθασμένη εντολή είναι η $k:=5$; διότι δεν υπάρχει δήλωση μεταβλητής k στο main.

Προσοχή: Η εντολή $x:=x+z$ στη συνάρτηση $F1$ είναι **τώρα σωστή** διότι υπάρχει μεταβλητή z στη συνάρτηση $F3$ που είναι το περιβάλλον μπλοκ της $F1$

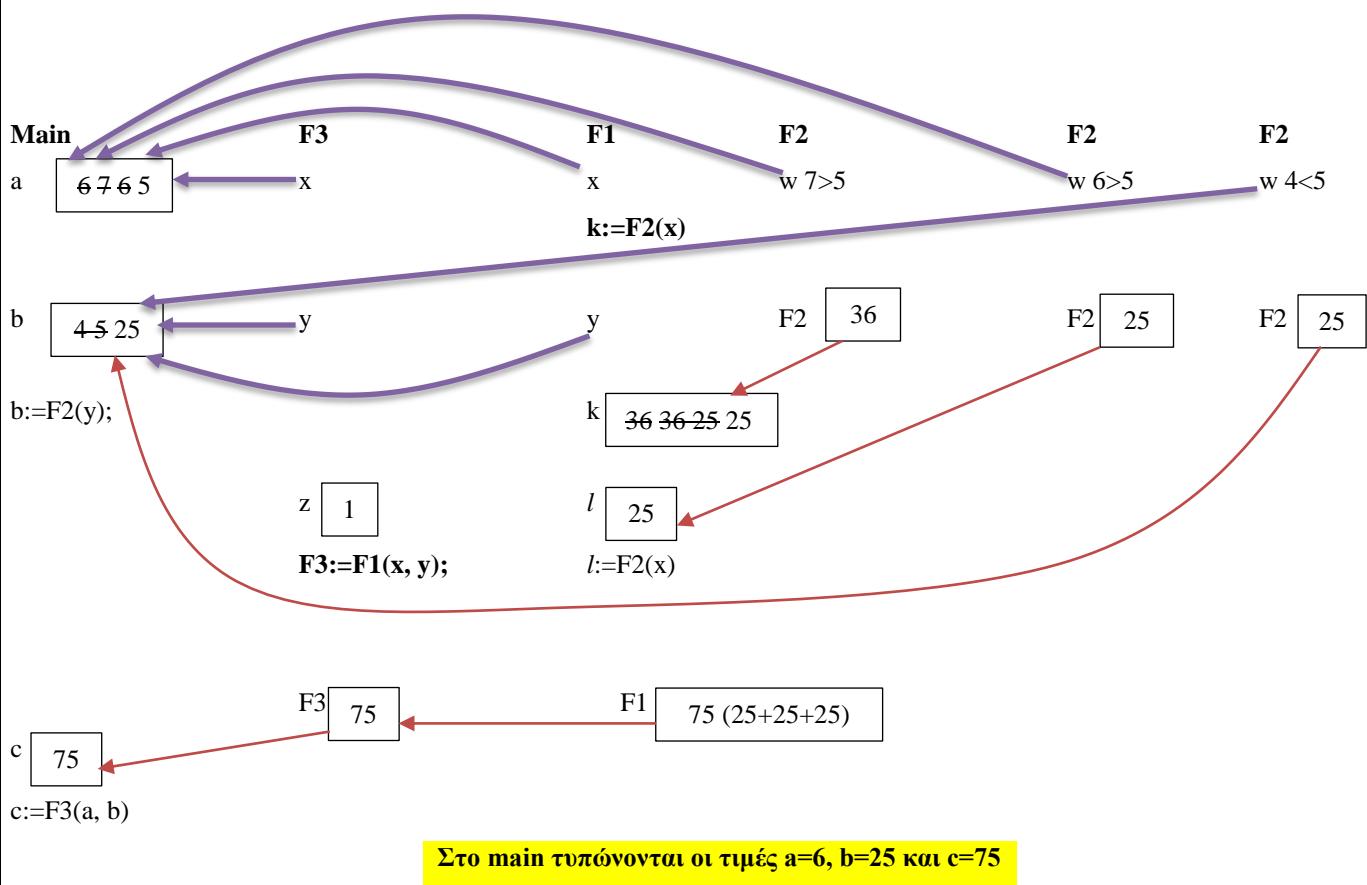
(e) Ερώτημα (c)



(e) Ερώτημα (d1) Call by value



(e) Ερώτημα (d2) Call by reference



1.38 Θέμα 3 Ιούνιος 2023

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal που χρησιμοποιεί το στατικό κανόνα εμβέλειας:

program MAIN;

var n: **integer**;

var A: **array[1..5] of integer**;

procedure F(k, b: **integer**);

begin

writeln(k);

 n:=n+1;

writeln(k);

 k:=k+4;

writeln(n);

 b:=b+5;

end;

BEGIN

 A[1]:=10; A[2]:=20; A[3]:=30; A[4]:=40; A[5]:=50;

 n:=1;

 F(n, A[n+2]);

writeln(n);

writeln(A[1], ' ', A[2], ' ', A[3], ' ', A[4], ' ', A[5]);

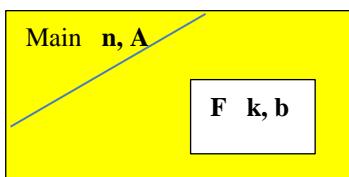
END.

(a) Θεωρώντας ότι ισχύει ο **στατικός κανόνας εμβέλειας** ποια τα περιβάλλοντα του προγράμματος (τοπικά, μη τοπικά και καθολικά) όλων των τμημάτων του προγράμματος

b) **Ποιες τιμές θα τυπωθούν με τις εντολές writeln** εφόσον η διαδικασία F καλείται με τις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων; Εξηγήστε σύντομα τις διαδικασίες υπολογισμού σε όλες τις περιπτώσεις

1. Κλήση με τιμή (call by value)
2. Κλήση με αναφορά (call by reference)
3. Κλήση με τιμή-αποτέλεσμα (call by value-result)
4. Κλήση με όνομα (call by name)

Απάντηση



Περιβάλλοντα Αναφοράς

Main: **τοπικό περιβάλλον:** n, A, F

ΜΗτοπικό και καθολικό περιβάλλον: –

Procedure F: **τοπικό περιβάλλον:** k, b

ΜΗτοπικό και καθολικό περιβάλλον: n, A, F

Call by value

main

1	2	3	4	5
10	20	30	40	50



στην procedure F

τυπώνεται 1

τυπώνεται 1

τυπώνεται 2

στο main τυπώνεται η τιμή 2 και ο πίνακας με τις τιμές 10 20 30 40 50

Call by value-result

main

1	2	3	4	5
10	20	30 35	40	50



στην procedure F

τυπώνεται 1

τυπώνεται 1

τυπώνεται 2

στο main τυπώνεται η τιμή 5 και ο πίνακας με τις τιμές 10 20 35 40 50

Call by reference

main

1	2	3	4	5
10	20	30 35	40	50



στην procedure F

τυπώνεται 1

τυπώνεται 2

τυπώνεται 6

στο main τυπώνεται η τιμή 6 και ο πίνακας με τις τιμές 10 20 35 40 50

Call by name

main

1	2	3	4	5
10	20	30	40	50



στην procedure F

τυπώνεται 1

τυπώνεται 2

τυπώνεται 6

στο main τυπώνεται η τιμή 6 και ο πίνακας με τις τιμές 10 20 30 40 50

Η καταχώριση $b:=b+5$ ΔΕΝ έχει καμία επίδραση διότι ο δείκτης b δείχνει σε απροσδιόριστη θέση μνήμης (δείχνει σε θέση εκτός του πίνακα)

2 ΘΕΜΑΤΑ ΚΑΙ ΑΣΚΗΣΕΙΣ ΜΕ NFA, DFA ΚΑΙ ΚΑΝΟΝΙΚΕΣ ΕΚΦΡΑΣΕΙΣ

2.1 Βασική Θεωρία

Θεωρία για DFA

Τα Deterministic Finite Automata (**DFA-Ντετερμινιστικά Πεπερασμένα Αυτόματα**) είναι μηχανές αναγνώρισης κανονικών γλωσσών και αναπαριστάνονται με διαγράμματα καταστάσεων. Έχουν τα ακόλουθα χαρακτηριστικά:

- Από κάθε κατάσταση ενός DFA πρέπει να υπάρχει μοναδική μετάβαση για κάθε χαρακτήρα του αλφαβήτου
- Τα DFA έχουν καταστάσεις εγκλωβισμού όταν παραβιάζονται οριστικά οι περιορισμοί της γλώσσας
- Τα DFA ΔΕΝ έχουν κενές μεταβάσεις δηλαδή η μετάβαση από μια κατάσταση σε άλλη γίνεται πάντα με χαρακτήρα

Θεωρία για NFA

Τα Non Deterministic Finite Automata (**NFA-ΜΗ Ντετερμινιστικά Πεπερασμένα Αυτόματα**) είναι μηχανές αναγνώρισης κανονικών γλωσσών και αναπαριστάνονται με διαγράμματα καταστάσεων. Έχουν τα ακόλουθα χαρακτηριστικά:

- ΔΕΝ απαιτούν μετάβαση από κάθε κατάσταση για κάθε χαρακτήρα του αλφαβήτου
- ΔΕΝ έχουν καταστάσεις εγκλωβισμού
- Μπορεί να έχουν μεταβάσεις με τον κενό χαρακτήρα ϵ
- Από μια κατάσταση μπορεί να υπάρχουν περισσότερες από μια μεταβάσεις με τον ίδιο χαρακτήρα. Αυτό δεν επιτρέπεται στα DFA όπου οι μεταβάσεις όπως αναφέραμε είναι μοναδικές

Θεωρία για Κανονικές Εκφράσεις

Οι Κανονικές Εκφράσεις (**regular expressions**) όπως και τα NFA και τα DFA είναι μηχανισμός περιγραφής κανονικών γλωσσών. Για την κατασκευή μιας κανονικής έκφρασης χρησιμοποιούνται τα ακόλουθα σύμβολα:

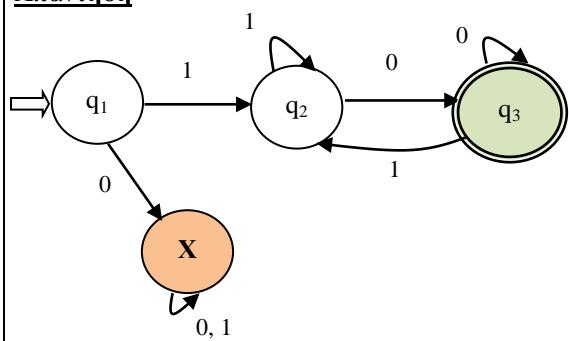
1. \cup ή | συμβολίζει **ένωση** δηλ. συμβολοσειρές που ανήκουν είτε στη μια γλώσσα είτε στην άλλη
2. • συμβολίζει **παράθεση** ή **συνένωση** δηλ μια συμβολοσειρά ακολουθείται από μια άλλη συμβολοσειρά
3. * (**Kleene star**) συμβολίζει 0 ή περισσότερες επαναλήψεις
4. + συμβολίζει 1 ή περισσότερες επαναλήψεις
5. ? συμβολίζει 0 ή 1 επανάληψη
6. ϵ συμβολίζει τον κενό χαρακτήρα

2.2 Ασκήσεις με DFA, NFA και Κανονικές Εκφράσεις

2.2.1 Παραδείγματα με DFA

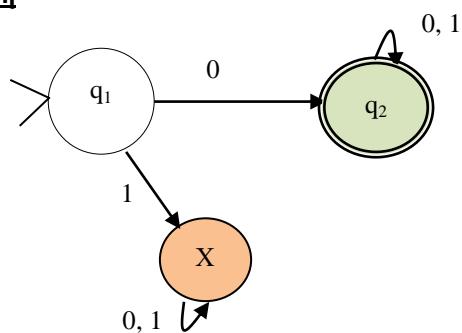
1) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ αρχίζει με } 1 \text{ και τελειώνει σε } 0\}$

Απάντηση



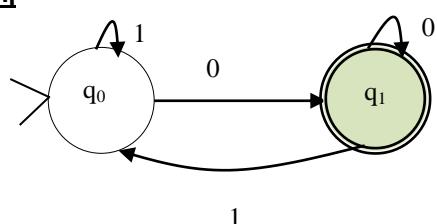
2) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ αρχίζει με } 0\}$

Απάντηση



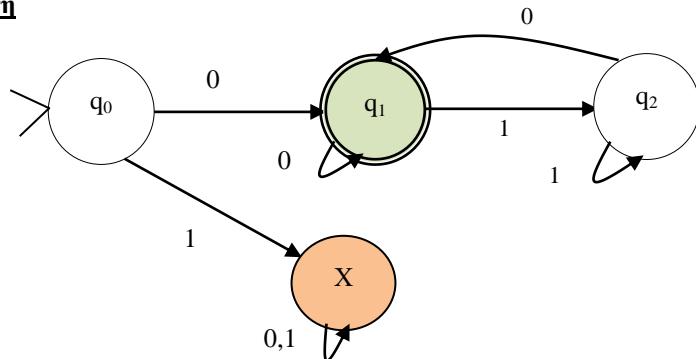
3) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ τελειώνει με } 0\}$

Απάντηση



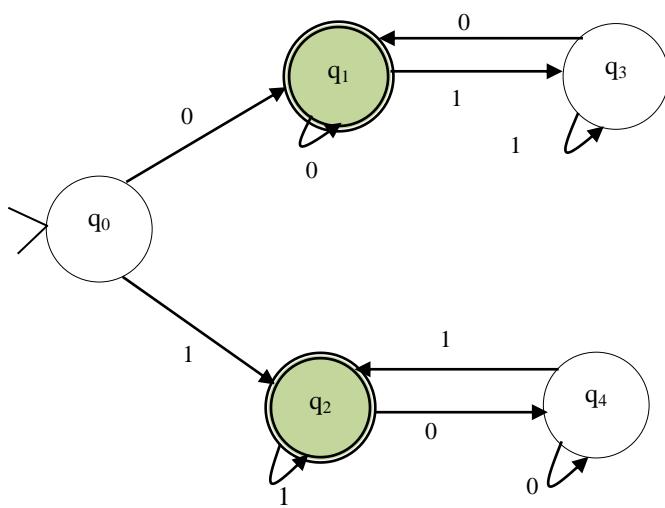
4) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ αρχίζει και τελειώνει με } 0\}$

Απάντηση



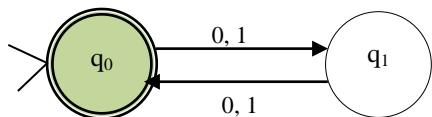
5) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ αρχίζει και τελειώνει με τον ίδιο χαρακτήρα}\}$

Απάντηση



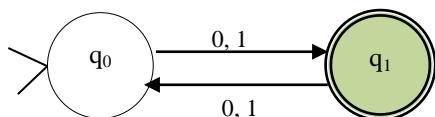
6) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ έχει άρτιο μήκος}\}$

Απάντηση



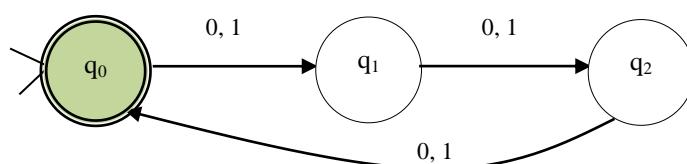
7) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ έχει περιττό μήκος}\}$

Απάντηση



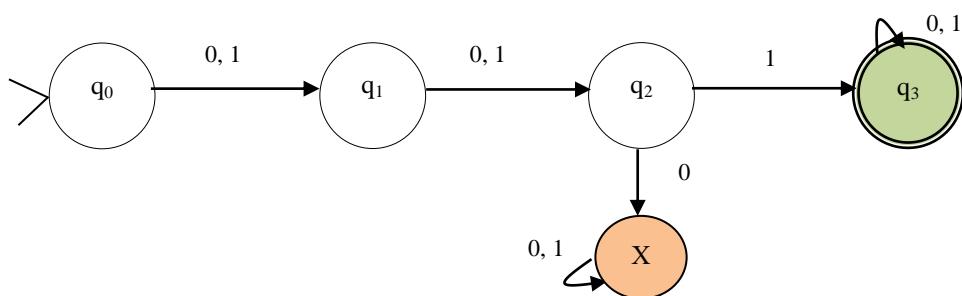
8) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \text{το μήκος της } w \text{ είναι πολλαπλάσιο του } 3\}$

Απάντηση



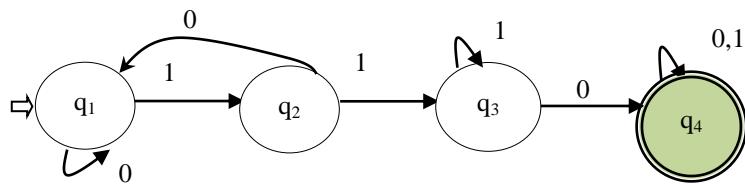
9) Να κατασκευάσετε DFA για τη γλώσσα $\{w \in \{0, 1\}^* \mid \text{το } |w| \geq 3 \text{ και ο } 3^{\text{ος}} \text{ χαρακτήρας να είναι το } 1\}$

Απάντηση



10) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ περιέχει την υπολέξη } 110\}$.

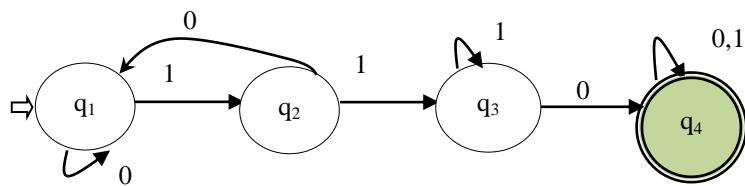
Απάντηση



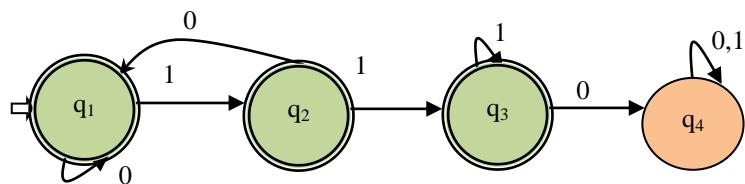
11) Να κατασκευάσετε DFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ ΔΕΝ περιέχει την υπολέξη } 110\}$.

Απάντηση

Κατασκευάζουμε πρώτα το συμπληρωματικό DFA δηλ. αυτό που περιέχει την υπολέξη **110** δηλ. το προηγούμενο



Το ζητούμενο DFA προκύπτει **με εναλλαγή καταστάσεων** δηλ. οι ΜΗ τελικές καταστάσεις του προηγούμενου θα γίνουν τελικές και οι τελικές θα γίνουν καταβόθρες δηλ:



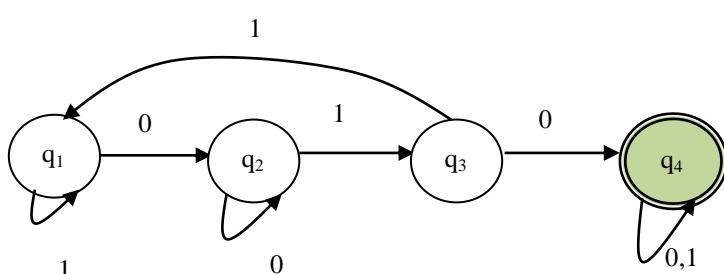
Παρατήρηση

Όταν ζητείται DFA που ΔΕΝ αποδέχεται κάποια συμβολοσειρά σχεδιάζουμε πρώτα το συμπληρωματικό του και μετά κάνουμε εναλλαγή καταστάσεων σε αυτό

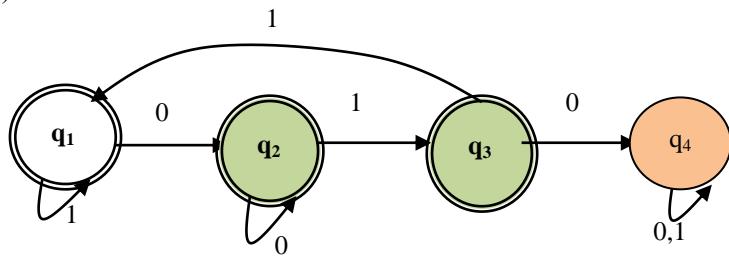
12) Να κατασκευάσετε το DFA α) για τη γλώσσα που αναγνωρίζει το **010** β) για τη γλώσσα που ΔΕΝ αναγνωρίζει το **010**

Απάντηση

α)

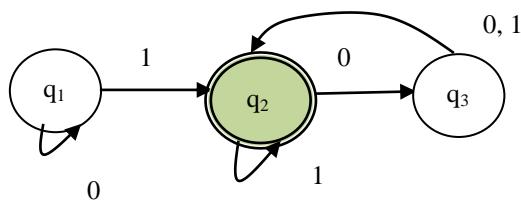


β)



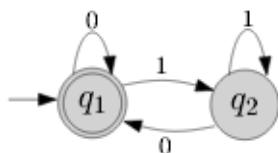
13) DFA για τη γλώσσα $L = \{w \in \{0,1\}^* \mid \text{η } w \text{ έχει τουλάχιστον ένα 1 και ο τελευταίος 1 ακολουθείται από άρτιο πλήθος 0}\}$

Απάντηση



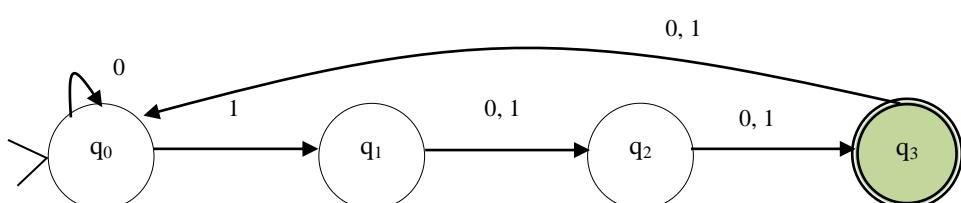
14) Να σχεδιαστεί DFA που να αναγνωρίζει τη γλώσσα $L = \{w \in \{0,1\}^* \mid \text{η } w \text{ λέξη έχει μήκος 0 ή τελειώνει σε 0}\}$

Απάντηση



15) Έστω $L = \{w \in \{0, 1\}^* \mid \text{o } 3^{\text{ος}} \text{ χαρακτήρας πριν το τέλος να είναι το 1}\}$

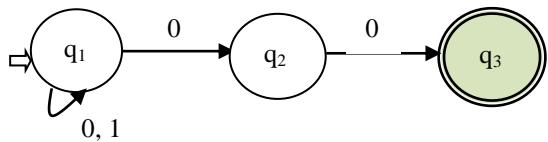
Απάντηση



2.2.2 Παραδείγματα με NFA

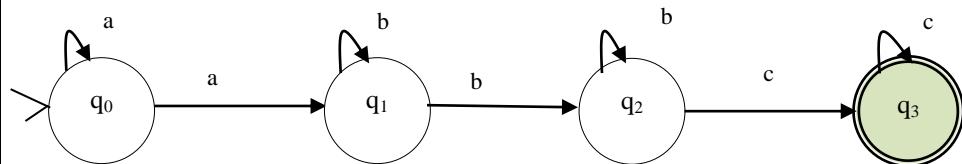
1) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid w \text{ τελειώνει σε } 00\}$.

Απάντηση



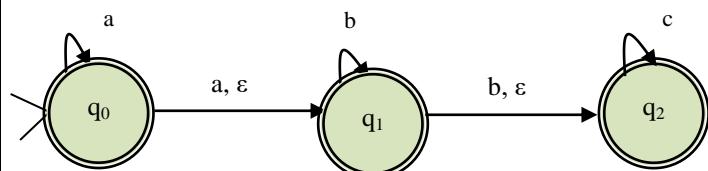
2) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{a, b, c\}^* \mid \text{τα } a \text{ προηγούνται των } b \text{ και τα } b \text{ προηγούνται των } c \text{ και πλήθος } a > 0, b > 0, c > 0\}$.

Απάντηση



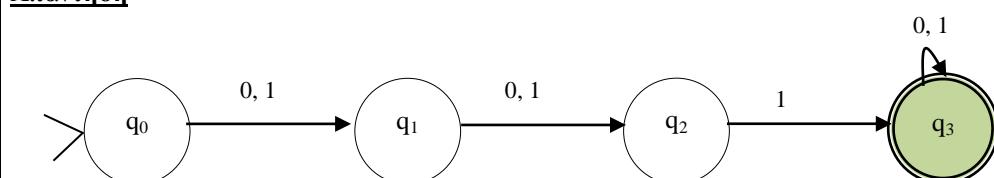
3) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{a, b, c\}^* \mid \text{τα } a \text{ προηγούνται των } b \text{ και τα } b \text{ προηγούνται των } c \text{ και πλήθος } a \geq 0, b \geq 0, c \geq 0\}$

Απάντηση

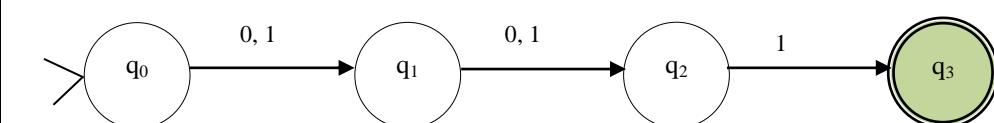


4) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \text{μήκος } |w| \geq 3 \text{ και ο } 3^{\text{ος}} \text{ χαρακτήρας να είναι το } 1\}$

Απάντηση

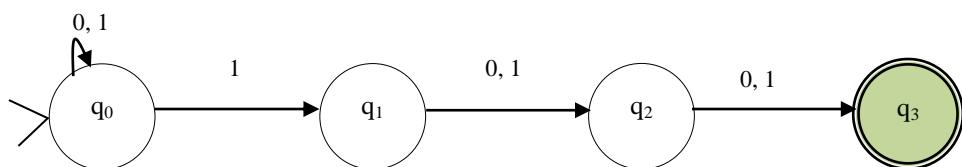


Εναλλακτικά:



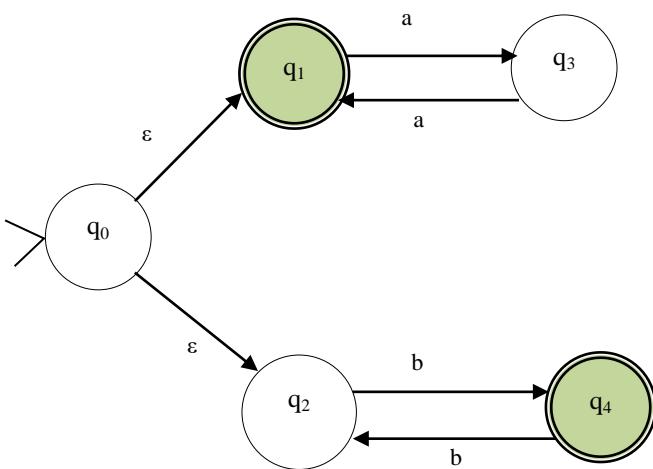
5) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid o 3^{0^c} \text{ χαρακτήρας πριν το τέλος να είναι το } 1\}$

Απάντηση



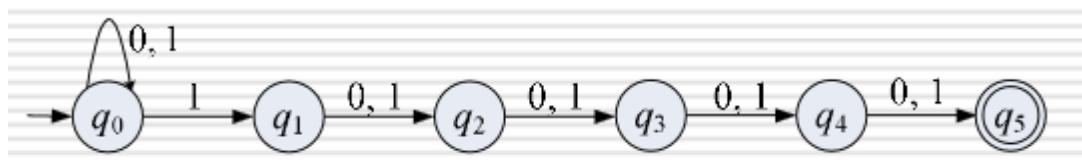
6) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{a, b\}^* \mid \text{το πλήθος των } a \text{ είναι άρτιο ή το πλήθος των } b \text{ είναι περιττό}\}$

Απάντηση



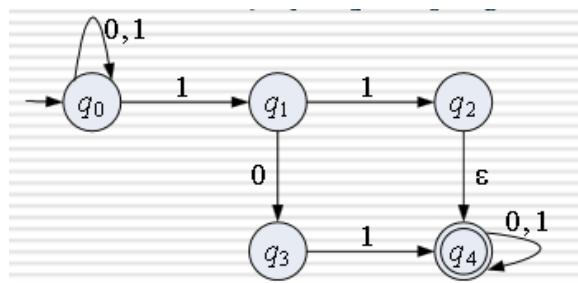
7) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ έχει } 1 \text{ στην } 5^{\text{η}} \text{ θέση από δεξιά}\}$

Απάντηση



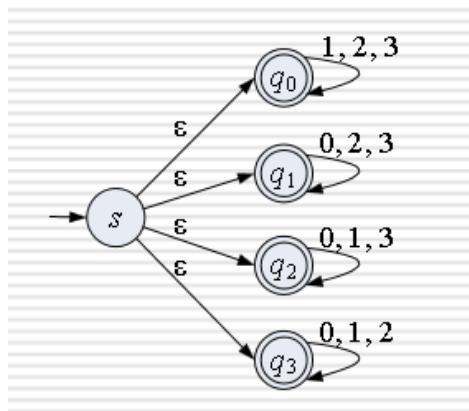
8) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* \mid \eta w \text{ περιέχει } 11 \text{ ή } 101\}$

Απάντηση



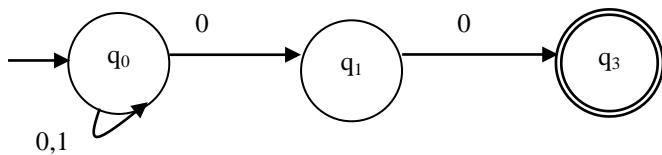
9) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1, 2, 3\}^* \mid \text{υπάρχει σύμβολο στο αλφάριθμο της γλώσσας που δεν εμφανίζεται στη w\}$

Απάντηση



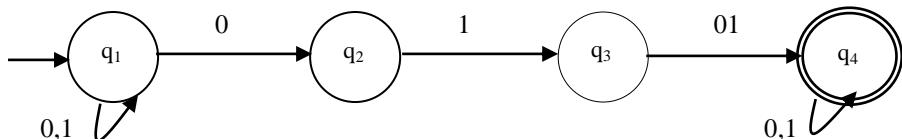
10) Να κατασκευάσετε NFA για την ακόλουθη γλώσσα $L = \{w \in \{0, 1\}^* \mid \text{η } w \text{ καταλήγει σε } 00\}$

Απάντηση



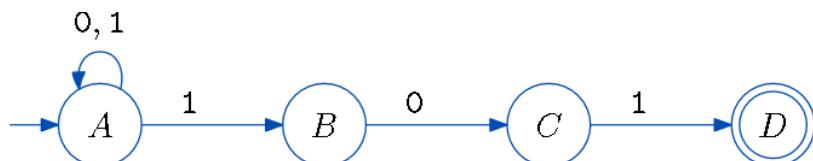
11) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0, 1\}^* : \text{η } w \text{ περιέχει } 0101\}$

Απάντηση



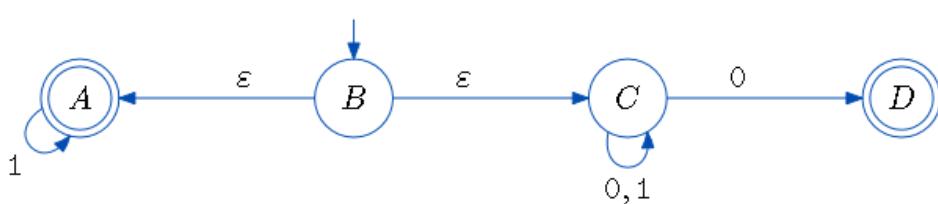
12) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0,1\}^* \mid \text{οι συμβολοσειρές τελειώνουν σε } 101\}$

Απάντηση



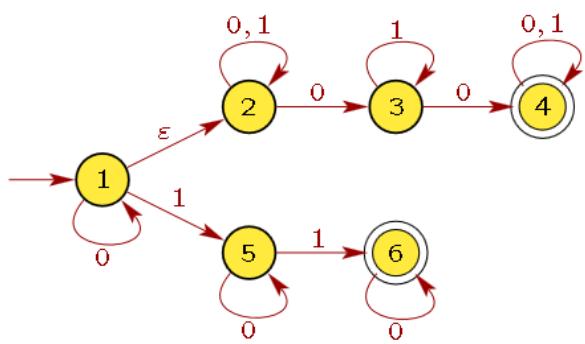
13) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0,1\}^* \mid \text{το τελευταίο σύμβολο κάθε συμβολοσειράς είναι το } 0 \text{ είτε οι συμβολοσειρές περιέχουν μόνο } 1\}$

Απάντηση



14) Να κατασκευάσετε NFA για τη γλώσσα $L = \{w \in \{0,1\}^* \mid \text{η συμβολοσειρά περιέχει τουλάχιστον δύο } 0 \text{ ή ακριβώς δύο } 1\}$.

Απάντηση



2.2.3 Παραδείγματα Κανονικών Εκφράσεων

1) Στα επόμενα παραδείγματα θεωρούμε ότι το αλφάβητο είναι το $\{0, 1\}$

- $(0 \cup 1)^*$ ή $(0 \mid 1)^*$ → οποιαδήποτε συμβολοσειρά από 0 και/ή 1 συμπεριλαμβανομένης της κενής
- $(0 \cup 1)^+$ → οποιαδήποτε συμβολοσειρά από 0 και/ή 1 χωρίς να συμπεριλαμβάνεται η κενή
- $(0 \cup 1)^*\cdot 0$ ή $(0 \mid 1)^*\cdot 0$ → συμβολοσειρές από 0 και/ή 1 που τελειώνουν σε 0
- $0\cdot(0 \cup 1)^*$ ή $0\cdot(0 \mid 1)^*$ → συμβολοσειρές από 0 και/ή 1 που αρχίζουν από 0
- $0\cdot(0 \cup 1)^*\cdot 0$ ή $0\cdot(0 \mid 1)^*\cdot 0$ → συμβολοσειρές από 0 και/ή 1 που αρχίζουν και τελειώνουν σε 0
- $\underline{0\cdot(0 \cup 1)^*\cdot 0} \cup \underline{1\cdot(0 \cup 1)^*\cdot 1}$ ή $0\cdot(0 \mid 1)^*\cdot 0 \mid 1\cdot(0 \mid 1)^*\cdot 1$ → συμβολοσειρές από 0 και/ή 1 που αρχίζουν και τελειώνουν με τον ίδιο χαρακτήρα
- $((0 \cup 1)(0 \cup 1))^*$ ή $((0 \mid 1)(0 \mid 1))^*$ → συμβολοσειρές από 0 και/ή 1 άρτιου μήκους
- $((0 \cup 1)(0 \cup 1))^*(0 \cup 1)$ ή $((0 \mid 1)(0 \mid 1))^*(0 \mid 1)$ → συμβολοσειρές από 0 και/ή 1 περιττού μήκους
- $((0 \cup 1)(0 \cup 1)(0 \cup 1))^*$ ή $((0 \mid 1)(0 \mid 1)(0 \mid 1))^*$ → συμβολοσειρές από 0 και/ή 1 μήκους πολλαπλάσιο του 3
- $(0 \cup 1)\cdot(0 \cup 1)\cdot(0 \cup 1)$ ή $(0 \mid 1)(0 \mid 1)(0 \mid 1)$ → συμβολοσειρές με μήκος ακριβώς 3 χαρακτήρες
- $(0 \cup 1 \cup \varepsilon)\cdot(0 \cup 1 \cup \varepsilon) \mathbf{1} (0 \cup 1 \cup \varepsilon)$ ή $(0 \mid 1 \mid \varepsilon)\cdot(0 \mid 1 \mid \varepsilon)\cdot(0 \mid 1 \mid \varepsilon)$ → συμβολοσειρές από 0 και/ή 1 με μήκος το πολύ 3 χαρακτήρες
- $(0 \cup 1)^*\cdot 00\cdot(0 \cup 1)^*$ → συμβολοσειρές από 0 και/ή 1 που περιέχουν την υποσυμβολοσειρά 00

2) Στα επόμενα παραδείγματα θεωρούμε ότι το αλφάβητο είναι το $\{a, b, c\}$

- $(a \cup b \cup c)^*$ ή $(a \mid b \mid c)^*$ → οποιαδήποτε συμβολοσειρά από a, b και/ή c συμπεριλαμβανομένης της κενής
- $(a \cup b \cup c)^+$ ή $(a \mid b \mid c)^+$ → οποιαδήποτε συμβολοσειρά από a, b και/ή c εκτός της κενής
- $(b \cup c)^*\cdot a \cdot(b \cup c)^*$ → συμβολοσειρές με μια μόνο εμφάνιση του χαρακτήρα a
- $a^*\cdot b^*\cdot c^*$ → συμβολοσειρές από a, b και/ή c συμπεριλαμβανομένης της κενής όπου τα a προηγούνται των b και τα b προηγούνται των c
- $a^+\cdot b^+\cdot c^+$ → ΜΗ ΚΕΝΕΣ συμβολοσειρές από a, b και/ή c όπου τα a προηγούνται των b και τα b προηγούνται των c
- $(a \cup \beta \cup \varepsilon)(a \cup \beta \cup \varepsilon)(a \cup \beta \cup \varepsilon)$ → συμβολοσειρές από a, b και/ή c με μήκος το πολύ 3 χαρακτήρες

3) Δώστε την κανονική έκφραση στο αλφάβητο $\Sigma = \{a, b\}$ όπου κάθε a ακολουθείται από 2b

Απάντηση

$b^*\cdot(\mathbf{ab})^*\cdot b^*$

4) Δώστε την κανονική έκφραση στο αλφάβητο $\Sigma = \{0, 1\}$ για συμβολοσειρές που καταλήγουν σε 1 και περιέχουν το 00

Απάντηση

$(0 \cup 1)^*\mathbf{00}(0 \cup 1)^*$ 1

5) Δώστε την κανονική έκφραση στο αλφάβητο $\Sigma = \{\alpha, \beta\}$ για συμβολοσειρές που δεν περιέχουν το $\alpha\alpha$

Απάντηση

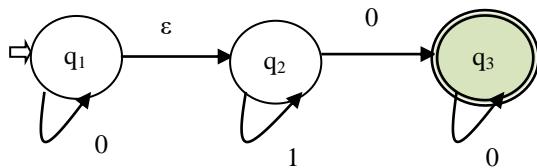
$(\alpha \cup \varepsilon) (\beta \cup \beta\alpha)^*$

2.2.4 Παραδείγματα με DFA, NFA και Κανονικές Εκφράσεις

1) Να κατασκευάσετε NFA για τη γλώσσα που περιγράφεται από την κανονική έκφραση $0^*1^*00^*$

Απάντηση

Κάθε κανονική έκφραση αναπαριστάνεται με NFA. Το συγκεκριμένο NFA είναι το ακόλουθο:

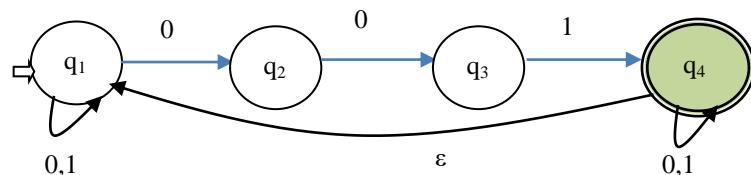


2) Δώστε την κανονική έκφραση και το πεπερασμένο αυτόματο που αναγνωρίζει το σύνολο των συμβολοσειρών που αποτελούνται από 0 και 1 και περιέχουν μια τουλάχιστον εμφάνιση του προτύπου 001.

Απάντηση

Η κανονική έκφραση είναι: $(0 \mid 1)^*(001)^+(0 \mid 1)^*$

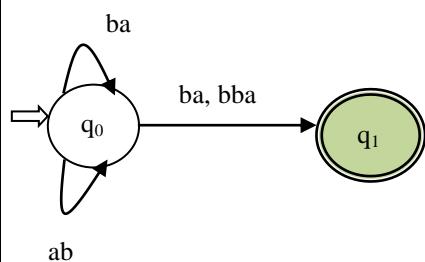
Το NFA είναι:



3) Δίνεται η κανονική έκφραση $(ba \cup ab)^* \cdot (ba \cup bba)$. Να κατασκευάσετε το αντίστοιχο NFA που αντιστοιχεί σε αυτή

Απάντηση

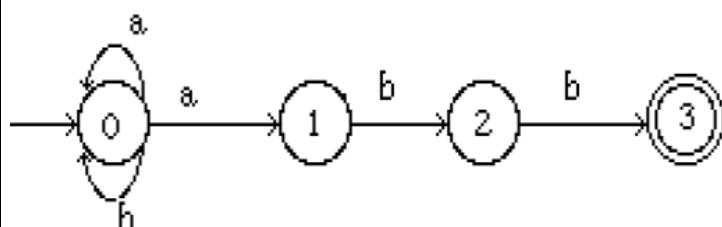
Το NFA για τη γλώσσα που αναγνωρίζει συμβολοσειρές που τελειώνουν σε ba ή σε bba είναι το ακόλουθο:



4) Να κατασκευάσετε το NFA για την ακόλουθη κανονική έκφραση $(a \mid b)^*abb$

Απάντηση

NFA

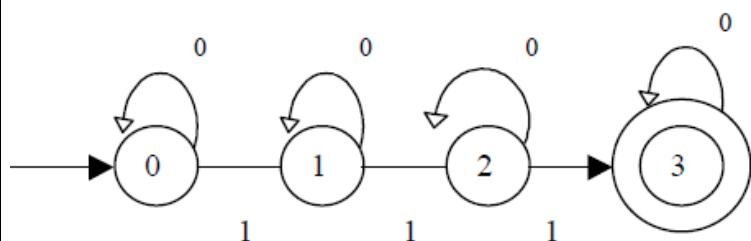


5) Να γραφεί η κανονική έκφραση και το αντίστοιχο NFA για τη γλώσσα που περιλαμβάνει όλες τις συμβολοσειρές από 0 και 1 οι οποίες περιλαμβάνουν τρία 1 και απροσδιόριστο αριθμό από «0» π.χ. 111, 0111, 010011, 10101000 κ.λ.π.

Απάντηση

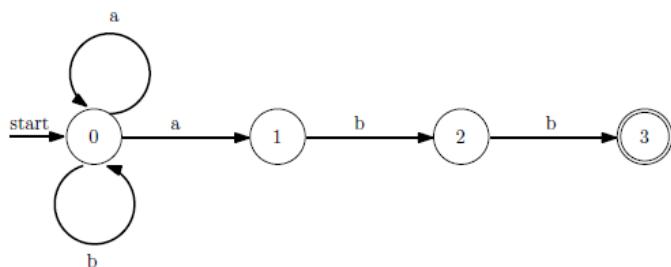
Η Κανονική Έκφραση που περιγράφει τις παραπάνω συμβολοσειρές είναι η $0^*10^*10^*10^*$

Το αντίστοιχο NFA είναι το εξής:



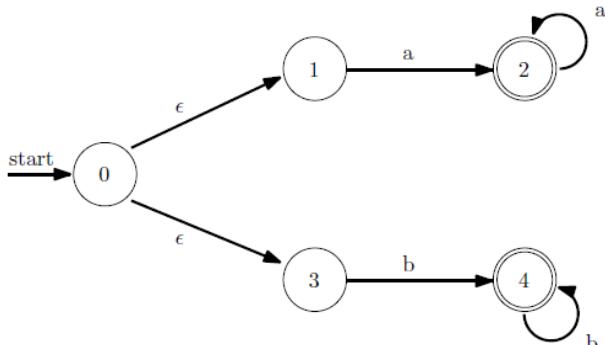
6) NFA για την κανονική έκφραση $(a | b)^*abb$

Απάντηση



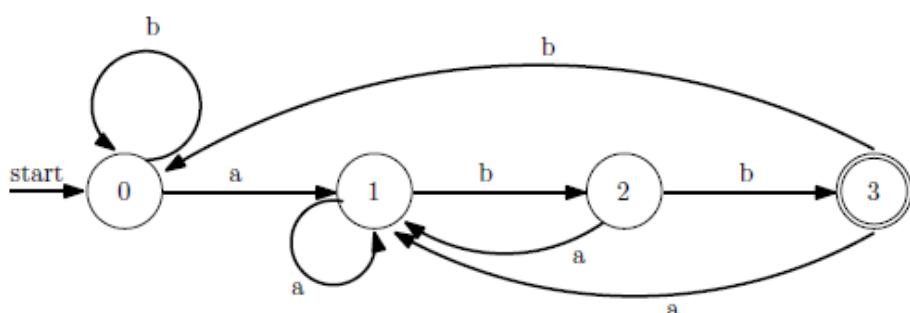
7) NFA για την κανονική έκφραση $aa^* | bb^*$

Απάντηση



8) DFA για την κανονική έκφραση $(a | b)^*abb$

Απάντηση



2.3 Θέμα 2 Ιούνιος 2012

Δίνονται οι παρακάτω 3 κανονικές εκφράσεις (regular expressions)

1. $a(bb)^*bc$ με αλφάβητο a, b, c

2. $(abc^+)^+$ με αλφάβητο a, b, c

3. $00(1|0)^*11$ με αλφάβητο 0, 1

όπου $x^+ = xx^*$

(α) Περιγράψτε τις λεξικές μονάδες (tokens) που παράγουν οι κανονικές εκφράσεις. Παρουσιάστε μερικά χαρακτηριστικά tokens από κάθε κανονική έκφραση

(β) Παρουσιάστε τα διαγράμματα καταστάσεων-μεταβάσεων που αναγνωρίζουν τις τρεις παραπάνω κανονικές εκφράσεις.

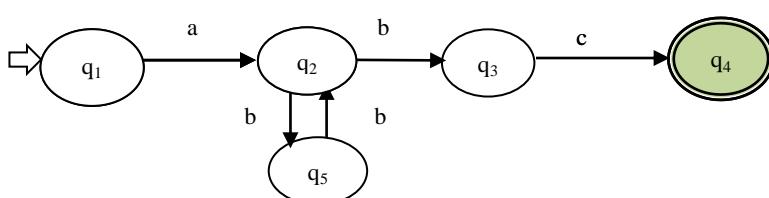
Απάντηση

(α)

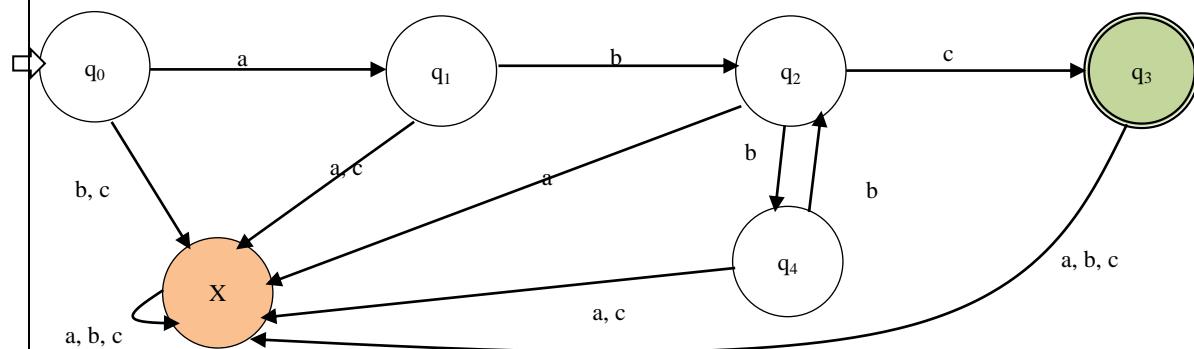
- Η 1^η κανονική έκφραση παράγει συμβολοσειρές (tokens) που αρχίζουν με a, τελειώνουν με c και ενδιάμεσα περιέχουν περιττό πλήθος από b. Το συνολικό μήκος των συμβολοσειρών είναι περιττό. Χαρακτηριστικά παραδείγματα tokens που παράγει η 1^η κανονική έκφραση είναι: abc, abbbc, abbbbcc κ.λ.π.
- Η 2^η κανονική έκφραση παράγει συμβολοσειρές που περιέχουν τουλάχιστον μια φορά το αλφαριθμητικό abc και ο χαρακτήρας c εμφανίζεται σε κάθε τέτοια συμβολοσειρά τουλάχιστον μια φορά. Αρχίζουν με ab και τελειώνουν σε c. Χαρακτηριστικά tokens που παράγει η 2^η κανονική έκφραση είναι: abc, abcc, abcabc, abccabccc κ.λ.π.
- Η 3^η κανονική έκφραση παράγει συμβολοσειρές που αρχίζουν με 00 και τελειώνουν σε 11 και ενδιάμεσα μπορεί να περιέχουν οποιαδήποτε συμβολοσειρά από 0 και/ή 1 συμπεριλαμβανόμενης της κενής συμβολοσειράς. Χαρακτηριστικά tokens που παράγει η 3^η κανονική έκφραση είναι: 0011, 00011, 00111, 000011010111 κ.λ.π.

(β)

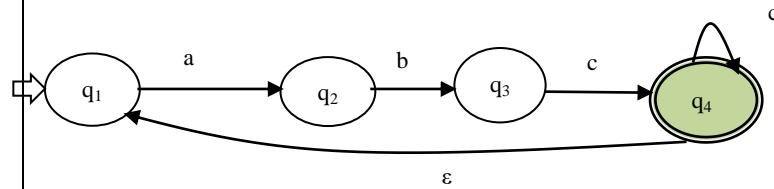
Το NFA για την Κανονική Έκφραση $a(bb)^*bc$ είναι:



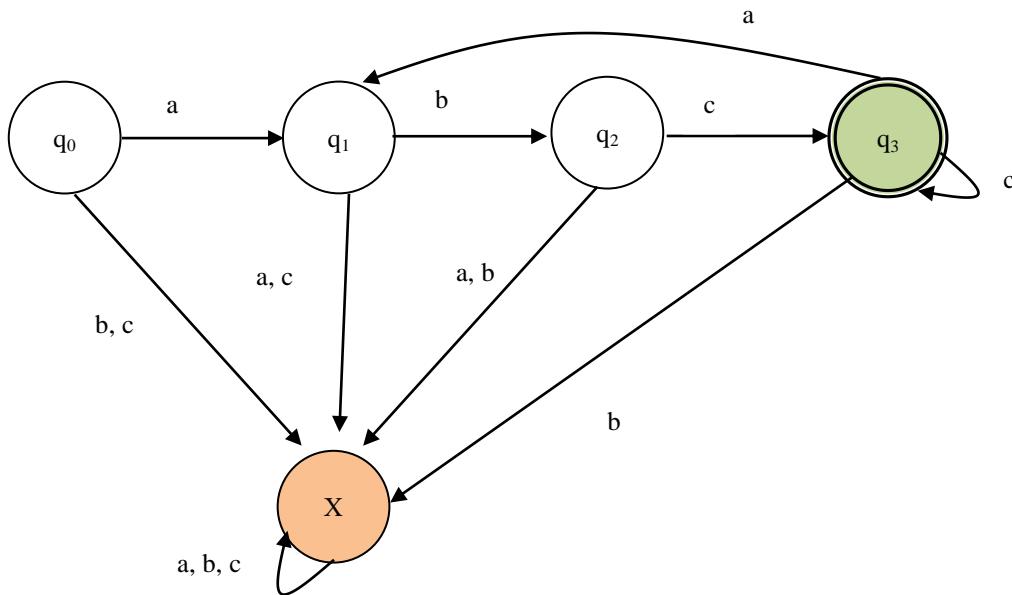
Το DFA για την Κανονική Έκφραση $a(bb)^*bc$ είναι:



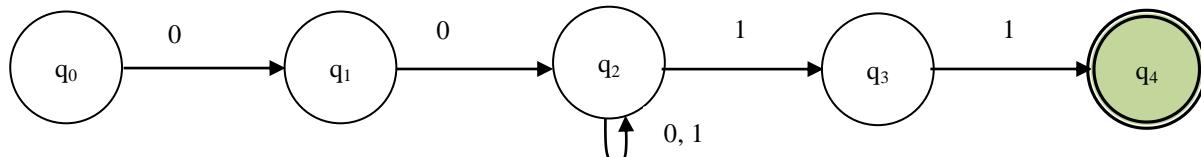
2. Το NFA για την κανονική έκφραση $(abc^+)^+$ είναι:



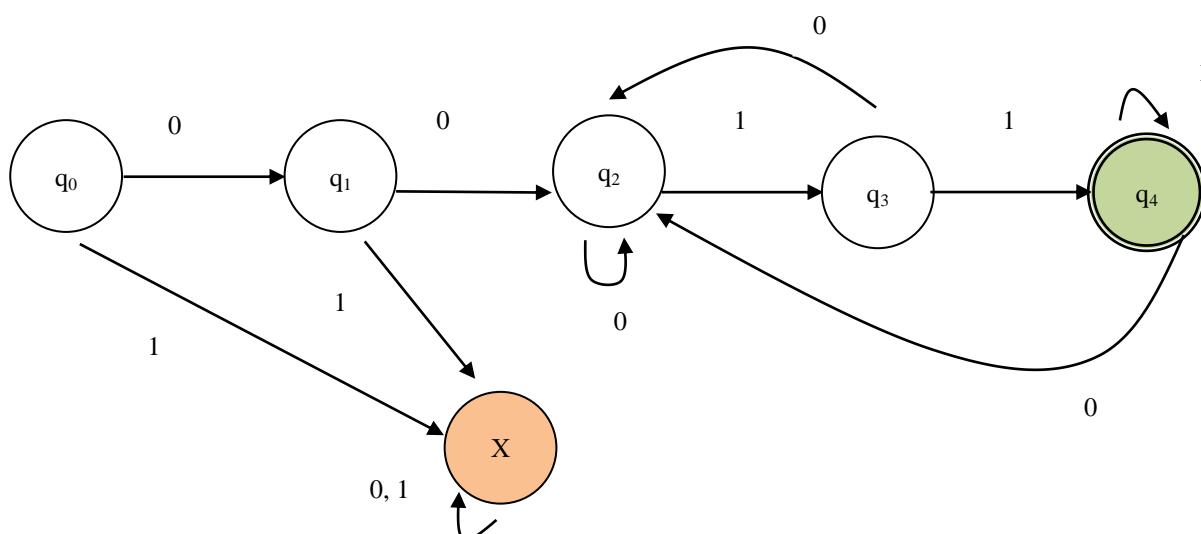
Το DFA για την κανονική έκφραση $(abc^+)^+$ είναι:



3. Το NFA για την Κανονική Έκφραση $00(1|0)^*11$ είναι:



Το DFA για την Κανονική Έκφραση $00(1|0)^*11$ είναι:



2.4 Θέμα 2 Ιούνιος 2008

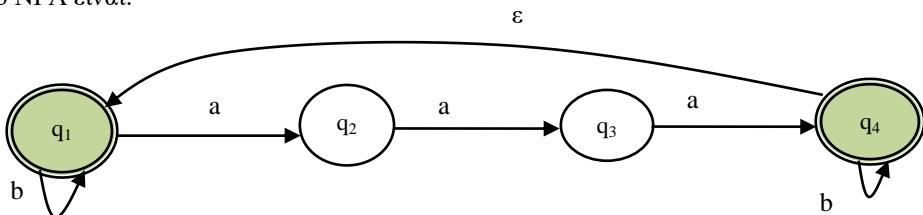
Να παρουσιάσετε τα διαγράμματα καταστάσεων-μεταβάσεων για δύο πεπερασμένα αυτόματα που αναγνωρίζουν:

(α) Όλες τις ακολουθίες χαρακτήρων που αποτελούνται από a και b (συμπεριλαμβανομένης της κενής ακολουθίας) οι οποίες περιέχουν τα a μόνο σε ομάδες των τριών π.χ. aaa , $\beta\alpha a\beta\beta$, $\alpha a\beta\beta\beta a a\beta$ κ.λ.π

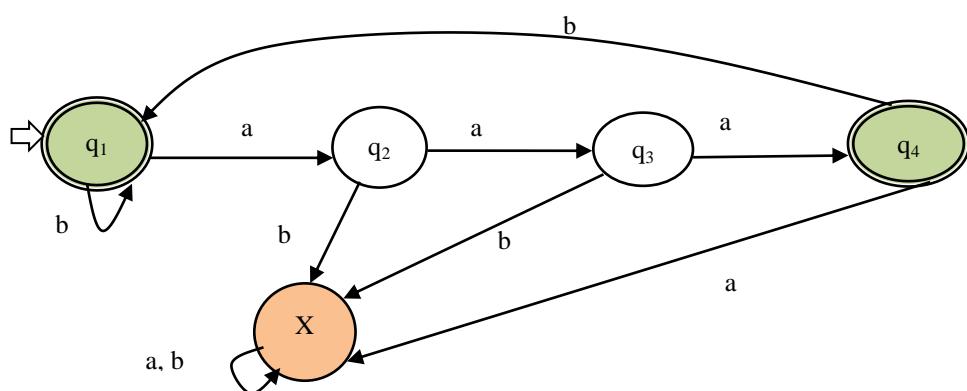
(β) Όλες τις ακολουθίες χαρακτήρων που αποτελούνται από a και b με εναλλασσόμενα a και b οι οποίες έχουν τουλάχιστον ένα a ή ϵ να β π.χ. a , $\beta a \beta$, $a \beta a \beta a \beta a \beta$ κ.λ.π.

Απάντηση

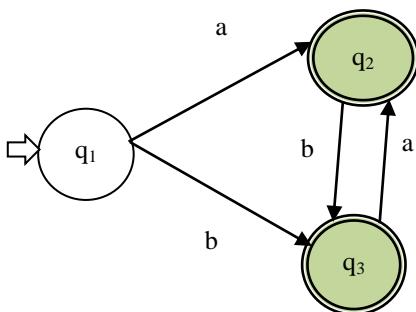
(α) To NFA είναι:



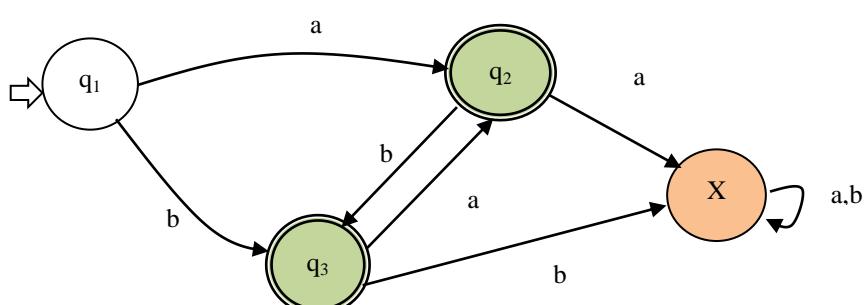
To DFA είναι:



(b) To NFA είναι:



To DFA είναι:



2.5 Θέμα 2 Ιούνιος 2007

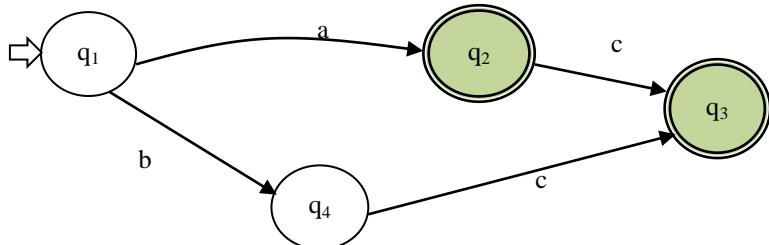
Να παρουσιάσετε τα διαγράμματα καταστάσεων-μεταβάσεων για δύο πεπερασμένα αυτόματα που αναγνωρίζουν:

(α) Όλα τα λεξικά αντικείμενα (tokens) της γλώσσας {a, ac, bc}

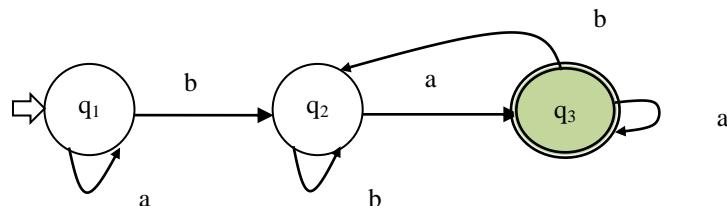
(β) Όλες τις ακόλουθες από a και b οι οποίες έχουν τουλάχιστον ένα b και ο τελευταίος χαρακτήρας είναι το a (δηλ. περιέχουν τουλάχιστον ένα a)

Απάντηση

(α) Όταν η γλώσσα είναι πεπερασμένη τότε κατασκευάζουμε **ΜΟΝΟ NFA γιαντή**. Το NFA για τη γλώσσα {a, ac, bc} είναι το ακόλουθο:



(β) Το DFA είναι:



2.6 Θέμα 2 Ιούνιος 2009

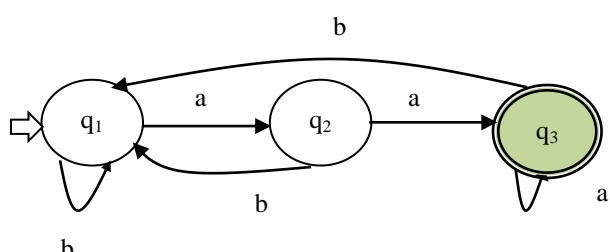
Να κατασκευάσετε πεπερασμένα αυτόματα για τις ακόλουθες γλώσσες στο αλφάριθμο {a, b}:

(α) οι συμβολοσειρές να τελειώνουν σε aa

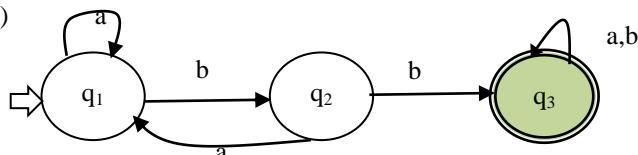
(β) οι συμβολοσειρές να έχουν τουλάχιστον 2 συνεχόμενα b

Απάντηση

(α)



(β)



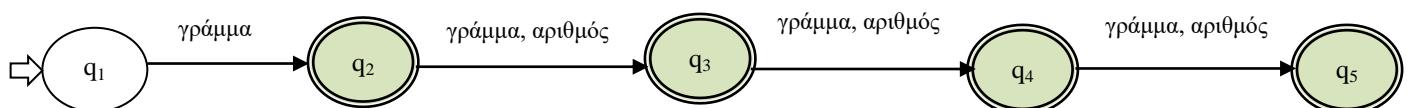
2.7 Θέμα 2 Σεπτέμβριος 2010

Να δημιουργηθεί πεπερασμένο αυτόματο που αναγνωρίζει:

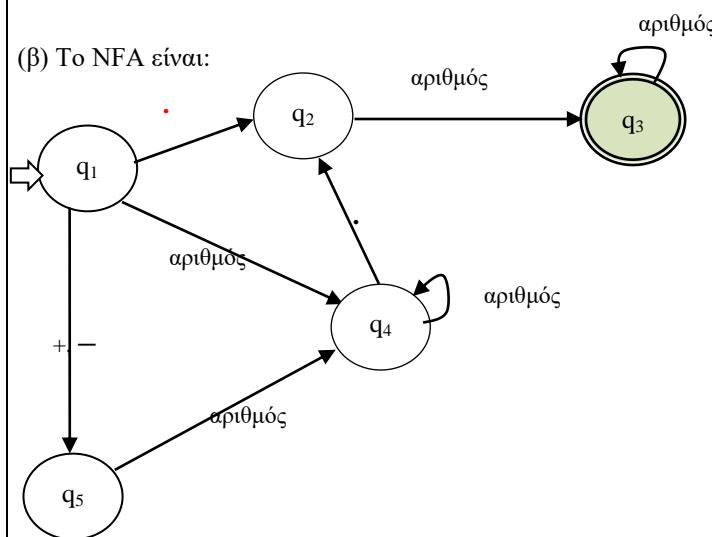
- (α) από 1 έως 4 χαρακτήρες (γράμματα ή αριθμούς) και ο πρώτος χαρακτήρας να είναι οπωσδήποτε γράμμα
- (β) αναγνωρίζει tokens που έχουν οπωσδήποτε δεκαδικό μέρος και πιθανότατα ακέραιο μέρος με ή χωρίς πρόσημο π.χ. 5.25, +237.85, -56.231, 0.00, .00

Απάντηση

(α) To NFA είναι:



(β) To NFA είναι:



2.8 Θέμα 2 Σεπτέμβριος 2013

(a)

1. Δίνεται η κανονική έκφραση (regular expression):

$$(a((a|b)^*a)?) \mid (b((a|b)^*b)?) \text{ με αλφάβητο } a, b.$$

Περιγράψτε τις λεξικές μονάδες (tokens) που παράγει η κανονική έκφραση. Δώστε μερικά παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση

(b)

1. Να παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει τις λέξεις με αλφάριθμο a, b οι οποίες έχουν ένα τουλάχιστον γράμμα και δεν έχουν δύο ίδια γράμματα συνεχόμενα. Παραδείγματα αποδεκτών λέξεων $a, bab, ababab$

2. Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο.

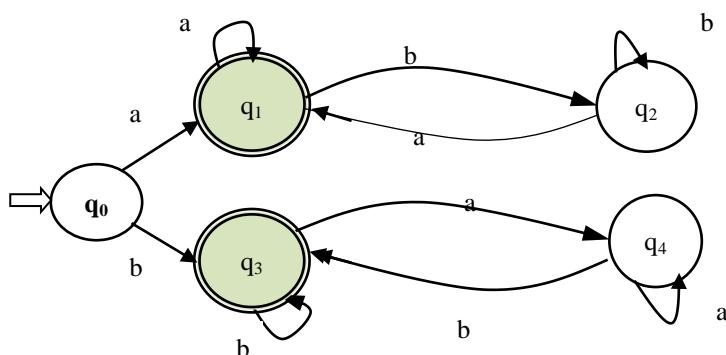
Απάντηση

(a)

1. Η κανονική έκφραση παράγει μη κενές συμβολοσειρές από a, b που αρχίζουν και τελειώνουν με το ίδιο σύμβολο.

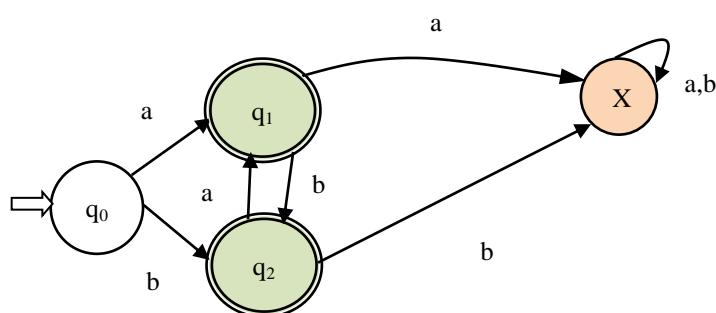
Παραδείγματα: a, b, aba, bab κ.λ.π.

2. Το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) που αναγνωρίζει την παραπάνω κανονική έκφραση.



(b)

(1) Το διάγραμμα καταστάσεων-μεταβάσεων (DFA) είναι το ακόλουθο:



2. Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι: $a(ba)^*b? \mid b(ab)^*a?$

Παρατήρηση

$\alpha? = a \cup e$ και $\beta? = b \cup e$

2.9 Θέμα 4 Ιούνιος 2012

(a)

1. Δίνεται η κανονική έκφραση (regular expression): $((0|1)(0|1)(0|1)(0|1))^*$ με αλφάβητο 0, 1

Περιγράψτε τις λεξικές μονάδες (tokens) που παράγει η κανονική έκφραση. Δώστε μερικά παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση.

(b)

1. Να παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει τους δυαδικούς αριθμούς οι οποίοι έχουν **ένα τουλάχιστον 0 και όλα τα 1 (αν υπάρχουν)** έχουν δεξιά και αριστερά τους δύο ή περισσότερα 0. Τα 1 εμφανίζονται **ένα τη φορά**. Παραδείγματα αποδεκτών tokens 00, 001000100010000

2. Παρουσιάστε την κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο.

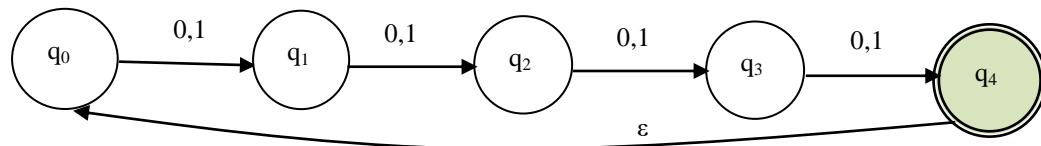
Απάντηση

a)

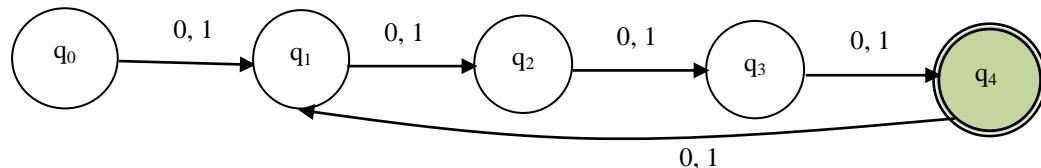
Η κανονική έκφραση αναγνωρίζει τις μη κενές συμβολοσειρές από 0 και 1 που το μήκος τους είναι πολλαπλάσιο του 4.

Παραδείγματα λεκτικών μονάδων είναι: 0000, 1111, 1001, 0110, 01101010, 00101001, 1111000010101010 κ.λ.π.

To NFA που αναγνωρίζει την παραπάνω κανονική έκφραση είναι το ακόλουθο:

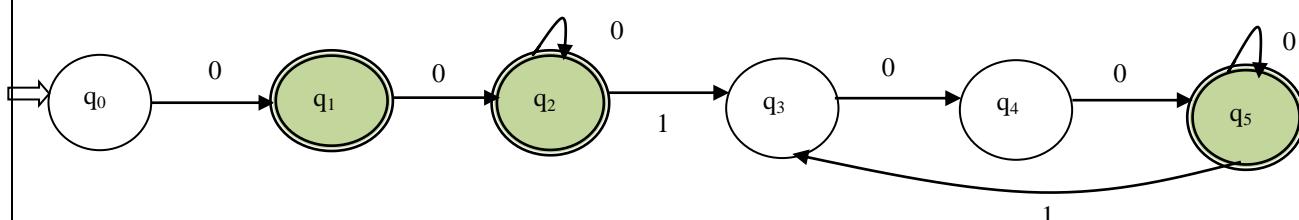


To DFA που αναγνωρίζει την παραπάνω κανονική έκφραση είναι το ακόλουθο:

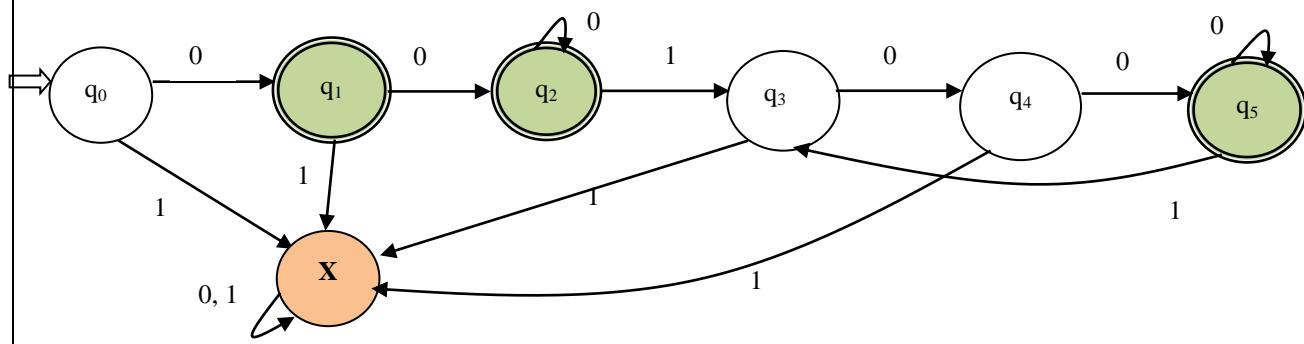


b)

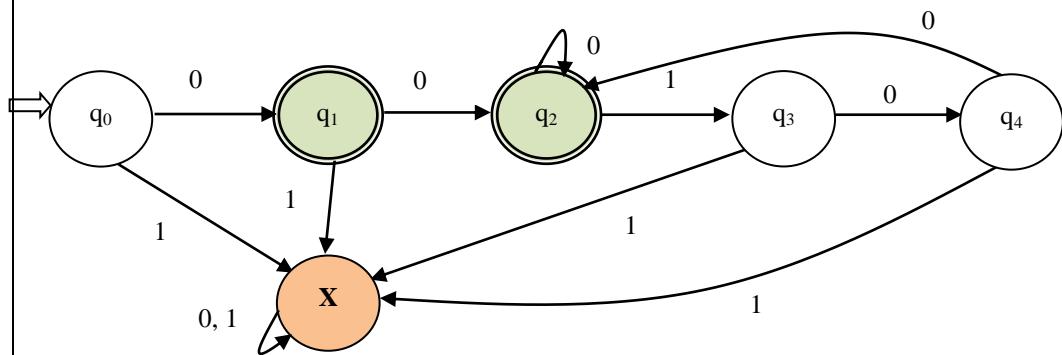
To NFA που αναγνωρίζει τη γλώσσα είναι το ακόλουθο:



Το DFA που αναγνωρίζει τη γλώσσα είναι το ακόλουθο:



Ένα εναλλακτικό ισοδύναμο DFA που αναγνωρίζει τη γλώσσα είναι το ακόλουθο:



- Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι $00^+ (100^+)^* | 0$
- Εναλλακτικά μια ισοδύναμη κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι $0^+ | (00^+ 100^+)^+$

2.10 Θέμα 1 Ιούνιος 2014

1. Δίνεται η κανονική έκφραση $(0^* 11^+ 0^*)^+$

α) Ποιες συμβολοσειρές αναγνωρίζει αυτή η κανονική έκφραση. Δώστε 2 παραδείγματα συμβολοσειρών

β) Να δώσετε το διάγραμμα καταστάσεων – μεταβάσεων του πεπερασμένου αυτομάτου για την κανονική έκφραση

2. Να φτιάξετε ένα πεπερασμένο αυτόματο με **περιττό αριθμό 0 και περιττό αριθμό 1** το οποίο να περιέχει ένα τουλάχιστον 0 και ένα τουλάχιστον 1.

Απάντηση

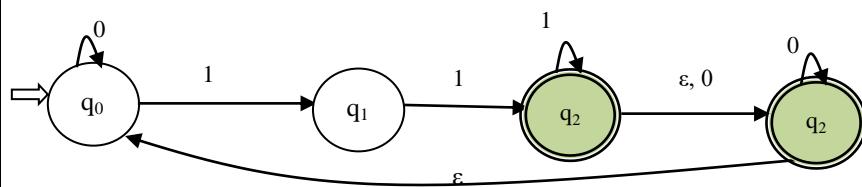
1.

α)

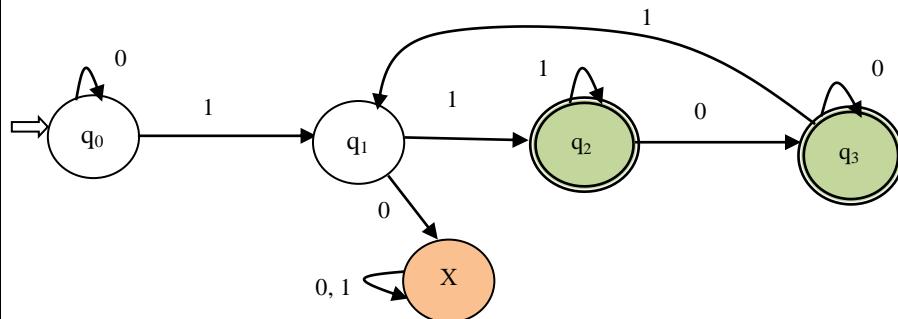
Συμβολοσειρές από 0 και 1 οι οποίες περιέχουν τουλάχιστον 2 διαδογικούς άσσους και το μήκος τους είναι τουλάχιστον 2 γαρακτήρες.

Παραδείγματα συμβολοσειρών είναι: 11, 111, 01100, 011, 11100, 000111000, 011001110001100 κ.λ.π.

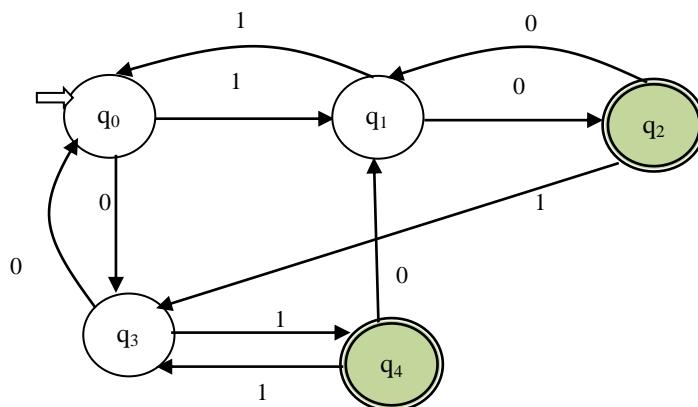
β) Το διάγραμμα καταστάσεων – μεταβάσεων **του μη ντετερμινιστικού πεπερασμένου αυτόματου (NFA)** για την κανονική έκφραση είναι το εξής:



Το διάγραμμα καταστάσεων – μεταβάσεων **του ντετερμινιστικού πεπερασμένου αυτόματου (DFA)** για την κανονική έκφραση είναι το εξής:



2. Το διάγραμμα καταστάσεων – μεταβάσεων **του ντετερμινιστικού πεπερασμένου αυτόματου (DFA)** είναι το εξής:



2.11 Θέμα 1 Σεπτέμβριος 2014 και Θέμα 1 Φεβρουάριος 2018

(a)

- Δίνεται η κανονική έκφραση $0^*1?0^*1?0^*$ με αλφάβητο 0, 1

Περιγράψτε τις λεξικές μονάδες που παράγει η κανονική έκφραση. Δώστε μερικά παραδείγματα

- Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση.

(b)

- Να παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι **αρχίζουν με τουλάχιστον τρία 0 και τελειώνουν με τουλάχιστον ένα 1**.

- Γράψτε μια **κανονική έκφραση** που αντιστοιχεί στο παραπάνω αυτόματο

Απάντηση

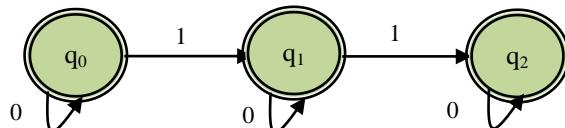
(a)

1.

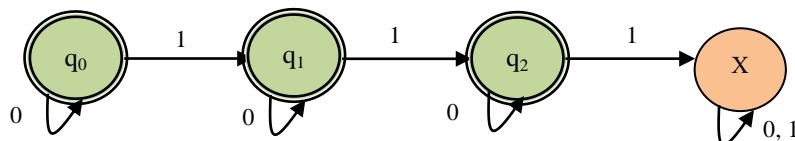
Η Κανονική Έκφραση περιγράφει συμβολοσειρές **από 0 και 1 που έχουν το πολύ δύο άσσους** συμπεριλαμβανομένης και της κενής συμβολοσειράς

Παραδείγματα συμβολοσειρών: Η κενή συμβολοσειρά, 11, 0, 000, 001000, 010010, 0001000010000

- Το **MΗ ντετερμινιστικό πεπερασμένο αυτόματο NFA** είναι:

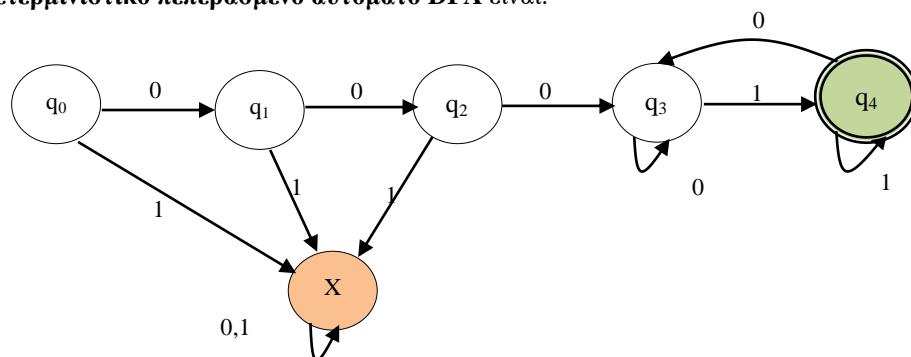


Το **ντετερμινιστικό πεπερασμένο αυτόματο DFA** είναι:



(b)

- Το **ντετερμινιστικό πεπερασμένο αυτόματο DFA** είναι:



- Η κανονική έκφραση που περιγράφει τη γλώσσα είναι η ακόλουθη: **$000^+ (0|1)^*1^+$**

Παρατηρήσεις

- 1? σημαίνει ένας άσσος ή κανένας άσσος
- 0? σημαίνει ένα μηδενικό ή κανένα μηδενικό
- Το ? αφορά αποκλειστικά το χαρακτήρα μπροστά από τον οποίο βρίσκεται

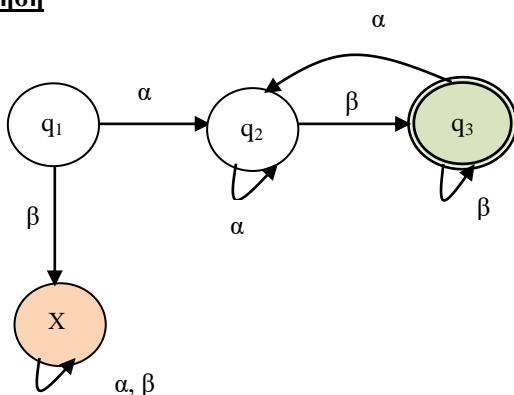
- 1^* σημαίνει 0 άσσοι, 1, 2, 3,...άπειροι άσσοι
- 1^+ σημαίνει 1 άσσος, 2, 3,...άπειροι άσσοι
- $(0|1)^*$ σημαίνει τα πάντα από 0 και/ή 1
- Στο DFA από ΚΑΘΕ κατάσταση απαιτείται μετάβαση με όλους τους χαρακτήρες του αλφαριθμητού
- Στο NFA κάνουμε τις ελάχιστες μεταβάσεις που αναγνωρίζουν σωστά τη γλώσσα

2.12 Θέμα 2 Ατυπη 2011

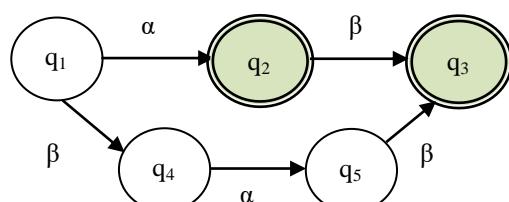
- α) Να κατασκευάσετε DFA που αρχίζει με α έχει τουλάχιστον ένα β και τελειώνει σε β. Το αλφάριθμητο είναι το $\{\alpha, \beta\}$.
- β) Να κατασκευάσετε NFA που αναγνωρίζει τις συμβολοσειρές α , $\alpha\beta$, $\beta\alpha\beta$
- γ) Να κατασκευάσετε NFA που αναγνωρίζει προσημασμένους ή μη ακέραιους
- δ) Να κατασκευάσετε NFA που αναγνωρίζει ονόματα μεταβλητών τα οποία αρχίζουν με γράμμα και ακολουθούνται από γράμματα και ψηφία

Απάντηση

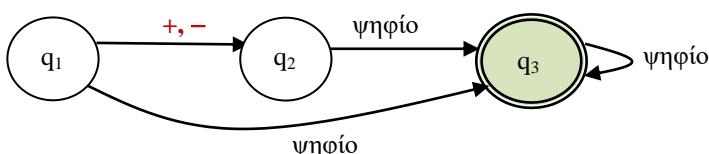
α)



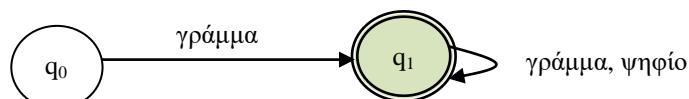
- β) Όταν κατασκευάζουμε πεπερασμένο αυτόματο που αναγνωρίζει πεπερασμένη γλώσσα **AYTO MPOREI NA EINAI MONO NFA**. Το συγκεκριμένο NFA αναγνωρίζει αποκλειστικά τις συμβολοσειρές α , $\alpha\beta$, $\beta\alpha\beta$ και καμία άλλη.



- γ) Το NFA που αναγνωρίζει προσημασμένους ή μη ακέραιους είναι:



- δ) Το NFA που αναγνωρίζει ονόματα μεταβλητών τα οποία αρχίζουν με γράμμα και ακολουθούνται από γράμματα και ψηφία



2.13 Θέμα 1 Ιούνιος 2015

(a). Δίνεται η κανονική έκφραση (regular expression): $(00)^*(11)^*1$ με αλφάβητο 0, 1.

1. Περιγράψτε τις λεξικές μονάδες (tokens) που παράγει η κανονική έκφραση. Δώστε παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση

(b)

1. Να παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι δεν περιέχουν παραπάνω από δύο συνεχόμενα 0. Το είναι αποδεκτός αριθμός.

2. Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο

Απάντηση

(a)

1.

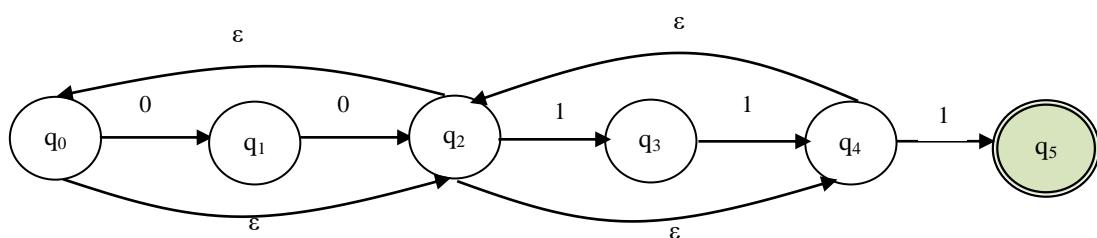
Παραδείγματα λεξικών μονάδων (tokens) που παράγει η κανονική έκφραση είναι: 1, 001, 111, 00001, 111111, 0000111 κ.λ.π.

Η κανονική έκφραση περιγράφει τις λεξικές μονάδες:

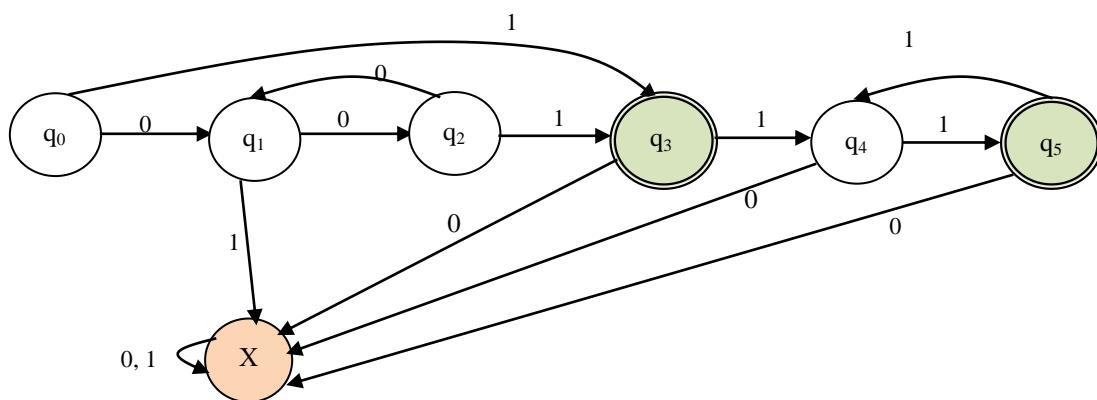
- που τελειώνουν σε 1
- είναι περιττού μήκους
- έχουν τουλάχιστον ένα 1
- ξεκινούν με άρτιο πλήθος 0 και τελειώνουν σε περιττό πλήθος 1
- τα 0 προηγούνται των 1

2.

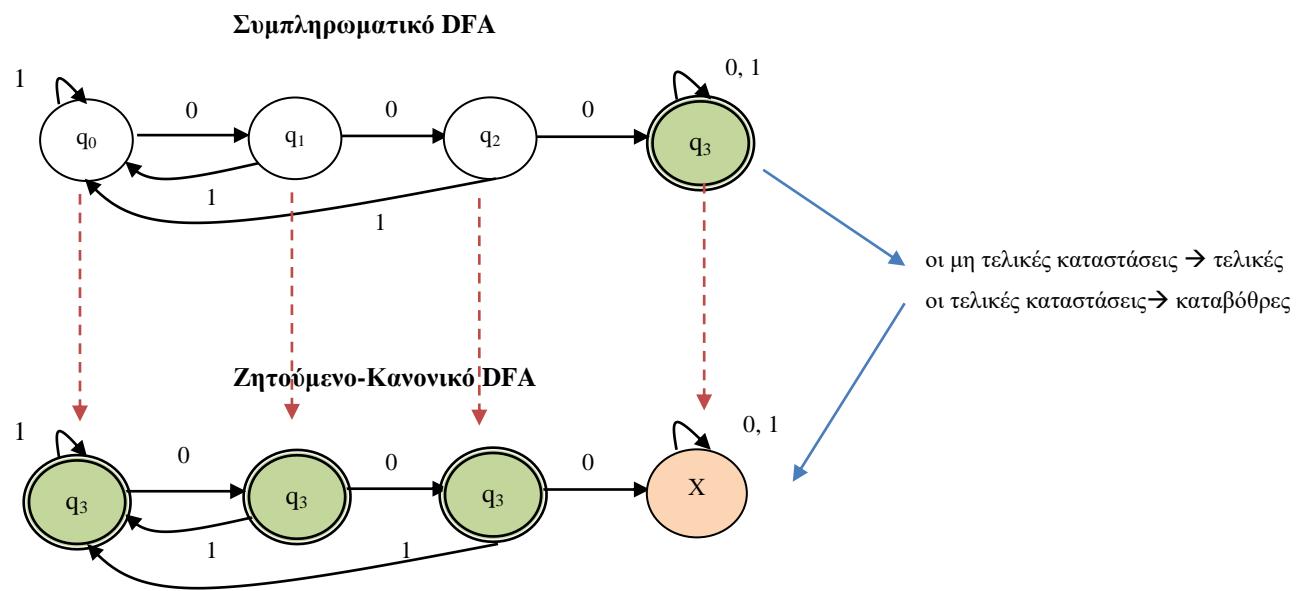
Το ΜΗ ντετερμινιστικό πεπερασμένο αυτόματο (NFA) είναι το εξής:



Το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) είναι το εξής:



(b) Όταν το DFA έχει άρνηση πρέπει να φτιάξουμε πρώτα το συμπληρωματικό DFA (δηλ. αυτό που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι περιέχουν παραπάνω από δύο συνεχόμενα 0) και μετά σε αυτό να κάνουμε εναλλαγή καταστάσεων ώστε να προκύψει το ζητούμενο DFA (δηλ. οι μη τελικές καταστάσεις να γίνουν τελικές και οι τελικές να γίνουν καταβόθρες).



2. Η κανονική έκφραση που περιγράφει το αυτόματο είναι: $(1^* 0? 1^*)^* (0? 1^*)^*$

Παρατηρήσεις

- 1? σημαίνει ένας άσσος ή κανένας άσσος
- 0? σημαίνει ένα μηδενικό ή κανένα μηδενικό
- Το ? αφορά αποκλειστικά το χαρακτήρα μπροστά από τον οποίο βρίσκεται

2.14 Θέμα 1 Σεπτέμβριος 2015

(a) 1. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3. Π.χ. οι αριθμοί 101 110000 011010111 είναι αποδεκτοί

2. Γράψετε μια κανονική έκφραση που να αντιστοιχεί στο παραπάνω αυτόματο

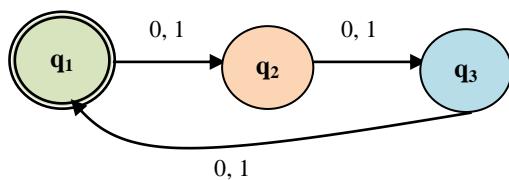
(b) 1. Δίνεται η κανονική έκφραση $0^* | 00^+ (100^+)^*$ με αλφάριθμο 0, 1. Περιγράψτε τις λεξικές μονάδες που παράγει η κανονική έκφραση. Δώστε παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει την παραπάνω κανονική έκφραση

Απάντηση

(a) Το DFA που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3 είναι το ακόλουθο:

1.



2. Η κανονική έκφραση που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3 είναι η ακόλουθη:

$$((0|1)(0|1)(0|1))^*$$

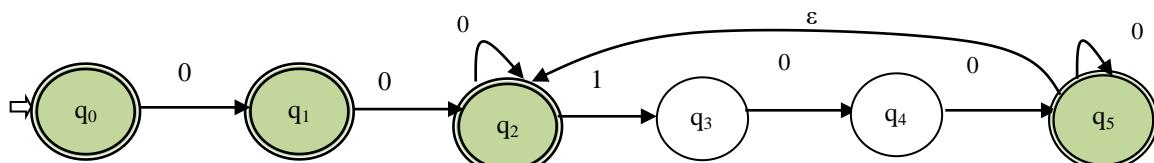
(b)

1. Η κανονική έκφραση $0^* | 00^+ (100^+)^*$ αναγνωρίζει λεξικές μονάδες (συμβολοσειρές) οι οποίες μπορούν να έχουν μόνο 0 και αν

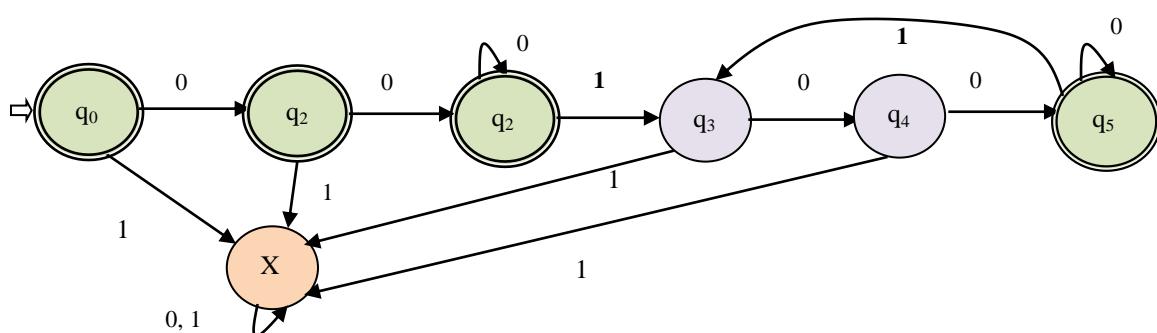
υπάρχει 1 τότε αριστερά και δεξιά από κάθε 1 βρίσκονται τουλάχιστον δύο 0. Στις συμβολοσειρές συμπεριλαμβάνεται και η κενή.

Παραδείγματα συμβολοσειρών είναι 0, 00, 00100, 001000, 00100100, κ.λ.π.

2. Το **ΜΗ ντετερμινιστικό πεπερασμένο αυτόματο (NFA)** που αναγνωρίζει την παραπάνω κανονική έκφραση είναι:



Το **ντετερμινιστικό πεπερασμένο αυτόματο (DFA)** που αναγνωρίζει την παραπάνω κανονική έκφραση είναι:



2.15 Θέμα 1 Φεβρουάριος 2015

Δίνονται οι παρακάτω δύο κανονικές εκφράσεις (regular expressions):

1. $(abc^+)^+$ με αλφάβητο a b c

2. $00(1|0)^*11$ με αλφάβητο 0 1

(a) Περιγράψτε τις λεξικές μονάδες (tokens) που παράγουν οι δύο κανονικές εκφράσεις. Δώστε μερικά χαρακτηριστικά tokens από κάθε κανονική έκφραση

(b) Παρουσιάστε τα διαγράμματα καταστάσεων-μεταβάσεων για τα δύο πεπερασμένα αυτόματα που αναγνωρίζουν τις παραπάνω δύο κανονικές εκφράσεις

Απάντηση

1.

(a) Η 1^η κανονική έκφραση παράγει συμβολοσειρές που περιέχουν τουλάχιστον μια υποσυμβολοσειρά της μορφής abc, στην οποία εμφανίζεται τουλάχιστον μια φορά ο χαρακτήρας c

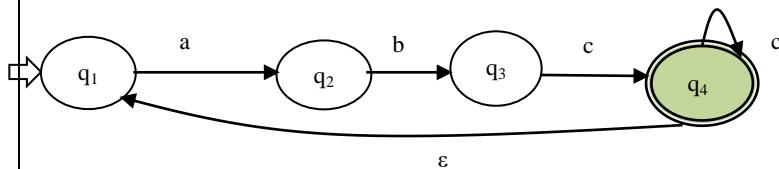
Παραδείγματα τέτοιων συμβολοσειρών είναι: abc, abcc, abccc, abcabccabccc κ.λ.π.

(b) Η 2^η κανονική έκφραση παράγει συμβολοσειρές που αρχίζουν με 00, τελειώνουν σε 11 και ενδιάμεσα μπορεί να υπάρχει οποιαδήποτε συμβολοσειρά από 0,1.

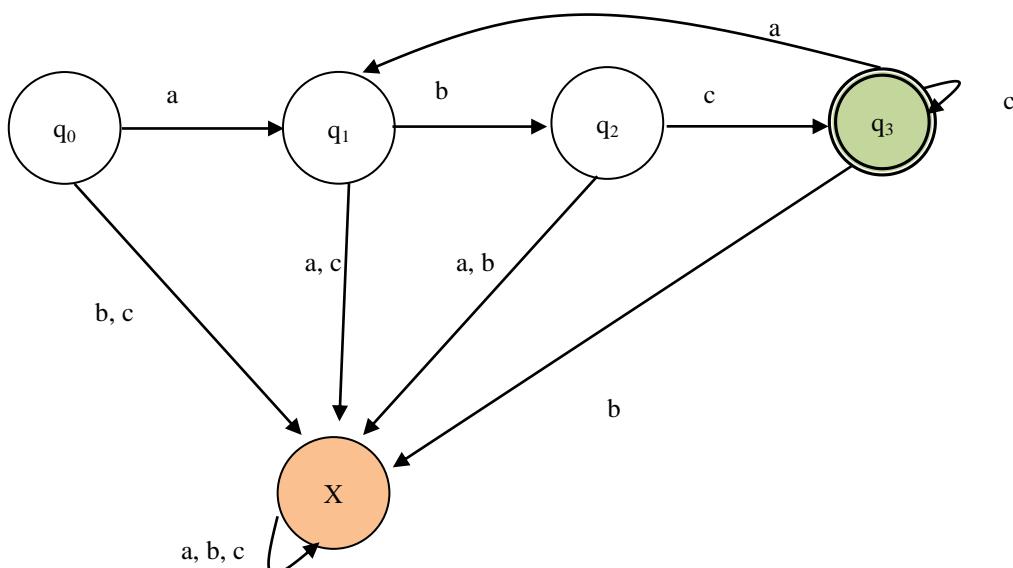
Παραδείγματα τέτοιων συμβολοσειρών είναι: 0011, 00111, 00011, 000101011 κ.λ.π.

2.

(a) To NFA για την κανονική έκφραση $(abc^+)^+$ είναι:

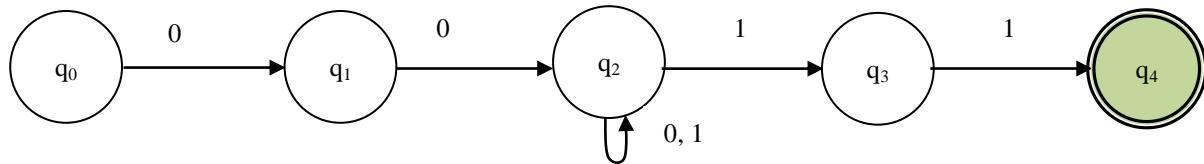


To DFA για την κανονική έκφραση $(abc^+)^+$ είναι:

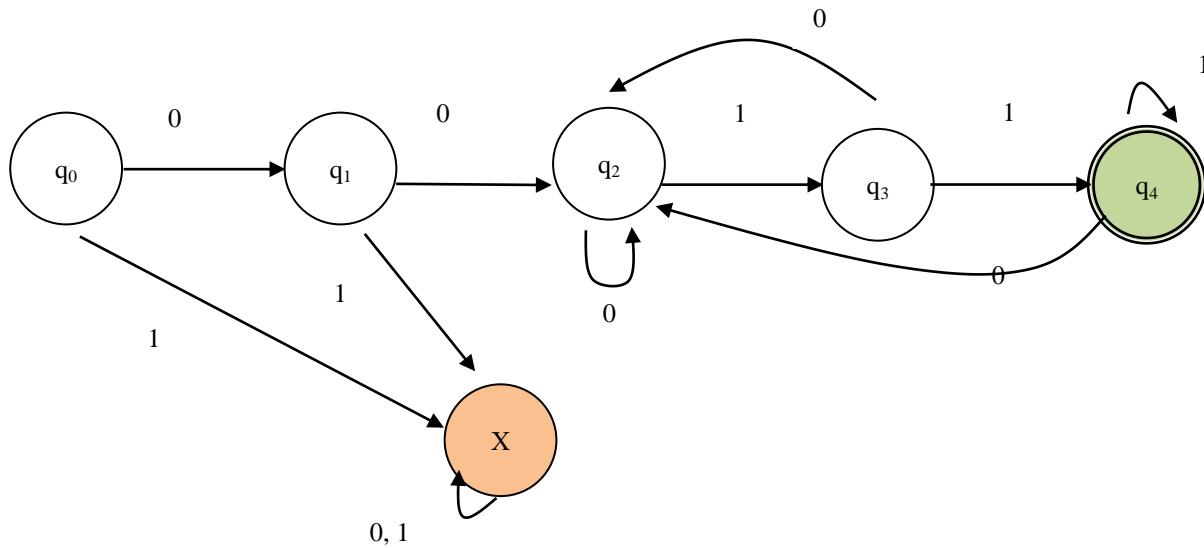


2.

(b) Το NFA για την Κανονική Έκφραση $00(1|0)^*11$ είναι:



Το DFA για την Κανονική Έκφραση $00(1|0)^*11$ είναι:



2.16 Θέμα 1 Ιούνιος 2016

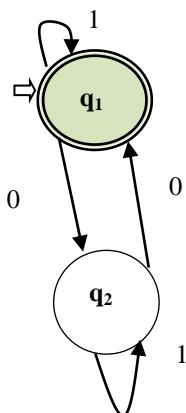
Να κατασκευάσετε τα ακόλουθα DFA:

- για άρτιο πλήθος 0
- για περιττό πλήθος 1
- με συνδυασμό των δύο προηγούμενων περιπτώσεων

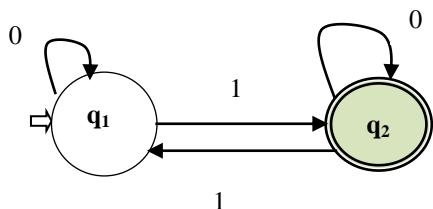
Απάντηση

1)

- To DFA για **άρτιο πλήθος 0** είναι το ακόλουθο:



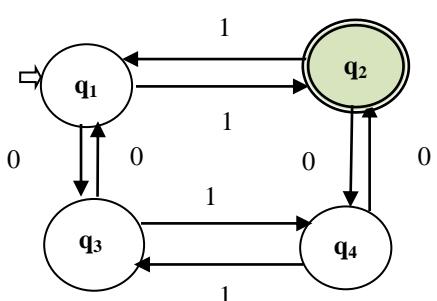
- To DFA για **περιττό πλήθος 1** είναι το ακόλουθο:



- To DFA με **συνδυασμό των 2 παραπάνω περιπτώσεων** είναι το ακόλουθο:

#0	#1	Κατάσταση
Άρτιο	Άρτιο	q1
Άρτιο	Περιττό	q2 (Τελική)
Περιττό	Άρτιο	q3
Περιττό	Περιττό	q4

Ο λόγος που κατασκευάσαμε τον πίνακα αυτό είναι για να εντοπίσουμε ευκολότερα όλες τις πιθανές καταστάσεις τις οποίες απεικονίζουμε στο DFA:



2.17 Θέμα 2 Σεπτέμβριος 2016

Δίνονται οι παρακάτω κανονικές εκφράσεις:

1. $a(bb)^*bc$ με αλφάβητο $\{a, b, c\}$
2. $(01^+)^+$ με αλφάβητο $\{0, 1\}$
3. $bb(b|a)^*aa$ με αλφάβητο $\{a, b\}$
4. $bb(b|a)^*$ με αλφάβητο $\{a, b\}$

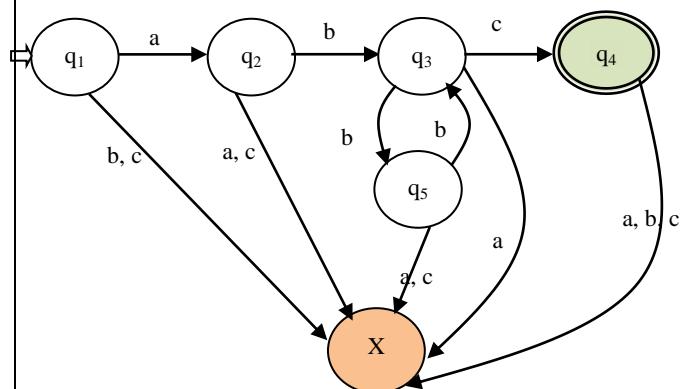
Παρουσιάστε τα διαγράμματα καταστάσεων-μεταβάσεων για τα **τέσσερα ντετερμινιστικά πεπερασμένα αυτόματα (DFA)** που αναγνωρίζουν τις ίδιες γλώσσες με τις παραπάνω κανονικές εκφράσεις.

Απάντηση

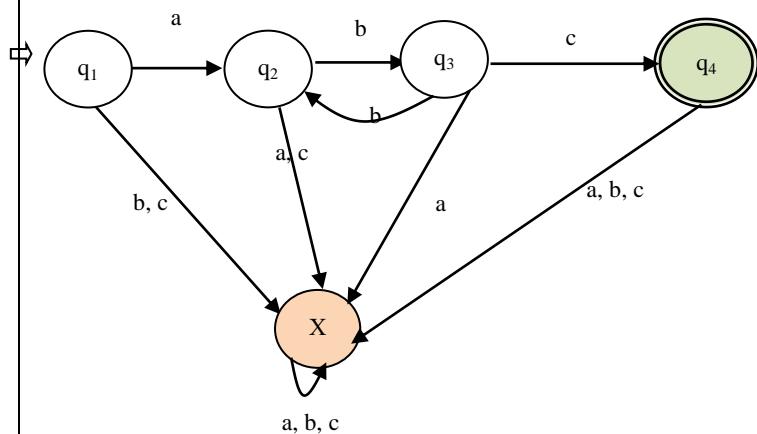
1. $a(bb)^*bc$ με αλφάβητο $\{a, b, c\}$. Παραδείγματα συμβολοσειρών abc , $abbcc$, $abbbbcb$, $abbbbbbbc$ κ.λ.π.

Συμβολοσειρές που αρχίζουν με a , τελειώνουν σε b και ενδιάμεσα υπάρχει μόνο b περιττού πλήθος b και το συνολικό μήκος των συμβολοσειρών είναι επίσης περιττό.

To DFA είναι το ακόλουθο:



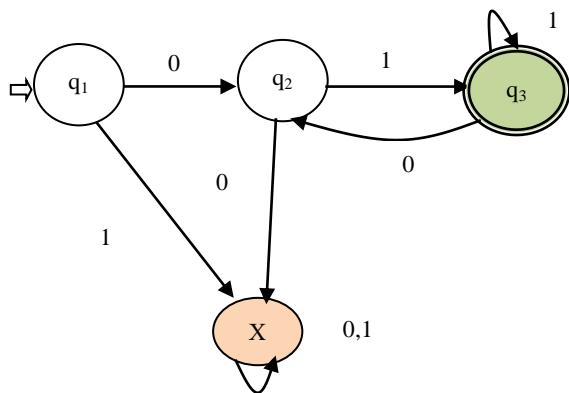
Εναλλακτικά το DFA μπορεί να γραφεί ως εξής:



2. $(01^+)^*$ με αλφάβητο $\{0, 1\}$. Παραδείγματα συμβολοσειρών 01, 011, 0111, 010111, 0110111011111 κ.λ.π.

Συμβολοσειρές με τουλάχιστον μια επανάληψη του προτύπου 01 το οποίο σε κάθε επανάληψη του θα έχει τουλάχιστον ένα 1

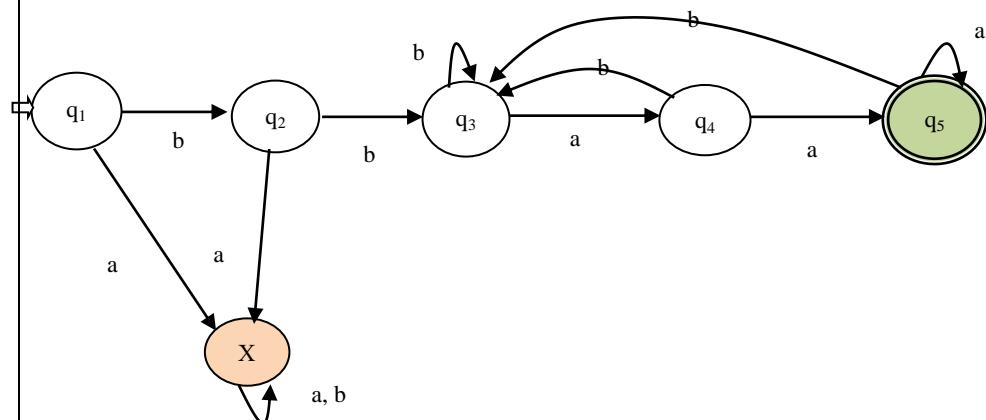
To DFA είναι το ακόλουθο:



3. $bb(b|a)^*aa$ με αλφάβητο $\{a, b\}$. Παραδείγματα συμβολοσειρών bbaa, bbababaaa, bbbaabbabb κ.λ.π.

Συμβολοσειρές που αρχίζουν με bb και τελειώνουν σε aa

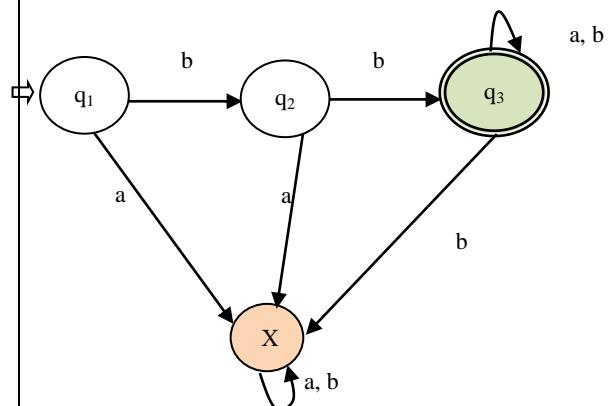
To DFA είναι το ακόλουθο:



4. $bb(b|a)^*$ με αλφάβητο $\{a, b\}$. Παραδείγματα συμβολοσειρών bb, bba, bbaaab, bbabbaab κ.λ.π.

Συμβολοσειρές που αρχίζουν με bb

To DFA είναι το ακόλουθο:



2.18 Θέμα 1 Ιούνιος 2017

(a)

1. Δίνεται η κανονική έκφραση (regular expression): $0^*1(0|1)(0|1)^*$ με αλφάριθμο 0, 1. Περιγράψτε τα χαρακτηριστικά των λεξικών μονάδων που παράγει η κανονική έκφραση. Δώστε παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το ντετερμινιστικό πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση

(b)

1. Να παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι: αν περιέχουν 1, αυτό θα ακολουθείται από ένα ακριβώς 0, εκτός από το τελευταίο 1 που θα ακολουθείται από τουλάχιστον δύο 0. Το είναι αποδεκτός αριθμός

2. Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο

Απάντηση

(a)

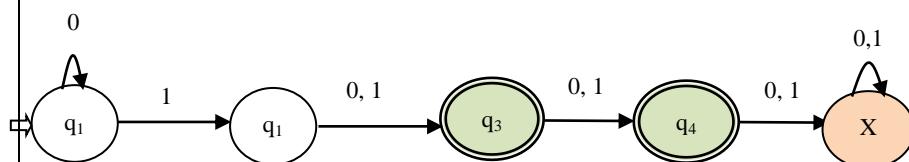
1.

Παραδείγματα Λεξικών Μονάδων: 11, 10, 011, 00110, 0000101, 0000010 κ.λ.π.

Η κανονική έκφραση παράγει συμβολοσειρές οι οποίες **περιέχουν τουλάχιστον ένα 1, το μήκος τωνς είναι τουλάχιστον 2 χαρακτήρες και περιέχουν το πολύ 3 άσσους**

2.

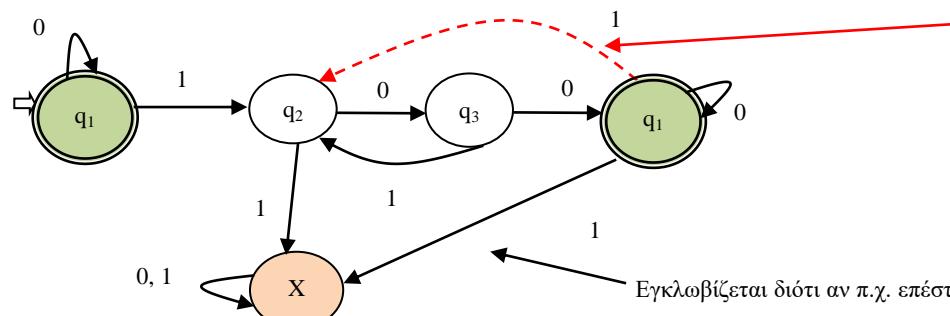
Το διάγραμμα καταστάσεων-μεταβάσεων για το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) είναι:



(b)

1.

Το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (DFA) είναι:



Εσφαλμένη μετάβαση διότι μπορεί να υπάρχει ενδιάμεσος 1 που ακολουθείται από περισσότερα 0 αντί για ένα μόνο 0

Εγκλωβίζεται διότι αν π.χ. επέστρεφε στην q2 τότε θα δεχόταν συμβολοσειρές οι οποίες μετά τον αρχικό 1 θα είχαν περισσότερα από δύο μηδενικά που αυτό εξορισμού απαγορεύεται από την εκφώνηση

2.

Η κανονική έκφραση είναι: $0^* | 0^*(10)^*100^+$

2.19 Θέμα 1 Σεπτέμβριος 2017

1.

a) Να γραφεί κανονική έκφραση που να περιέχει τουλάχιστον έναν 1 (άσο) και αν περιέχει 0 (μηδέν) τότε δεξιά και αριστερά του υπάρχουν ένας ή περισσότεροι 1 (άσσοι).

β) Να κατασκευάσετε το πεπερασμένο αυτόματο για τη γλώσσα αυτή.

2.

a) Δίνεται η έκφραση $1? 0(10)^* | 0? 1(01)^*$. Δώστε παραδείγματα και εξηγείστε τα χαρακτηριστικά της κανονικής έκφρασης

β) Να κατασκευάσετε το πεπερασμένο αυτόματο για τη γλώσσα αυτή

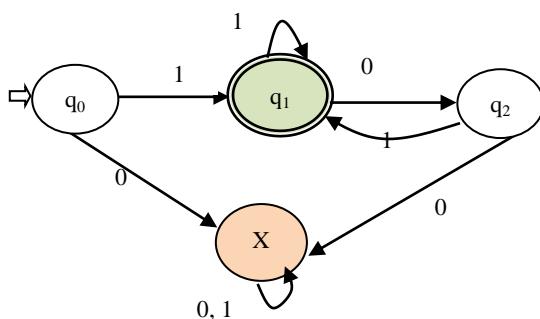
Απάντηση

1.

a) Παραδείγματα συμβολοσειρών 1, 11, 111, 10111, 11101101101

Η κανονική έκφραση είναι: $1^+(01^+)^*$

β) Το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) για τη γλώσσα αυτή είναι το ακόλουθο:



2.

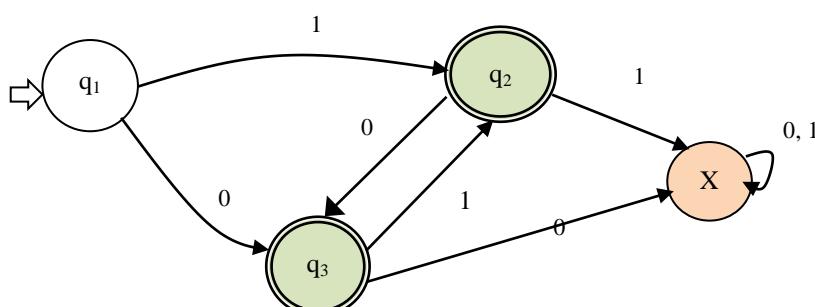
a)

1? 0(10)* | 0? 1(01)*. Παραδείγματα συμβολοσειρών είναι π.χ. 1, 10, 1010, 101010, 0, 01, 0101, 010101 κ.λ.π.

Η κανονική έκφραση περιγράφει μη κενές συμβολοσειρές με μήκος τουλάχιστον ένα χαρακτήρα που **παρουσιάζουν διαδοχική εναλλαγή χαρακτήρων εφόσον έχουν μήκος πάνω από 1 χαρακτήρα**

β) Το ντετερμινιστικό πεπερασμένο αυτόματο για τη γλώσσα αυτή είναι το ακόλουθο:

(b)



Παρατηρήσεις

- $1?$ σημαίνει ένας άσσος ή κανένας άσσος
- $0?$ σημαίνει ένα μηδενικό ή κανένα μηδενικό
- Το $?$ αφορά αποκλειστικά το χαρακτήρα μπροστά από τον οποίο βρίσκεται

2.20 Θέμα 1 Φεβρουάριος 2019

(α)

- Να παρουσιάσετε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3. Π.χ. οι αριθμοί 101, 110000, 011010111
- Γράψτε μια κανονική έκφραση που να αντιστοιχεί το παραπάνω αυτόματο

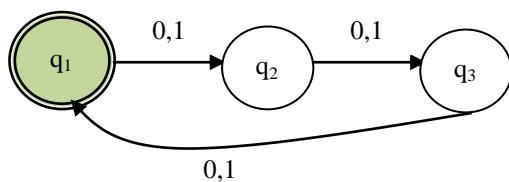
(β) Δίνεται η κανονική έκφραση $1^* | 11^+(011)^+$ με αλφάριθμο 0, 1

- Περιγράψτε τις λεξικές μονάδες που αναγνωρίζει η κανονική έκφραση
- Να παρουσιάσετε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει την παραπάνω κανονική έκφραση

Απάντηση

(α) Το DFA που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3 είναι το ακόλουθο:

1.

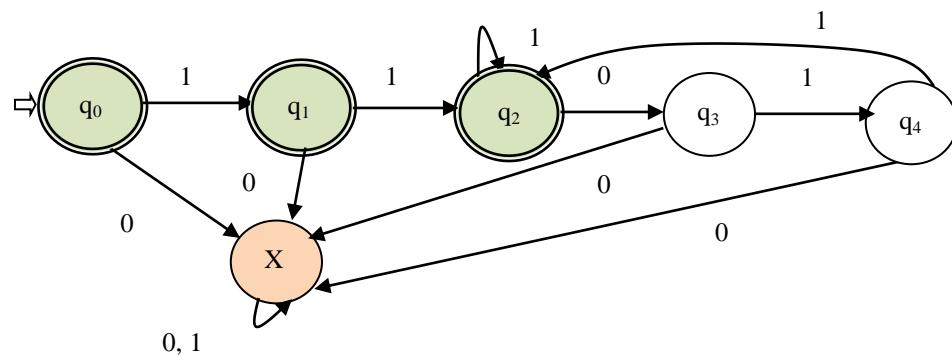


2. Η κανονική έκφραση που αναγνωρίζει όλους τους δυαδικούς αριθμούς με μήκος πολλαπλάσιο του 3 είναι η ακόλουθη:

$$((0|1)(0|1)(0|1))^*$$

(β)

- Η κανονική έκφραση αναγνωρίζει συμβολοσειρές που αποτελούνται μόνο από 1 συμπεριλαμβανομένης της κενής συμβολοσειράς στις οποίες αν εμφανίζεται 0 τότε αριστερά και υπάρχουν τουλάχιστον δύο 1
- Το πεπερασμένο αυτόματο (DFA) που αναγνωρίζει την παραπάνω κανονική έκφραση είναι:

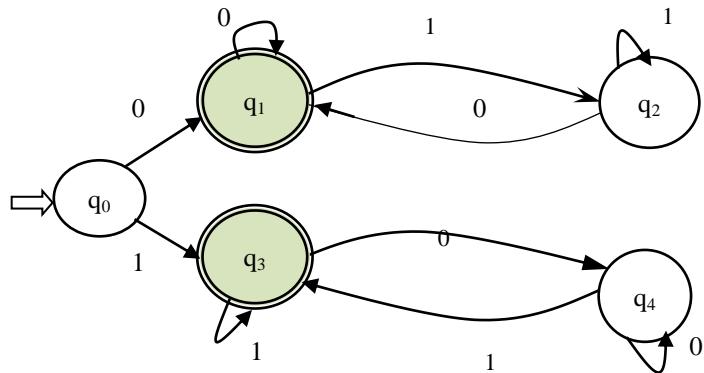


2.21 Θέμα 2 Ιούνιος 2019-Ομάδα Α

(α) Να παρουσιάσετε το διάγραμμα καταστάσεων – μεταβάσεων για το ντετερμινιστικό πεπερασμένο αυτόματο που αναγνωρίζει τους δυναδικούς αριθμούς οι οποίοι έχουν ένα τουλάχιστον ψηφίο και αρχίζουν και τελειώνουν με το ίδιο ψηφίο. Παραδείγματα αποδεκτών λέξεων: 0, 101, 01010110

(β) Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο.

Απάντηση



2. Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι: **1((0|1)* 1)? | 0((0|1)* 0)?**

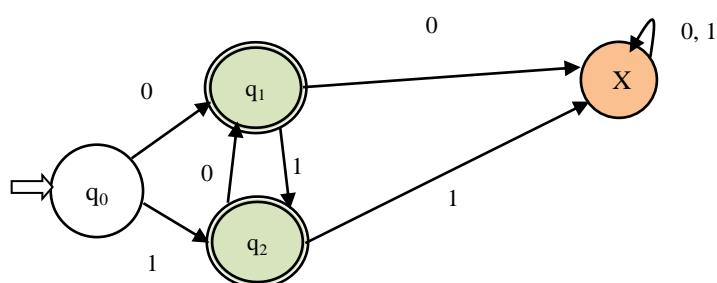
2.22 Θέμα 2 Ιούνιος 2019-Ομάδα Β

(α) Να παρουσιάσετε το διάγραμμα καταστάσεων – μεταβάσεων για το ντετερμινιστικό πεπερασμένο αυτόματο που αναγνωρίζει τους δυναδικούς αριθμούς οι οποίοι έχουν ένα τουλάχιστον ψηφίο και δεν έχουν δύο ίδια ψηφία συνεχόμενα. Παραδείγματα αποδεκτών λέξεων: 0, 101, 01010110

(β) Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο.

Απάντηση

(α) Το διάγραμμα καταστάσεων-μεταβάσεων (DFA) είναι το ακόλουθο:



(β). Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι: **0(10)*1? | 1(01)*0?**

2.23 Θέμα 1 Σεπτέμβριος 2018

(a) 1. Δίνεται η κανονική έκφραση (regular expression):

$$(1|01|001)^*(0|00)? \text{ με αλφάβητο } 0, 1$$

Περιγράψτε τις λεξικές μονάδες που παράγει η κανονική έκφραση. Δώσε παραδείγματα.

2. Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το **ντετερμινιστικό πεπερασμένο αυτόματο** που αναγνωρίζει την παραπάνω κανονική έκφραση

(b)

- Να παρουσιάσετε **διάγραμμα καταστάσεων-μεταβάσεων** για το **ντετερμινιστικό πεπερασμένο αυτόματο** που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι περιέχουν **ένα αριθμό από 1** (**υπάρχει τουλάχιστον ένα 1**) οι οποίοι είναι όλοι συνεχόμενοι και αν **υπάρχουν 0 αυτά είναι όλα μαζί πριν τα 1** και ο αριθμός τους είναι ζυγός (μπορεί να μην υπάρχουν 0)
- Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο

Απάντηση

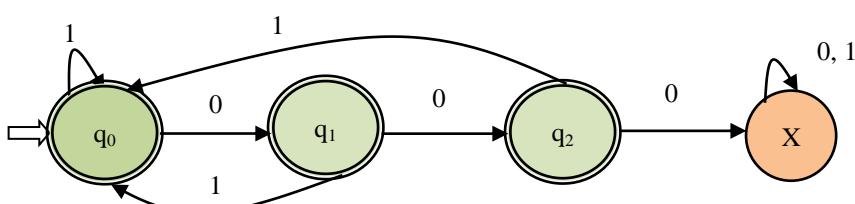
(a)

- ϵ
- 1
- 10
- 100
- 010
- 0100
- 0010
- 00100
- 00
- 0
- 101001
- 1010010
- 10100100
- 0010010010100
- 1111
- 1110

Οι λεξικές μονάδες που παράγει η κανονική έκφραση έχουν το πολύ δύο διαδοχικά μηδενικά και σε αυτές συμπεριλαμβάνεται η κενή συμβολοσειρά

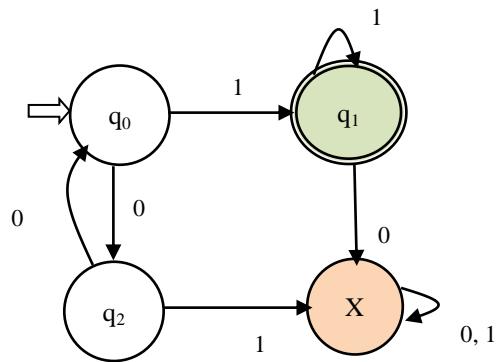
Παραδείγματα λεξικών μονάδων: $\epsilon, 1, 01, 0101, 101, 1001011, 10, 100, 010, 0100, 0010, 00100, 0010010100$

2. Το διάγραμμα καταστάσεων-μεταβάσεων (DFA) είναι το ακόλουθο:



(b)

1. Το διάγραμμα καταστάσεων-μεταβάσεων (DFA) είναι το ακόλουθο:



2. Η κανονική έκφραση είναι $(00)^*1^+$ ή εναλλακτικά $1^+ | (00)^+1^+$

2.24 Θέμα 1 Φεβρουάριος 2020

(a) Δίνεται η κανονική έκφραση (regular expression) $a(bb)^*ba$ με αλφάβητο a, b

- Περιγράψτε τις λεξικές μονάδες που παράγει η κανονική έκφραση. Δώστε παραδείγματα
- Παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο (Finite State Automaton) που αναγνωρίζει την παραπάνω κανονική έκφραση

(b)

- Παρουσιάστε διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο που αναγνωρίζει τις λέξεις με αλφάβητο a και b οι οποίες αρχίζουν με δύο a και τελειώνουν με δύο b. Ενδιάμεσα μπορεί να υπάρχει οποιοσδήποτε αριθμός από a και /ή b (ακόμα και μηδενικός)
- Γράψτε μια κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο

Απάντηση

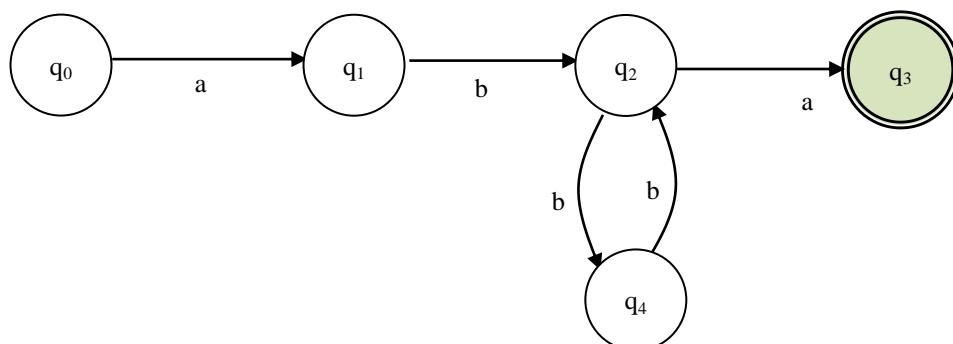
(a)

- aba
- abbba
- abbbbbba
- abbbbbbbba κ.λ.π.

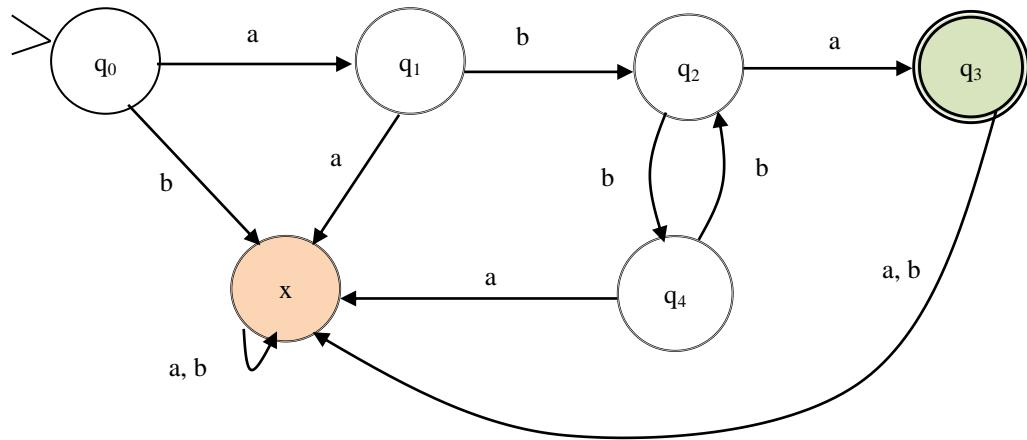
Οι λεξικές μονάδες που παράγει η κανονική έκφραση έχουν τα ακόλουθα χαρακτηριστικά:

- Αρχίζουν με a και τελειώνουν με a
- Έχουν περιττό πλήθος b
- Έχουν περιττό μήκος
- Η μικρότερη συμβολοσειρά έχει μήκος 3 ή ενναλακτικά οι συμβολοσειρές έχουν μήκους τουλάχιστον 3 χαρακτήρες
- Οι χαρακτήρες b είναι διαδοχικοί και δεν παρεμβάλλονται ανάμεσα τους
- Έχουν ακριβώς δύο χαρακτήρες a

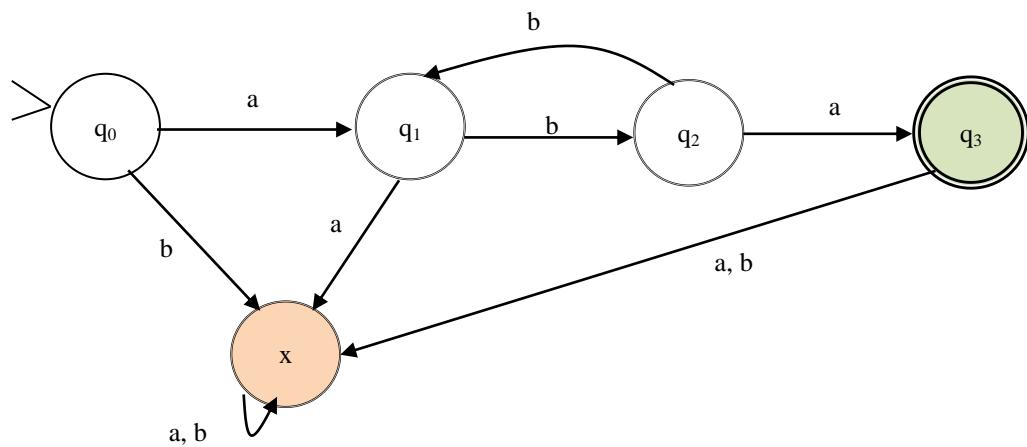
To NFA για τη γλώσσα είναι το ακόλουθο:



To DFA για τη γλώσσα είναι το ακόλουθο:

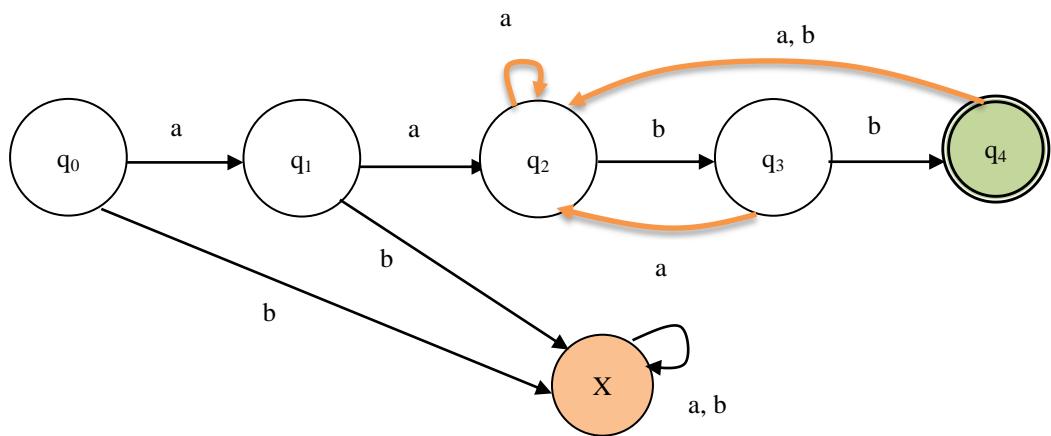


Εναλλακτικά το ζητούμενο DFA είναι το ακόλουθο:



(b)

Το διάγραμμα καταστάσεων-μεταβάσεων για το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) είναι το ακόλουθο:



Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι: **aa (a | b)* bb**

2.25 Θέμα 3 Σεπτέμβριος 2012

Δίνεται ο πίνακας καταστάσεων-μεταβάσεων ενός πεπερασμένου αυτομάτου που αναγνωρίζει ένα υποσύνολο λέξεων που αποτελούνται από τα γράμματα a, b, c με αρχική κατάσταση την 1.

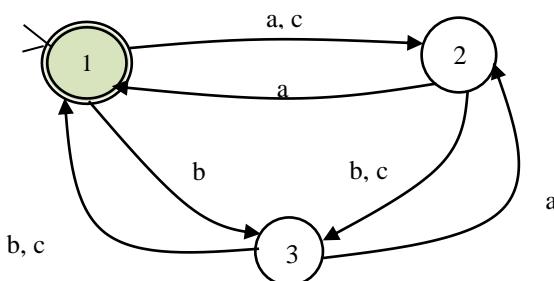
Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
1	a	2	OXI
1	b	3	OXI
1	c	2	OXI
2	a	1	ΝΑΙ
2	b	3	OXI
2	c	3	OXI
3	a	2	OXI
3	b	1	ΝΑΙ
3	c	1	ΝΑΙ

- a) Να παρουσιάσετε το διάγραμμα καταστάσεων-μεταβάσεων για το πεπερασμένο αυτόματο
- b) Ποια ή ποιες από τις λέξεις caba, baaca, acccabb αναγνωρίζει το αυτόματο;
- c) Περιγράψτε όσες ιδιότητες των λέξεων που αναγνωρίζει το αυτόματο μπορείτε να διακρίνετε

Απάντηση

(a)

Κάθε γραμμή του πίνακα καταστάσεων-μεταβάσεων ενός πεπερασμένου αυτομάτου αντιστοιχεί και σε μια μετάβαση στο DFA που σχεδιάζουμε



(b)

- Η λέξη **caba** δεν αναγνωρίζεται από το αυτόματο διότι όταν τη διαβάσει πλήρως τερματίζει στην κατάσταση 2 που δεν είναι τελική
- Η λέξη **baaca** αναγνωρίζεται από το αυτόματο διότι όταν τη διαβάσει πλήρως τερματίζει στην κατάσταση 1 που είναι τελική
- Η λέξη **acccabb** αναγνωρίζεται από το αυτόματο διότι όταν τη διαβάσει πλήρως τερματίζει στην κατάσταση 1 που είναι τελική

(c) Η κανονική έκφραση που περιγράφει τη γλώσσα που παράγεται από το πεπερασμένο αυτόματο είναι η εξής:

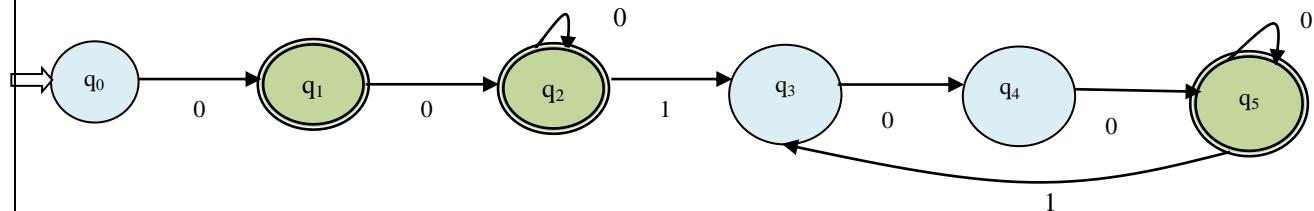
$$((a|c)(b|c)(b|c))^* | (b(b|c))^* | (baa)^* | ((a|c)a)^* | (ba(b|c)(b|c))^*$$

2.26 Θέμα 1 Ιούνιος 2021

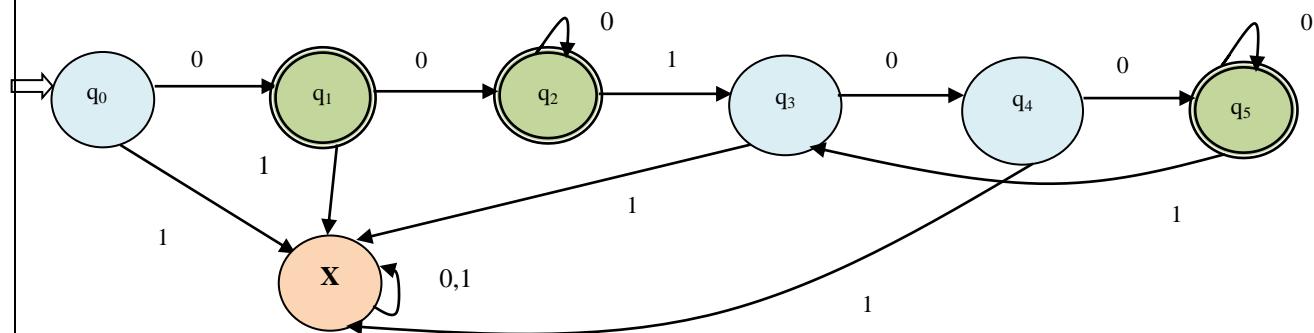
- a. Να παρουσιάστε τον Πίνακα Καταστάσεων-Μεταβάσεων για το **πεπερασμένο αυτόματο** του διαγράμματος καταστάσεων-μεταβάσεων που αναγνωρίζει τους δυαδικούς αριθμούς οι οποίοι: Περιέχουν ένα τουλάχιστον 0 ενώ τα 1 (αν υπάρχουν) εμφανίζονται ένα τη φορά και έχουν δεξιά και αριστερά τους δύο ή περισσότερα 0. Παραδείγματα 00, 0000, 001000100001000
- b. Δείξτε σε Πίνακα τη λειτουργία και το αποτέλεσμα του αυτόματου για την είσοδο 001001
- c. Παρουσιάστε την κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο.

Απάντηση

- a. Το NFA που αναγνωρίζει τη γλώσσα είναι το ακόλουθο:



- To DFA που αναγνωρίζει τη γλώσσα είναι το ακόλουθο:



Από το DFA κατασκευάζουμε τον **πίνακα Καταστάσεων-Μεταβάσεων** ο οποίος είναι ο ακόλουθος: (κάθε μετάβαση στο DFA είναι και μια γραμμή του πίνακα Καταστάσεων-Μεταβάσεων)

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
q ₀	0	q ₁	ΝΑΙ
q ₀	1	X	ΟΧΙ
q ₁	0	q ₂	ΝΑΙ
q ₁	1	X	ΟΧΙ
q ₂	0	q ₂	ΝΑΙ
q ₂	1	q ₃	ΟΧΙ
q ₃	0	q ₄	ΟΧΙ
q ₃	1	X	ΟΧΙ
q ₄	0	q ₅	ΝΑΙ
q ₄	1	X	ΟΧΙ
q ₅	0	q ₅	ΝΑΙ
q ₅	1	q ₃	ΟΧΙ

b. Λειτουργία και αποτέλεσμα του αυτόματου για την είσοδο 001001

Τρέχουσα Κατάσταση	Είσοδος	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
q ₀	001001	0	q ₁	
q ₁	001001	0	q ₂	
q ₂	001001	1	q ₃	
q ₃	001001	0	q ₄	
q ₄	001001	0	q ₅	
q ₅	001001	1	q ₃	ΟΧΙ

c.

- Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι $00^+ (100^+)^* | 0$
- Εναλλακτικά η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι $0^+ | (00^+ 100^+)^+$

2.27 Φροντιστήριο 2023 – Άσκηση 4

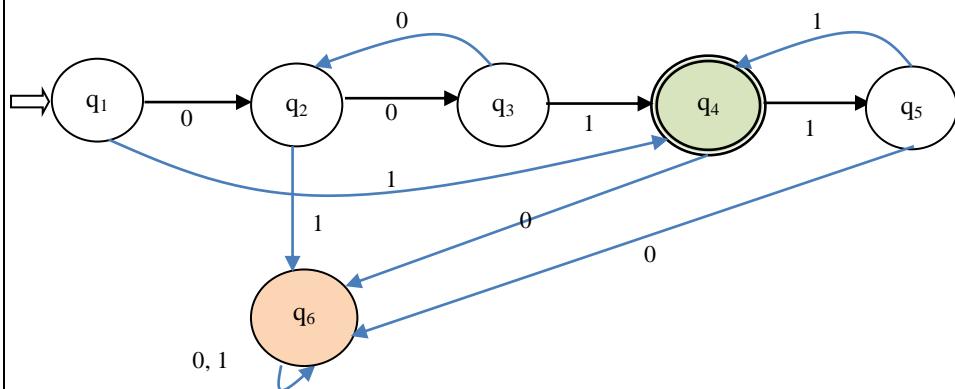
a) Να παρουσιάσετε το διάγραμμα καταστάσεων – μεταβάσεων για το **ντετερμινιστικό πεπερασμένο αυτόματο** που αναγνωρίζει όλους τους δυαδικούς αριθμούς οι οποίοι έχουν **μονό αριθμό από 1** (υπάρχει τουλάχιστον ένα 1) τα οποία είναι όλα συνεχόμενα, και αν υπάρχουν **0** αυτά είναι όλα μαζί πριν τα **1** και ο αριθμός τους είναι ζυγός (μπορεί να μην υπάρχει 0).

b) Γράψετε μια **κανονική έκφραση** που αντιστοιχεί στο παραπάνω αυτόματο.

c) Παρουσιάστε τον **Πίνακα Καταστάσεων – Μεταβάσεων** που αντιστοιχεί στο διάγραμμα καταστάσεων – μεταβάσεων του ερώτηματος (a)

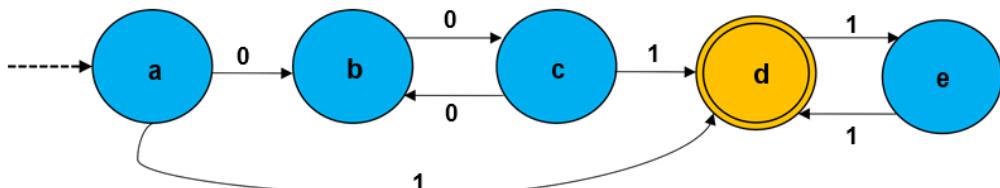
Απάντηση

a) Το **ντετερμινιστικό πεπερασμένο αυτόματο (DFA)** που αναγνωρίζει τη γλώσσα είναι:



Παρατήρηση

Το ΜΗ **ντετερμινιστικό πεπερασμένο αυτόματο (NFA)** που αναγνωρίζει τη γλώσσα είναι:



b)

Η κανονική έκφραση που αντιστοιχεί στο παραπάνω αυτόματο είναι: **(00)*·1·(11)*** ή **(00)*·(11)*·1**

c) Ο **Πίνακας Καταστάσεων – Μεταβάσεων** που αντιστοιχεί στο **DFA** είναι:

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
q ₁	0	q ₂	ΟΧΙ
q ₁	1	q ₄	ΝΑΙ
q ₂	0	q ₃	ΟΧΙ
q ₂	1	q ₆	ΝΑΙ
q ₃	0	q ₂	ΟΧΙ
q ₃	1	q ₄	ΝΑΙ
q ₄	0	q ₆	ΟΧΙ

q_4	1	q_5	OXI
q_5	0	q_6	OXI
q_5	1	q_4	NAI
q_6	0	q_6	OXI
q_6	1	q_6	OXI

Παρατήρηση

Ο Πίνακας Καταστάσεων – Μεταβάσεων που αντιστοιχεί στο NFA είναι:

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
a	0	b	OXI
a	1	d	NAI
b	0	c	OXI
c	0	b	OXI
c	1	d	NAI
d	1	e	OXI
e	1	d	NAI

3 ΘΕΜΑΤΑ ΚΑΙ ΑΣΚΗΣΕΙΣ ΜΕ ΓΡΑΜΜΑΤΙΚΕΣ ΧΩΡΙΣ ΣΥΦΜΡΑΖΟΜΕΝΑ

3.1 Ορισμός Γραμματικής Χωρίς Συμφραζόμενα

Μια Γραμματική Χωρίς Συμφραζόμενα είναι μια τετράδα της μορφής $G = \{V, \Sigma, R, S\}$

- $V =$ Σύνολο ΜΗ Τερματικών συμβόλων δηλ. συμβόλων από τα οποία αρχίζουν κανόνες
- $\Sigma =$ Σύνολο Τερματικών συμβόλων δηλ. συμβόλων από τα οποία δεν αρχίζουν κανόνες
- $R =$ οι κανόνες της γραμματικής
- S το αρχικό σύμβολο της γραμματικής από το οποίο αρχίζουν οι κανόνες

3.1.1 Παράδειγμα 1 με Γραμματική Χωρίς Συμφραζόμενα

Δίνεται η Γραμματική Χωρίς Συμφραζόμενα με 4 κανόνες $E \rightarrow E+E | E^*E | (E) | id$

- **Τα χαρακτηριστικά μεγέθη της γραμματικής είναι τα ακόλουθα:**

$V = \{E\}$ Μη Τερματικό Σύμβολο

$\Sigma = \{+, *, (,), id\}$ Τερματικά Σύμβολα

$R = \{E \rightarrow E+E, E \rightarrow E^*E, E \rightarrow (E), E \rightarrow id\}$ Κανόνες

$S = E$ Αρχικό σύμβολο

- **Τι συμβολοσειρές-παράγει αυτή η γραμματική;**

Κάνουμε παραγωγές δηλ. εφαρμόζουμε ακριβώς τους κανόνες που δίνονται με τυχαία σειρά και με διαφορετική σειρά και προσπαθούμε να κατασκευάσουμε πολλές συμβολοσειρές (τουλάχιστον 5-6) και από αυτές να συμπεράνουμε τα χαρακτηριστικά τους

$E \Rightarrow id$

$E \Rightarrow E+E \Rightarrow id+id$

$E \Rightarrow E^*E \Rightarrow id^*id$

$E \Rightarrow E^*E \Rightarrow (E)^*(E) \Rightarrow (E+E)^*(E+E) \Rightarrow (id+id)+(id^*id)$

$E \Rightarrow E^*E \Rightarrow E+E \Rightarrow E+E^*(E) \Rightarrow E+E^*(E+E) \Rightarrow id+id*(id+id)$

$E \Rightarrow E^*E \Rightarrow E^*E \Rightarrow E^*E^*E \Rightarrow (E)^*E+E^*E^*E \Rightarrow (id)^*id+id^*id*id^*id$

Συμπέρασμα: Η Γραμματική κατασκευάζει αριθμητικές παραστάσεις με τους τελεστές $+, ^*, ()$ και id

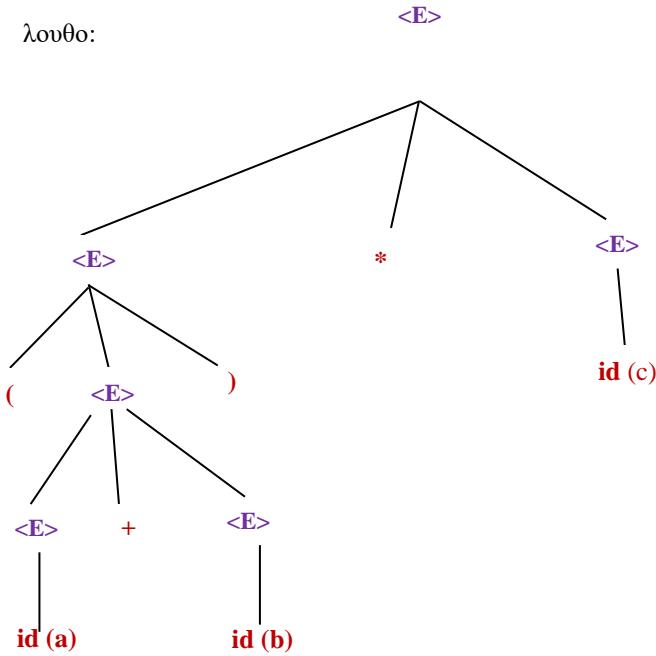
- **Πως γράφεται η γραμματική σε συμβολισμό BNF;**

Ο συμβολισμός BNF αλλάζει απλά τη μορφή των κανόνων (και όχι τη λειτουργία τους) και βάζει τα ΜΗ τερματικά σύμβολα από τα οποία ξεκινάνε οι κανόνες μέσα σε tags $\langle \rangle$ ενώ το σύμβολο \rightarrow γράφεται ως διπλή άνω και κάτω τελεία και ίσον δηλ. $::=$. Τα τερματικά σύμβολα παραμένουν στη γραφή τους ως έχουν

Η προηγούμενη γραμματική σε συμβολισμό BNF γράφεται ως: $\langle E \rangle ::= \langle E \rangle + \langle E \rangle | \langle E \rangle ^* \langle E \rangle | (\langle E \rangle) | id$

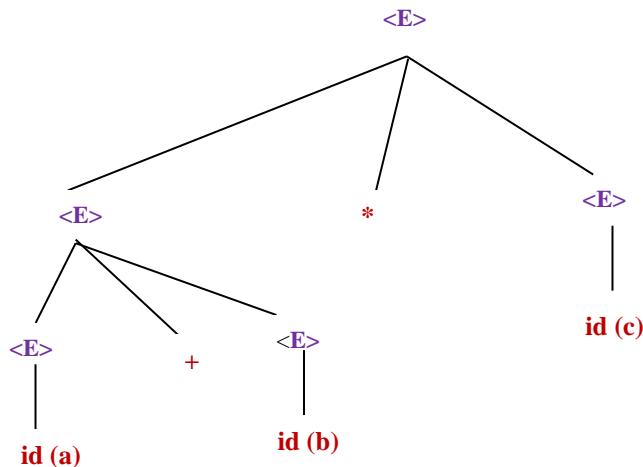
- Να κατασκευάσετε το συντακτικό δέντρο (parse tree) για τη συμβολοσειρά $(a+b)^*c$

Ένα συντακτικό δέντρο (parse tree) είναι μια γραφική αναπαράσταση μιας παραγωγής. Στα φύλλα του parse tree τοποθετείται η συμβολοσειρά που παράγεται και το parse tree 'διαβάζεται' από πάνω προς τα κάτω και από αριστερά προς τα δεξιά. Η παραγωγή για τη συμβολοσειρά $E \Rightarrow E^*E \Rightarrow (E)^*E \Rightarrow (E+E)^*E \Rightarrow (id+id)^*id$. To parse tree για την παραγωγή αυτή είναι το ακόλουθο:

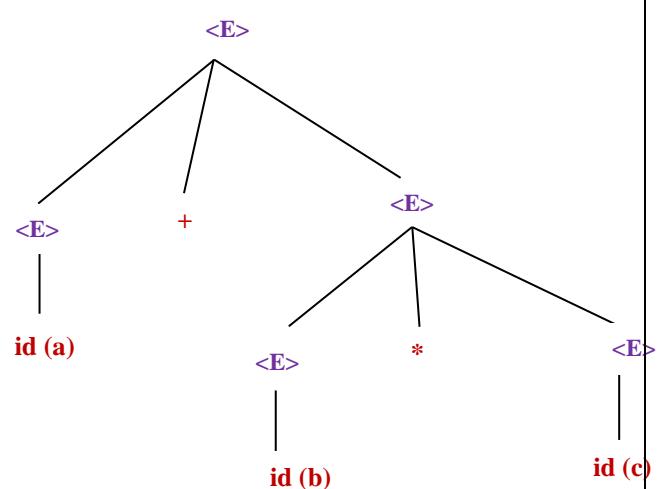


- Να εξετάσετε αν η γραμματική είναι διφορούμενη

Μια γραμματική ονομάζεται διφορούμενη αν μπορούμε να κατασκευάσουμε 2 διαφορετικά parse tree για την ίδια συμβολοσειρά. Για παράδειγμα η συμβολοσειρά $a+b^*c$ που παράγεται από τη γραμματική αυτή έχει δύο διαφορετικά parse trees τα οποία είναι τα ακόλουθα:



$E \Rightarrow E^*E \Rightarrow \underline{E+E^*E} \Rightarrow id(a)+id(b)^*id(c)$



$E \Rightarrow E+E \Rightarrow E+E^*E \Rightarrow id(a)+id(b)^*id(c)$

Άρα η γραμματική αυτή είναι διφορούμενη

- Να εξετάσετε αν η γραμματική αυτή είναι κανονική

Ορισμός: Μια Γραμματική είναι κανονική αν σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο

Για παράδειγμα η γραμματική που δίνεται ΔΕΝ είναι κανονική διότι π.χ. στον κανόνα $E \rightarrow E+E$ το δεξί μέλος αρχίζει με MH τερματικό σύμβολο

3.1.2 Παράδειγμα 2 με Γραμματική Χωρίς Συμφραζόμενα

Δίνονται οι ακόλουθες τρεις γραμματικές Χωρίς Συμφραζόμενα σε συμβολισμό BNF:

- 1^η Γραμματική $<A> ::= 0 <A> | 1 <A> | 0$
- 2^η Γραμματική $<\Sigma> ::= <\Sigma> <\Sigma> | 1 | 0$
- 3^η Γραμματική $<\Lambda> ::= 0 <\Lambda> 0 | 1 <\Lambda> 1 | 0 | 1$

Τι είδους συμβολοσειρές παράγει η κάθε γραμματική;

- Κάνουμε πρώτα παραγωγές

$A \Rightarrow 0$

$A \Rightarrow 0A \Rightarrow 00$

$A \Rightarrow 1A \Rightarrow 10$

$A \Rightarrow 1A \Rightarrow 11A \Rightarrow 110$

$A \Rightarrow 0A \Rightarrow 00A \Rightarrow 001A \Rightarrow 0010$

$A \Rightarrow 1A \Rightarrow 10A \Rightarrow 101A \Rightarrow 1010A \Rightarrow 10100$

Η 1^η Γραμματική παράγει συμβολοσειρές που τελειώνουν σε 0

- Κάνουμε πρώτα παραγωγές

$\Sigma \Rightarrow 1$

$\Sigma \Rightarrow 0$

$\Sigma \Rightarrow \Sigma \Sigma \Rightarrow 01$

$\Sigma \Rightarrow \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Rightarrow 111$

$\Sigma \Rightarrow \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Rightarrow 111$

$\Sigma \Rightarrow \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Sigma \Rightarrow 1110$

$\Sigma \Rightarrow \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Rightarrow \Sigma \Sigma \Sigma \Sigma \Rightarrow 01100$

Η 2η Γραμματική παράγει όλες τις μη κενές συμβολοσειρές από 0 και 1

- Κάνουμε πρώτα παραγωγές

$\Lambda \Rightarrow 0$

$\Lambda \Rightarrow 1$

$\Lambda \Rightarrow 0\Lambda 0 \Rightarrow 010$

$\Lambda \Rightarrow 1\Lambda 1 \Rightarrow 10\Lambda 01 \Rightarrow 10001$

$\Lambda \Rightarrow 1\Lambda 1 \Rightarrow 10\Lambda 01 \Rightarrow 101\Lambda 101 \Rightarrow 1010101$

$\Lambda \Rightarrow 0\Lambda 0 \Rightarrow 00\Lambda 00 \Rightarrow 001\Lambda 100 \Rightarrow 0010\Lambda 0100 \Rightarrow 001010100$

Η 3^η Γραμματική παράγει παλινδρομικές συμβολοσειρές περιττού μήκους

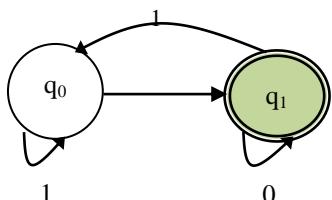
- Ποια(ες) Γραμματική (ες) είναι διφορούμενη (ες);

- Η 1^η Γραμματική ΔΕΝ είναι διφορούμενη

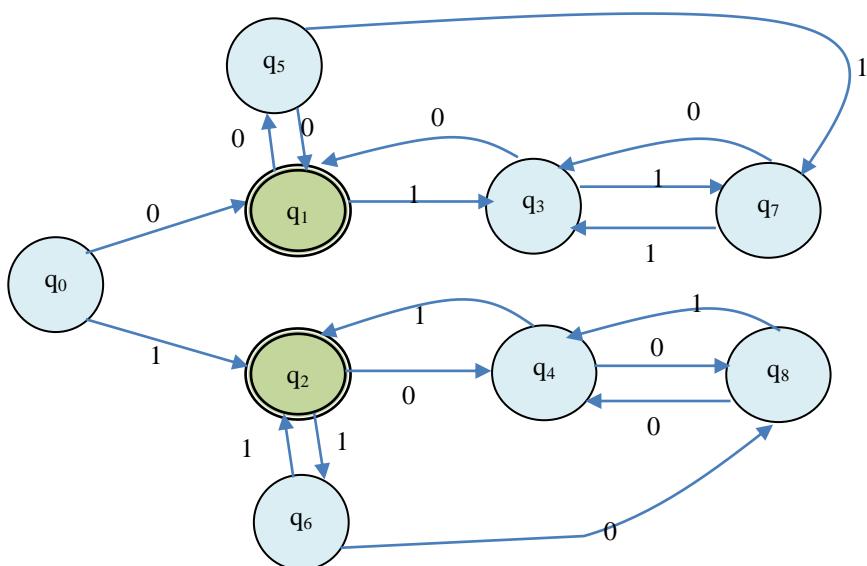
- Η 2^η Γραμματική είναι διφορούμενη διότι π.χ. η συμβολοσειρά 101 έχει 2 διαφορετικές παραγωγές:
 - $\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \underline{\Sigma\Sigma} \Rightarrow 101$ και $\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \Sigma\underline{\Sigma} \Rightarrow 101$
- Η 3^η Γραμματική ΔΕΝ είναι διφορούμενη
- **Ποια(ες) Γραμματική (ες) είναι κανονική (ες);**
- **Ορισμός: Μια Γραμματική είναι κανονική αν σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο**
 - Η 1^η Γραμματική είναι κανονική διότι σε κάθε κανόνα $\langle A \rangle ::= \underline{0} \langle A \rangle \mid \underline{1} \langle A \rangle \mid \underline{0}$ το δεξιά μέλος του αρχίζει με τερματικό σύμβολο
 - Η 2^η Γραμματική ΔΕΝ είναι κανονική διότι στον κανόνα $\langle \Sigma \rangle ::= \langle \Sigma \rangle \langle \Sigma \rangle \mid \underline{0} \mid \underline{1}$ το δεξιά μέλος του ΔΕΝ αρχίζει με τερματικό σύμβολο
 - Η 3^η Γραμματική είναι κανονική διότι σε κάθε κανόνα $\langle \Lambda \rangle ::= \underline{0} \langle \Lambda \rangle \mid \underline{1} \langle \Lambda \rangle \mid \underline{0} \mid \underline{1}$ το δεξιά μέλος του αρχίζει με τερματικό σύμβολο

- **Για τη γραμματική που είναι κανονική να κατασκευάσετε DFA**

Για την 1^η γραμματική που είναι κανονική μπορούμε να κατασκευάσουμε DFA που αναγνωρίζει τις συμβολοσειρές της γλώσσας που τελειώνουν σε 0.



Για την 3^η γραμματική που είναι κανονική μπορούμε να κατασκευάσουμε DFA που αναγνωρίζει τις συμβολοσειρές της γλώσσας που είναι παλινδρομικές περιττού μήκους.



3.2 Κατασκευή Γραμματικών Χωρίς Συμφραζόμενα

Για να κατασκευάσουμε μια Γραμματική Χωρίς Συμφραζόμενα (Γ.Χ.Σ.) δεν υπάρχει συγκεκριμένος τρόπος ή μεθοδολογία. Βασι-
ζόμαστε μόνο σε εμπειρικούς κανόνες.

Παραδείγματα Κατασκευής Γραμματικών Χωρίς Συμφραζόμενα

Να κατασκευαστούν Γραμματικές Χωρίς Συμφραζόμενα (Γ.Χ.Σ.) για τις ακόλουθες γλώσσες:

1) $L = \{w \in \{a, b\}^* \mid \eta w \text{ έχει } n \text{ "a" ακολουθούμενα από } n \text{ "b" με } n \geq 0\}$

Απάντηση

$S \rightarrow aSb \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon, S \Rightarrow aSb \Rightarrow ab, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα 'a' προηγούνται των 'b' 2) τα 'a' είναι ίσου πλήθους με τα 'b' και 3) επιτρέπεται η κενή συμβολοσειρά αφού το $n \geq 0$. Το ε συμβολίζει τη κενή συμβολοσειρά.

2) $L = \{w \in \{a, b\}^* \mid \eta w \text{ έχει } n \text{ "a" ακολουθούμενα από } n \text{ "b" με } n > 0\}$

Απάντηση

$S \rightarrow aSb \mid ab$

Παραδείγματα

$S \Rightarrow ab, S \Rightarrow aSb \Rightarrow aabb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaabbbb$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα 'a' προηγούνται των 'b' 2) τα 'a' είναι ίσου πλήθους με τα 'b' και 3) δεν επιτρέπεται η κενή συμβολοσειρά αφού το $n > 0$ και η μικρότερη συμβολοσειρά είναι η 'ab'

3) $L = \{w \in \{a, b\}^* \mid \eta w \text{ έχει } "a" \text{ ακολουθούμενα από } "b" \text{ και το πλήθος των } "a" \text{ είναι μικρότερο ή ίσο από το πλήθος των } "b" \text{ και μήκος συμβολοσειράς } > 0\}$

Απάντηση

$S \rightarrow aSb \mid Sb \mid ab \mid b$

Παραδείγματα

$S \Rightarrow ab, S \Rightarrow b, S \Rightarrow aSb \Rightarrow abb, S \Rightarrow aSb \Rightarrow aabb, S \Rightarrow Sb \Rightarrow Sbb \Rightarrow abbb, S \Rightarrow Sb \Rightarrow Sbb \Rightarrow bbb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbbbb$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα 'a' προηγούνται των 'b' 2) τα 'a' είναι ίσου πλήθους με τα 'b' από τον κανόνα aSb 3) τα 'b' είναι περισσότερα από τα 'a' από τον κανόνα Sb και 4) δεν επιτρέπεται η κενή συμβολοσειρά αφού το $n > 0$ και η μικρότερη συμβολοσειρά είναι είτε η 'ab' είτε η 'b'

4) $L = \{w \in \{a, b\}^* \mid \eta w \text{ έχει } a \text{ ακολουθούμενα από } b \text{ και το πλήθος των } a \text{ είναι μικρότερο ή ίσο με το πλήθος των } b \text{ και μήκος συμβολοσειράς } \geq 0\}$

Απάντηση

$S \rightarrow aSb \mid Sb \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon, S \Rightarrow aSb \Rightarrow ab, S \Rightarrow Sb \Rightarrow b, S \Rightarrow aSb \Rightarrow aSbb \Rightarrow abb, S \Rightarrow aSb \Rightarrow aSbb \Rightarrow aaSbbb \Rightarrow aabbb$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα 'a' προηγούνται των 'b' 2) τα 'a' είναι ίσου πλήθους με τα 'b' από τον κανόνα aSb 3) τα 'b' είναι περισσότερα από τα 'a' από τον κανόνα Sb και 4) επιτρέπεται η κενή συμβολοσειρά αφού το $n \geq 0$

5) $L = \{w \in \{a, b\}^* \mid \text{τα "a" προηγούνται των "b"}\}$

Απάντηση

$S \rightarrow aS \mid Sb \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon, S \Rightarrow aS \Rightarrow a, S \Rightarrow Sb \Rightarrow b, S \Rightarrow aS \Rightarrow aSb \Rightarrow ab, S \Rightarrow aS \Rightarrow aSb \Rightarrow aSbb \Rightarrow abb$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα 'a' προηγούνται των 'b' από τον κανόνα aS 2) τα 'b' έπονται των 'a' από τον κανόνα aS και 3) επιτρέπεται η κενή συμβολοσειρά αφού δεν αναφέρεται περιορισμός στην εκφώνηση

6) $L = \{w \in \{0, 1\}^* \mid \text{το μήκος της } w \text{ είναι περιττός με μεσαίο σύμβολο το } 0\}$

Απάντηση

$S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 0$

Παραδείγματα

$S \Rightarrow 0, S \Rightarrow 0S0 \Rightarrow 000, S \Rightarrow 1S0 \Rightarrow 100, S \Rightarrow 1S1 \Rightarrow 11S01 \Rightarrow 110S001 \Rightarrow 1100S1001 \Rightarrow 110001001$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) περιττό μήκος από τους κανόνες $0S0 \mid 0S1 \mid 1S0 \mid 1S1$ και επίσης ο τερματισμός γίνεται σε 0 και 2) Με τον κανόνα τερματισμού $S \rightarrow 0$ το μεσαίο σύμβολο είναι πάντα το 0

7) $L = \{w \in \{0, 1\}^* \mid \text{όλες οι λέξεις που } \underline{\text{αρχίζουν και τελειώνουν με το ίδιο σύμβολο}}\}$

Απάντηση

$\langle S \rangle ::= 0 < A > 0 \mid 1 < A > 1 \mid 0 \mid 1$

$\langle A \rangle ::= 0 < A > \mid 1 < A > \mid \epsilon \quad // \text{αυτοί οι κανόνες παράγουν τα πάντα από 0 και 1 } (0|1)^*$

Παραδείγματα

$S \Rightarrow 0, S \Rightarrow 1, S \Rightarrow 0A0 \Rightarrow 01\underline{A}0 \Rightarrow 0100, S \Rightarrow 1A1 \Rightarrow 10A1 \Rightarrow 101$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική 1) αρχίζουν με 0 και τελειώνουν σε 0 και ενδιάμεσα μπορεί να έχουν οτιδήποτε από τον κανόνα $\langle S \rangle ::= 0 < A > 0$ 2) αρχίζουν με 1 και τελειώνουν σε 1 και ενδιάμεσα μπορεί να έχουν οτιδήποτε από τον κανόνα $\langle S \rangle ::= 1 < A > 1$ και 3) από τους κανόνες $\langle A \rangle ::= 0 < A > \mid 1 < A > \mid \epsilon$ βάζουμε ανάμεσα στο αρχικό και στο τελικό 0 είτε ανάμεσα στο αρχικό και στο τελικό 1 οποιοδήποτε άλλο χαρακτήρα. Προσοχή είναι λάθος να γραφούν κανόνες της μορφής $0S0$ ή $1S1$ γιατί ενδιάμεσα οι χαρακτήρες τοποθετούνται συμμετρικά και όχι τυχαία

8) $L = \{w \in \{0, 1\}^* \mid \eta \text{ } w \text{ περιέχει παλίνδρομο}\}$

Απάντηση

$\langle S \rangle ::= 0 < S > 0 \mid 1 < S > 1 \mid 0 \mid 1 \mid \epsilon \quad // \text{αυτοί οι κανόνες παράγουν παλίνδρομα οποιουδήποτε μήκους}$

Υπο-Περιπτώσεις Προηγούμενης Γραμματικής

- **Παλίνδρομα Άρτιου Μήκους** $S \rightarrow 0S0 \mid 1S1 \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon, S \Rightarrow 0S0 \Rightarrow 00, S \Rightarrow 1S1 \Rightarrow 10S01 \Rightarrow 101101, S \Rightarrow 1S1 \Rightarrow 11S11 \Rightarrow 110S011 \Rightarrow 110011$

• **Παλίνδρομα Περιττού Μήκους S→0S0 | 1S1 | 0 | 1**

Παραδείγματα

S⇒**0**, S⇒**1**, S⇒0S0⇒01S10⇒**01010**, S⇒1S1⇒11S11⇒**11011**

• **Παλίνδρομα Οποιουδήποτε Μήκους S→0S0 | 1S1 | 0 | 1 | ε**

Παραδείγματα

S⇒**ε**, S⇒**0**, S⇒**1**, S⇒1S1⇒11S11⇒110S011⇒**110011**, S⇒0S0⇒01S10⇒**01010**

9) L={ w∈{0, 1}* | οι λέξεις έχουν περισσότερα 0 από ότι 1 }

Απάντηση

S→TOT | 0

T→0T1 | 1T0 | 0T | ε

Παραδείγματα

S⇒**0**

S⇒T0T⇒**0T101T0**⇒**01010**

S⇒T0T⇒**0T01T0**⇒**00T010**⇒**00010**

S⇒T0T⇒**1T00T**⇒**1000T**⇒**1000**

S⇒T0T⇒0T0T⇒001T⇒**001**

Οι κανόνες T→0T1 | 1T0 προσθέτουν για κάθε 0 και ένα 1 άρα έτσι οι 1 και τα 0 είναι ισόποσα αλλά από τον αρχικό κανόνα S→TOT που μας οδηγεί στο T παράγεται πάντα ένα επιπλέον 0. Ο κανόνας T→0T δίνει όσα περισσότερα 0 επιθυμούμε. Ο κανόνας S→0 απαγορεύει την κενή συμβολοσειρά γιατί τα 0 πρέπει να είναι πάντα περισσότερα

10) L={ w∈{0, 1}* | η w περιέχει τουλάχιστον τρεις 1 }

Απάντηση

S→A1A1A1A //οι άσσοι δεν είναι υποχρεωτικά διαδοχικοί

A→0A |1A |ε //αντοί οι κανόνες δίνουν οτιδήποτε από 0 και 1

Παραδείγματα

S⇒A1A1A1A⇒**111**

S⇒A1A1A1A⇒**0A10A10A1**⇒**0100A100A1**⇒**01001000A1**⇒**010010001**

Ο κανόνας S→A1A1A1A δίνει τους 3 τουλάχιστον άσσους οι οποίοι δεν είναι πρέπει να είναι οπωσδήποτε διαδοχικοί καθώς δεν αναφέρεται κάτι τέτοιο στην εκφώνηση. Οι κανόνες A→0A |1A |ε δίνουν οτιδήποτε από 0 και 1.

11) Να γραφεί Γραμματική Χωρίς Συμφραζόμενα η οποία περιγράφει απλές αριθμητικές εκφράσεις με τερματικά σύμβολα τα +, -, *, /, ^, (,)

Απάντηση

E→ E A E | (E) | -E | id //το -E αντιπροσωπεύει το μοναδιαίο τελεστή προσήμουν

A→+ | - | * | / | ^ //δυναδικοί τελεστές δηλ εφαρμόζονται ανάμεσα σε 2 μεταβλητές

Παραδείγματα

$E \Rightarrow E A E \Rightarrow E A E A E \Rightarrow E A E A E A \Rightarrow id + id * id ^ id$

$E \Rightarrow (E) \Rightarrow (EAE) \Rightarrow (id+id)$

Ο κανόνας $E \rightarrow E A E$ δίνει παραστάσεις με δυαδικούς τελεστές αφού το $A \rightarrow + | - | * | / | ^$ δίνει ένα δυαδικό τελεστή. Ο κανόνας $-E$ δίνει το μοναδιαίο τελεστή προσήμου και λέγεται μοναδιαίος διότι εφαρμόζεται σε μια μεταβλητή

12) Να κατασκευάσετε γραμματική χωρίς συμφραζόμενα για τη γλώσσα $L = \{ w \in \{0,1\}^* \mid \eta w \text{ περιέχει ίσο πλήθος } 0 \text{ και } 1 \}$

Απάντηση

$S \rightarrow S0S1S \mid S1S0S \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon, S \Rightarrow S0S1S \Rightarrow 01, S \Rightarrow S0S1S \Rightarrow \underline{S0S1S0S1S0S1S0S} \Rightarrow 01010110$

Οι κανόνες $S \rightarrow S0S1S \mid S1S0S$ δίνουν σε κάθε παραγωγή ισόποσα 0 και 1 και σε οποιαδήποτε θέση.

13) Να κατασκευάσετε γραμματική χωρίς συμφραζόμενα για τις γλώσσες στο αλφάβητο $\Sigma = \{0, 1\}$:

- $L = \{w \mid \text{το μήκος της } w \text{ είναι περιττό}\}$
- $L = \{0^n 1^{2n} \mid n \geq 0\}$
- $L = \{\eta \mid w \text{ έχει άρτιο μήκος}\}$
- $L = \{\eta \mid w \text{ έχει περιττό μήκος}\}$

Απάντηση

a) $S \rightarrow 0S1 \mid 1S0 \mid 1S1 \mid 0S0 \mid 0 \mid 1$

Οι κανόνες $S \rightarrow 0S1 \mid 1S0 \mid 1S1 \mid 0S0$ δίνουν συμβολοσειρές περιττού μήκους με όλους τους συνδυασμούς 0 και 1 και τερματίζουν είτε σε 0 είτε σε 1 οπότε και πάλι διασφαλίζεται το περιττό μήκος

b) $S \rightarrow 0S11 \mid \epsilon$

Παρατηρούμε ότι όλες οι συμβολοσειρές που παράγονται από αυτή τη γραμματική έχουν 1) διάταξη στους χαρακτήρες τους διότι τα '0' προηγούνται των '1' από τον κανόνα $0S11$ και 2) οι '1' είναι διπλάσιοι των '0' αφού για κάθε '0' προστίθενται δύο '1'

c) $S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid \epsilon$

Οι κανόνες $S \rightarrow 0S1 \mid 1S0 \mid 1S1 \mid 0S0$ δίνουν συμβολοσειρές περιττού μήκους με όλους τους συνδυασμούς 0 και 1 και τερματίζουν με την κενή συμβολοσειρά οπότε προκύπτει πάντα άρτιο μήκος

d) $S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 0 \mid 1$

Οι κανόνες $S \rightarrow 0S1 \mid 1S0 \mid 1S1 \mid 0S0$ δίνουν συμβολοσειρές περιττού μήκους με όλους τους συνδυασμούς 0 και 1 και τερματίζουν με 0 ή 1 οπότε προκύπτει πάντα περιττό μήκος

14) Να κατασκευάσετε γραμματική χωρίς συμφραζόμενα για τη γλώσσα $L = \{w \in \{a,b\}^* \mid \eta w \text{ έχει διπλάσια } a \text{ από } b\}$

Απάντηση

$S \rightarrow aSaSbS \mid bSaSaS \mid aSbSa \mid \epsilon$

Παραδείγματα

$S \Rightarrow SS \Rightarrow aSaSb \Rightarrow aab, S \Rightarrow aSbSa \Rightarrow ab\underline{SaS}a\underline{SaS}b \Rightarrow abaabaaba$

Με τους κανόνες $S \rightarrow aSaSbS \mid bSaSaS \mid aSbSa$ μπαίνουν διπλάσια a από b σε όλες τις δυνατές θέσεις

17) Να κατασκευάσετε γραμματική χωρίς συμφραζόμενα για τη γλώσσα $L = \{w \in \{a, b\}^* \mid \eta w \text{ περιέχει άρτιο πλήθος } a\}$

Απάντηση

$S \rightarrow SaSaS \mid bS \mid \epsilon$

Παραδείγματα

$S \Rightarrow \epsilon$

$S \Rightarrow SaSaS \rightarrow aa$

$S \Rightarrow bS \Rightarrow b$

$S \Rightarrow bS \Rightarrow bSaSaS \rightarrow baa$

Με τον κανόνα $S \rightarrow SaSaS$ κάθε φορά προστίθενται διπλάσια a. Ο κανόνας $S \rightarrow bS$ βάζει οσαδήποτε b αφού γιαυτά δεν υπάρχει περιορισμός

15) $L = \{ w \in \{a, b\}^* \mid \text{η } w \text{ έχει n "1" ακολουθούμενα από } 3n \text{ "0" με } n \geq 0 \}$

Απάντηση

$S \rightarrow 1S000 \mid \epsilon$

3.3 Θέματα με Κατασκευή Γραμματικών ΓΧΣ

3.3.1 Θέμα 5 Ιούνιος 2010

(α) Παρουσιάστε μια BNF γραμματική χωρίς συμφραζόμενα η οποία να παράγει/αναγνωρίζει όλες τις δυνατές ακολουθίες χαρακτήρων αποτελουμένες από a και b με την εξής ιδιότητα: **Αρχίζουν με n a, ακολουθούμενα από 2n b όπου n≥0**. Παραδείγματα μελών της γλώσσας: abb, aaabbbaaaabb

(β) Χρησιμοποιώντας τη γραμματική σας, να αποφανθείτε αν οι ακολουθίες **aabb** και **aabb** είναι μέλη της γλώσσας κατασκευάζοντας **τα δέντρα συντακτικής ανάλυσης τους**

(γ) Ποιες **ακολουθίες χαρακτήρων** παράγει/αναγνωρίζει η παρακάτω BNF γραμματική με αλφάριθμο {a, b, c}. Ποιες είναι οι δύο βασικές ιδιότητες των ακολουθιών αυτών σε σχέση i) **με τη θέση τους** και ii) **με τον αριθμό τους**

$\langle F \rangle ::= a \langle F \rangle c | \langle T \rangle$

$\langle T \rangle ::= b \langle T \rangle c | \epsilon$

Απάντηση

(α) $\langle S \rangle ::= a \langle S \rangle bb | \epsilon$

Παραδείγματα

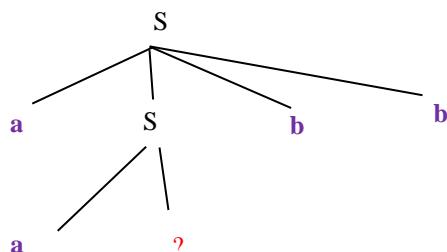
$S \Rightarrow \epsilon$

$S \Rightarrow aSbb \Rightarrow abbb$

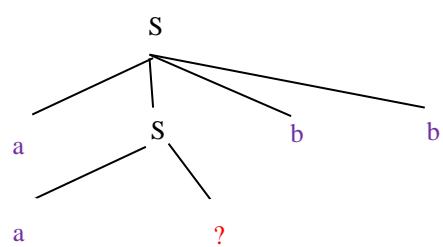
$S \Rightarrow aSbb \Rightarrow aaSbbbb \Rightarrow aabbbaaaabb$

(β) Οι δύο συμβολοσειρές **ΔΕΝ ανήκουν στη γλώσσα διότι ΔΕΝ μπορεί να κατασκευαστεί parse tree γιαυτές**. Πιο συγκεκριμένα έχουμε τα εξής:

aabb



aabb



(γ)

$\langle F \rangle ::= a \langle F \rangle c | \langle T \rangle$

$\langle T \rangle ::= b \langle T \rangle c | \epsilon$

Παραγωγές

$F \Rightarrow T \Rightarrow \epsilon$

$F \Rightarrow aFc \Rightarrow aTc \Rightarrow ac$

$F \Rightarrow aFc \Rightarrow aaFcc \Rightarrow aaTcc \Rightarrow aacc$

$F \Rightarrow aFc \Rightarrow aaFcc \Rightarrow aaTcc \Rightarrow aabTccc \Rightarrow abccc$

$F \Rightarrow T \Rightarrow bTC \Rightarrow bc$

$F \Rightarrow T \Rightarrow bTC \Rightarrow bbTcc \Rightarrow bbcc$

$F \Rightarrow aFc \Rightarrow aaFcc \Rightarrow aaTcc \Rightarrow aabTccc \Rightarrow abccc$

Χαρακτηριστικά Συμβολοσειρών

- Οι ΜΗ κενές συμβολοσειρές αρχίζουν με α ή β και τελειώνουν σε γ
- Αν υπάρχουν και α και β και γ τα α προηγούνται των β και τα β προηγούνται των γ
- Ο συνολικός **αριθμός των α και β μαζί** είναι ίσος με το **συνολικό αριθμό των γ**
- Οι συμβολοσειρές έχουν άρτιο μήκος (σε σχέση με τον αριθμό τους)

3.3.2 Θέμα 2 Ατυπη Νοέμβριος 2011

Θέλουμε να ορίσουμε μια γραμματική $G=(\Sigma, N, R, S)$ με αλφάβητο $\Sigma=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, =, X\}$ που θα περιγράφει τις εκφράσεις πρόσθεσης/αφαίρεσης δύο μονοψήφιων ακεραίων αριθμών της μορφής $X=-2+6$. Σημειώστε ότι το $-$ χρησιμοποιείται τόσο ως **μοναδιαίος όσο και ως δυαδικός τελεστής**. Οι εκφράσεις δεν περιλαμβάνουν κενά.

(a) Περιγράψτε το συντακτικό της γλώσσας με χρήση συμβολισμού BNF

(b) Να επεκτείνετε τη γραμματική G ώστε να περιλαμβάνονται στις αποδεκτές εκφράσεις και εκφράσεις πρόσθεσης/αφαίρεσης περισσότερων των δύο αριθμών χωρίς παρενθέσεις

(c) Κατασκευάζοντας τα **δέντρα συντακτικής ανάλυσης** (parse tree) για τις εκφράσεις $X=9-5+6$ και $X=-3+8+$ να αποφανθείτε αν δύο εκφράσεις είναι μέλη της γλώσσας

Απάντηση

a)

$<S> ::= X= <\text{UNAR_OPER}> <\text{DIGIT}> <\text{BIN_OPER}> <\text{DIGIT}>$

$<\text{UNAR_OPER}> ::= - \mid \epsilon$

$<\text{BIN_OPER}> ::= + \mid -$

$<\text{DIGIT}> ::= 0|1|2|3|4|5|6|7|8|9$

Παραδείγματα

$S \Rightarrow X = \text{UNAR_OPER } \text{DIGIT } \text{BIN_OPER } \text{DIGIT} \Rightarrow X = -2+6$

$S \Rightarrow X = \text{UNAR_OPER } \text{DIGIT } \text{BIN_OPER } \text{DIGIT} \Rightarrow X = 2+6$

$S \Rightarrow X = \text{UNAR_OPER } \text{DIGIT } \text{BIN_OPER } \text{DIGIT} \Rightarrow X = 2-6$

b)

- $\langle S \rangle ::= X = \langle UNAR_OPER \rangle \langle DIGIT \rangle \langle BIN_OPER \rangle \langle NEXTNUM \rangle$
- $\langle NEXTNUM \rangle ::= \langle DIGIT \rangle \langle BIN_OPER \rangle \langle NEXTNUM \rangle \mid \langle DIGIT \rangle$
- $\langle UNAR_OPER \rangle ::= - \mid \epsilon$
- $\langle BIN_OPER \rangle ::= + \mid -$
- $\langle DIGIT \rangle ::= 0|1|2|3|4|5|6|7|8|9$

Παραδείγματα

$S \Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ NEXTNUM} \Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ NEXTNUM}$

$\Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \Rightarrow X = 9 - 5 + 6$

$S \Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ NEXTNUM} \Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \underline{\text{DIGIT}} \underline{\text{BIN_OPER}} \underline{\text{NEXTNUM}}$

$\Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \text{ BIN_OPER} \underline{\text{DIGIT}} \underline{\text{BIN_OPER}} \underline{\text{NEXTNUM}}$

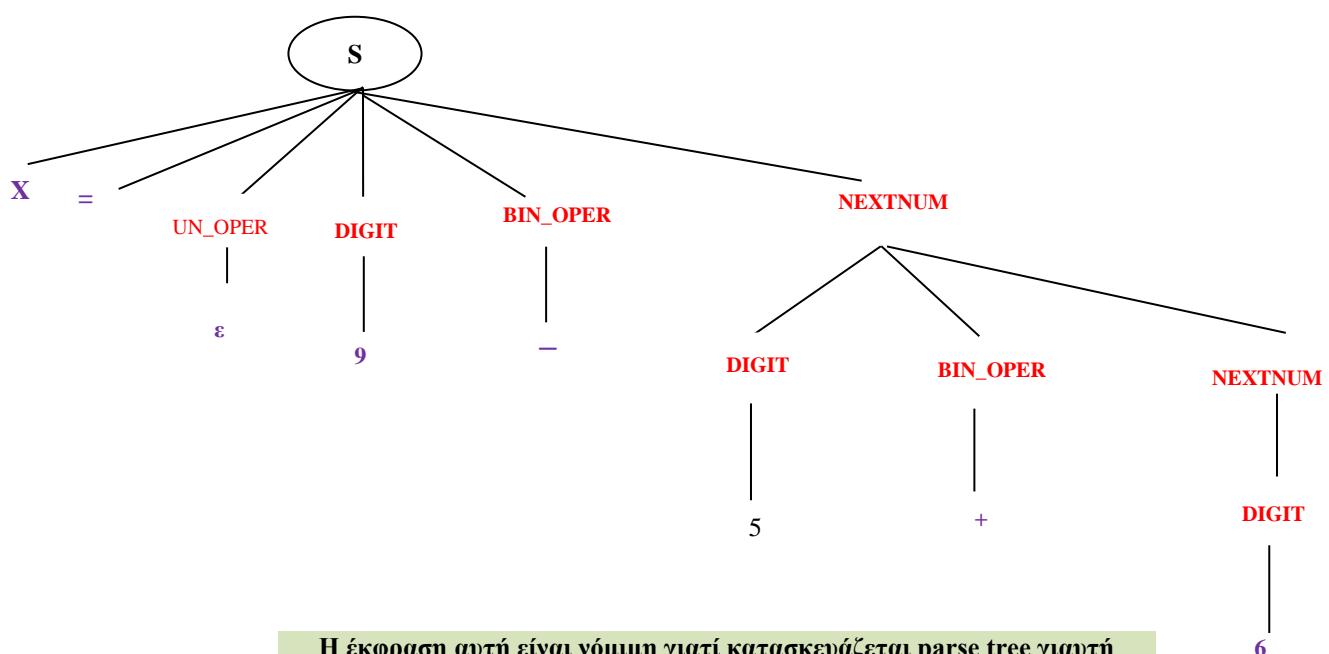
$\Rightarrow X = \text{UNAR_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \text{ BIN_OPER} \text{ DIGIT} \underline{\text{BIN_OPER}} \underline{\text{DIGIT}}$

$\Rightarrow X = 5 - 7 + 9 - 6$

c)

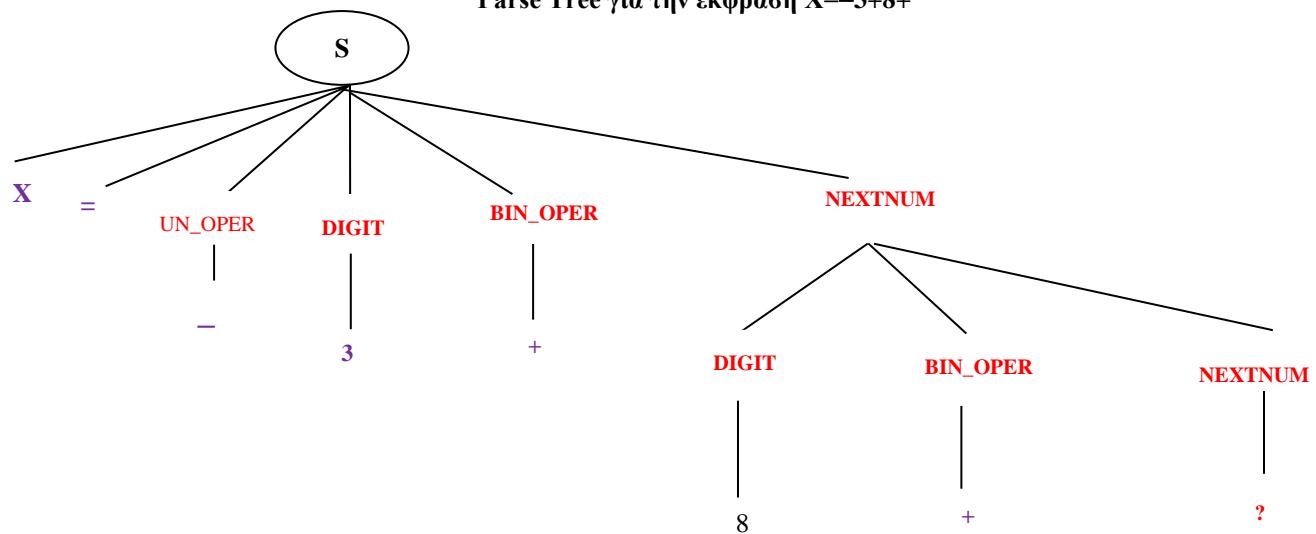
i)

Parse Tree για την έκφραση $X = 9 - 5 + 6$



ii)

Parse Tree για την έκφραση $X = -3 + 8 +$



Η έκφραση αυτή δεν είναι νόμιμη γιατί δεν κατασκευάζεται parse tree γιαυτή

3.3.3 Θέμα 2 Σεπτέμβριος 2012

Σας δίνεται η παρακάτω BNF γραμματική:

```

<A> ::= <B> <D>
<B> ::= ε | <C> ; <B>
<C> ::= INT <I>
<D> ::= <E> | <D> ; <E>
<E> ::= <I> = <F> | PRINT <I>
<F> ::= <G> | <F> + <G> | <F> - <G>
<G> ::= <H> | <I>
<H> ::= <J> | <H> <J>
<I> ::= a | b | c
<J> ::= 0 | 1

```

στην οποία είναι το "τίποτα" (null) και ; INT = PRINT + - a b c 0 1 σύμβολα της γλώσσας. Χαρακτήρας(-ες) κενού (space) μπορεί να υπάρχει(-ου) της γλώσσας, χωρίζοντας τα τερματικά σύμβολα.

(a) Περιγράψτε τη μικρή γλώσσα προγραμματισμού που παράγεται από την πο
Τι εντολές φαίνεται να έχει η γλώσσα, ποια(-ες) δομή(-ες) μπορεί να έχει και τι είδους αντικείμενα μπορεί να χειριστεί; Ποιες είναι οι δυνατότητες σε σχέση με μια γνωστή γλώσσα όπως η C;

(b) Γράψτε ένα μικρό – αλλά πλήρες – πρόγραμμα της γλώσσας, το οποίο ν
υποχρεωτικές και προαιρετικές εντολές και δομές που προβλέπει η γραμματική. Κατασκευάστε το δέντρο συντακτικής ανάλυσης (parse tree) για αποφανθείτε αν είναι νόμιμο με βάση τη γραμματική που δίνεται.

Απάντηση

Ενδεικτικές Παραγωγές

A⇒BD⇒D⇒D;E ⇒E;PRINT I⇒I=F; PRINT I; a=G;PRINT I ⇒a=H; PRINT I ⇒ a=J; PRINT I ⇒ a=0;PRINT I⇒a=0;PRINT a

A⇒BD⇒C;BD;E ⇒ INT I; E; I=F ⇒ INT I; PRINT I; b=F+G ⇒ INT I; PRINT I; b=G+H ⇒ INT I; PRINT I; b=I+J ⇒ INT I; PRINT I; b=c+1 ⇒ INT b; PRINT b; b=c+1

A⇒BD⇒ E ⇒I=F ⇒ c=F-G ⇒c= G - I ⇒ c= I - b ⇒ c=a-b

A⇒BD⇒ C;BD;E ⇒ INT I;C;B E ; PRINT I ⇒ INT I; INT I; I=F ; PRINT I ⇒ INT I; INT I; a=F+G ; PRINT I ⇒ INT I; INT I; a= G + I ; PRINT I ⇒ INT I; INT I; a= I + b; PRINT I ⇒ INT a; INT b; a=a+b; PRINT a

A⇒BD⇒C;BD⇒INT I;CBE⇒ INT I; INT I; I=F⇒ INT I; INT I; a=G ⇒ INT a; INT b; a=b

(a)

Οι εντολές της γλώσσας είναι:

- Καταχώρισης
- Εκτύπωσης
- Δήλωσης Μεταβλητών
- Εντολές με αριθμητικές παραστάσεις με τους τελεστές + και -

Το πρόγραμμα χρησιμοποιεί την ακολουθιακή δομή διότι οι εντολές εκτελούνται η μια μετά την άλλη.

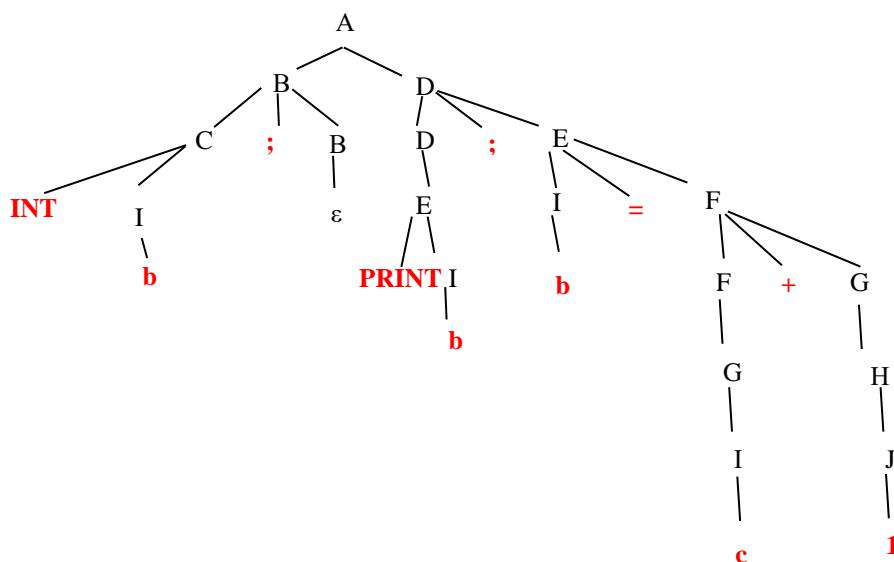
Σημείωση

- Η δομή επανάληψης προσδιορίζεται από εντολές επανάληψης όπως for, while και do..while
 - Η δομή επιλογής προσδιορίζεται από εντολές ελέγχου όπως if και switch
 - Η ακολουθιακή δομή προσδιορίζεται από εντολές καταχώρισης και εκτύπωσης που εκτελούνται η μια μετά την άλλη
- Η γλώσσα αυτή χειρίζεται αποκλειστικά ακέραια αντικείμενα τύπου INT και μπορεί να τα εκτυπώνει και να καταχωρεί σε αυτά είτε αποτελέσματα αριθμητικών εκφράσεων με + και - είτε άλλα ακέραια αντικείμενα.
- Οι δυνατότητες αυτής της γλώσσας σε σχέση με μια γνωστή γλώσσα όπως η C είναι ότι κάνει **προαιρετική δήλωση μεταβλητών** (όχι υποχρεωτικά όπως απαιτεί η C), έχει εντολές καταχώρισης και εκτύπωσης όπως η C. Επίσης υπολογίζει αριθμητικές εκφράσεις όπως η C αλλά όχι με όλους τους αριθμητικούς τελεστές της C (χρησιμοποιεί μόνο + και -) και όχι με όλους τους τύπους της C (χρησιμοποιεί μόνο τον τύπο INT). Επίσης σε σχέση με τη C δεν έχει δομή επιλογής και δομή επανάληψης. Η συγκεκριμένη γλώσσα μπορεί να μην βάζει ερωτηματικά σε εντολές καταχώρισης ενώ στη C το ; είναι ερωτηματικό είναι υποχρεωτικό σε κάθε εντολή καταχώρισης

(b)

Ένα μικρό αλλά πλήρες πρόγραμμα είναι η δεύτερη παραγωγή από αυτές που δείξαμε πριν δηλ το **INT b; PRINT b; b=c+1**

$A \Rightarrow BD \Rightarrow C; BD; E \Rightarrow INT\ I; E; I=F \Rightarrow INT\ I; PRINT\ I; b=F+G \Rightarrow INT\ I; PRINT\ I; b=G+H \Rightarrow INT\ I; PRINT\ I; b=I+J \Rightarrow INT\ I; PRINT\ I; b=c+1 \Rightarrow \underline{INT\ b}; \underline{PRINT\ b}; \underline{b=c+1}$



3.3.4 Θέμα 1 Ιούνιος 2009

Θέλουμε να ορίσουμε μια γλώσσα η οποία να αποτελείται από όλα τα string που περιλαμβάνουν χαρακτήρες από τα 4 γράμματα α, β, γ, δ με την εξής ιδιομορφία: **Τα α, β εμφανίζονται οπωσδήποτε στο string αλλά μόνο σε μια ακολουθία από n α ακολουθόνυμα από n β όπου n ακέραιος με n>0 και χωρίς να παρεμβάλλεται μεταξύ των ακολουθιών των α και β άλλος χαρακτήρας.** Οι χαρακτήρες γ και δ μπορεί να βρίσκονται στην αρχή ή και στο τέλος της συμβολοσειράς και έχουν οποιοδήποτε πλήθος. Για παράδειγμα τα string **αβ, γαααβββδδδ** ανήκουν στη γλώσσα αλλά όχι τα string **α, αββ, ββαα, ααγββ, αααβββδβ**.

(α) Περιγράψτε το συντακτικό της γλώσσας με χρήση BNF

(β) Μεταφράστε τη γραμματική BNF σε συντακτικά διαγράμματα

(γ) Κατασκευάζοντας τα δέντρα συντακτικής ανάλυσης για τα string ααβγγ και αααββδ δνα αποφανθείτε αν τα δύο string είναι μέλη της γλώσσας

Απάντηση

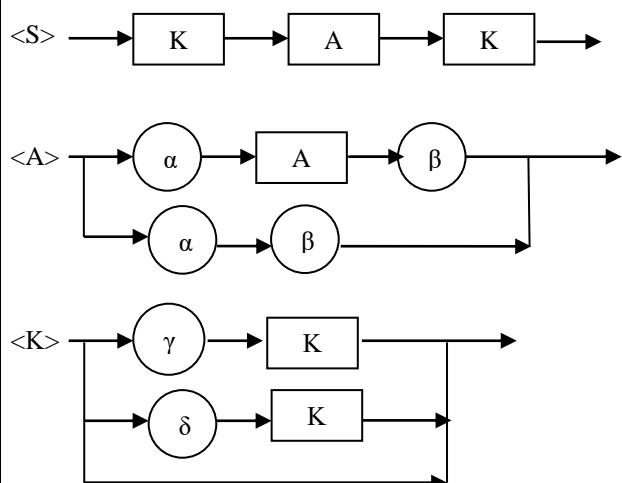
(α)

$<S> ::= <K><A><K>$

$<A> ::= \alpha <A>\beta \mid \alpha\beta$

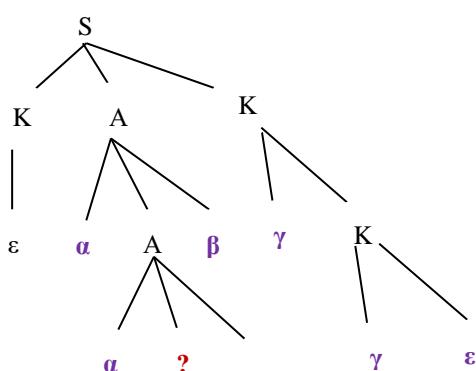
$<K> ::= \gamma <K> \mid \delta <K> \mid \epsilon$

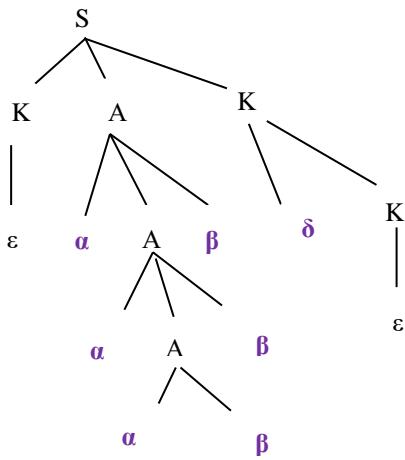
(β) Συντακτικά Διαγράμματα Κανόνων



(γ)

Parse Tree για το string ααβγγ. Το string **ααβγγ ΔΕΝ ανήκει στη γλώσσα** γιατί ΔΕΝ μπορεί να κατασκευαστεί Parse Tree γιαντό





3.3.5 Θέμα 3 Σεπτέμβριος 2004

Θέλουμε να ορίσουμε μια γλώσσα περιγραφής **τριψήφιων ακεραίων** (προσημασμένων και μη).

a) Περιγράψτε το συντακτικό με χρήση BNF

β) Μεταφράστε τη γραμματική BNF σε συντακτικά διαγράμματα

Απάντηση

a) Η γραμματική BNF είναι η εξής:

$<A> ::= <UN_OP> <NUM> <NUM> <NUM>$

$<UN_OP> ::= - | \epsilon$

$<NUM> ::= 0 | 1 | 2 | \dots | 9$

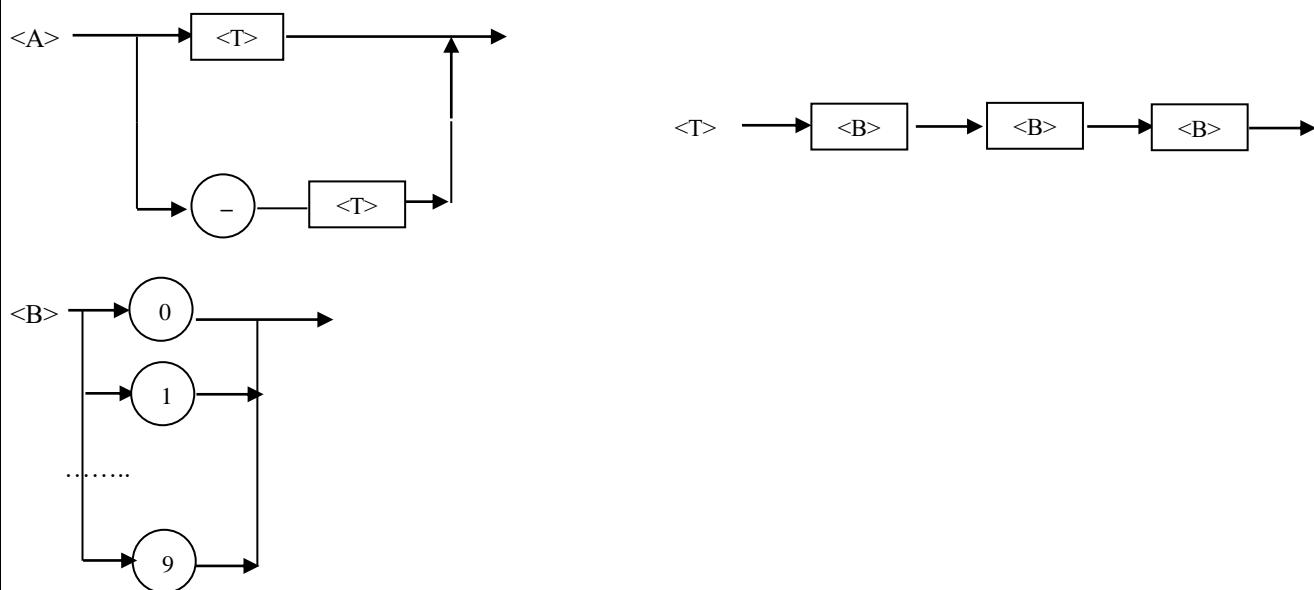
είτε εναλλακτικά η εξής:

$<A> ::= <T> | -<T>$

$<T> ::= $

$::= 0 | 1 | 2 | \dots | 9$

β) Τα συντακτικά διαγράμματα είναι τα ακόλουθα:



3.4 Θέματα με Γραμματικές LL/1

3.4.1 Προϋποθέσεις Γραμματικής LL/1

- **Να MHN περιέχει Αριστερή Αναδρομή δηλ. κανόνες της μορφής $A \rightarrow Aa$ δηλ. το MH τερματικό σύμβολο αριστερά του κανόνα να βρίσκεται και στην 1^η θέση δεξιά του κανόνα**
- **Να MHN έχει εναλλακτικούς κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο δηλ. κανόνες της μορφής $A \rightarrow aB | a\Gamma$**

3.4.2 Κανόνες Μετατροπής-Διόρθωσης Γραμματικής σε LL/1

Αριστερή Αναδρομή

$A \rightarrow \Delta \alpha_1 | \Delta \alpha_2 | \dots | \Delta \alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$

Η απαλοιφή της Αριστερής Αναδρομής γίνεται ως εξής:

$A \rightarrow \beta_1 B | \beta_2 B | \dots | \beta_m B$

$B \rightarrow \alpha_1 B | \alpha_2 B | \dots | \alpha_n B | \epsilon$

1. Πηγαίνουμε σε κάθε κανόνα που ΔΕΝ έχει αριστερή αναδρομή και βάζουμε στο τέλος του ένα νέο MH τερματικό σύμβολο. Π.χ. οι κανόνες $\beta_1 | \beta_2 | \dots | \beta_m$ δεν έχουν αναδρομή και προσθέτουμε στο τέλος τους ένα νέο MH τερματικό σύμβολο π.χ. το B
2. Από το νέο MH τερματικό σύμβολο π.χ. από το B πηγαίνουμε σε κάθε κανόνα που έχει αριστερή αναδρομή, γράφουμε ότι υπάρχει μετά την αναδρομή ακολουθούμενο από το νέο σύμβολο ή κενό. Π.χ. πηγαίνουμε στο α_1 που είναι μετά την αναδρομή και βάζουμε το νέο σύμβολο δηλ. το B. Μετά πηγαίνουμε στο α_2 που είναι μετά την αναδρομή και βάζουμε το νέο σύμβολο δηλ. το B. Επαναλαμβάνουμε την ίδια διαδικασία μέχρι και το α_n και στο τέλος βάζουμε ένα επιπλέον κανόνα που οδηγεί στο ε

Εναλλακτικοί Κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο

$A \rightarrow a \beta_1 | a \beta_2 | \dots | a \beta_n$

Η Αριστερή Παραγοντοποίηση γίνεται ως εξής:

$A \rightarrow aB$

$B \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$

1. Βγάζουμε από κάθε εναλλακτικό κανόνα που αρχίζει με το ίδιο τερματικό σύμβολο το σύμβολο αυτό ως κοινό παράγοντα και προσθέτουμε μετά τον κοινό παράγοντα ένα νέο MH τερματικό σύμβολο π.χ. βγάζουμε το a ως κοινό παράγοντα και μετά προσθέτουμε ένα νέο μη τερματικό σύμβολο π.χ. το B
2. Από το νέο MH τερματικό σύμβολο πηγαίνουμε σε κάθε κανόνα που έχει κοινό παράγοντα και γράφουμε ότι υπάρχει META τον κοινό παράγοντα. Αν δεν υπάρχει κάτι μετά βάζουμε το ε. Π.χ. πηγαίνουμε στον κανόνα $a\beta_1$ και γράφουμε ότι είναι δηλ. μετά τον κοινό παράγοντα δηλ. μόνο το β_1 , μετά πηγαίνουμε στον κανόνα $a\beta_2$ και γράφουμε ότι είναι δηλ. μετά τον κοινό παράγοντα δηλ. μόνο το β_2 και το ίδιο γίνεται μέχρι και το β_n

3.4.3 Παράδειγμα 1 με LL/1 γραμματική

Η ακόλουθη γραμματική είναι LL/1; Αν όχι να τη μετατρέψετε σε LL/1

$<S> ::= <A>a \mid b$

$<A> ::= <A>c \mid <S>d \mid f$

Απάντηση

Η Γραμματική δεν είναι LL/1 διότι ο κανόνας $<A> ::= <A>c$ παρουσιάζει **αριστερή αναδρομή**:

Η απαλοιφή της αριστερής αναδρομής γίνεται ως εξής:

$<A> ::= <S>d <K> \mid f <K>$

$<K> ::= c <K> \mid \epsilon$

Η τελική LL/1 γραμματική είναι:

$<S> ::= <A>a \mid b$

$<A> ::= <S>d <K> \mid f <K>$

$<K> ::= c <K> \mid \epsilon$

3.4.4 Παράδειγμα 2 με LL/1 γραμματική

Να εξετάσετε αν η ακόλουθη γραμματική είναι LL/1 και αν όχι να τη μετατρέψετε σε LL/1

$E \rightarrow E + T \mid T$

$T \rightarrow T * E \mid F$

$F \rightarrow (E) \mid id$

Απάντηση

Η Γραμματική δεν είναι LL/1 και τη μετατρέπουμε σε LL/1 κάνοντας απαλοιφή της αριστερής αναδρομής που υπάρχει στους κανόνες $E \rightarrow E + T$ και $T \rightarrow T * E$. Η νέα γραμματική που προκύπτει είναι η ακόλουθη:

E → TB

B → +T B | ε

T → FK

K → *EK | ε

$F \rightarrow (E) \mid id$

3.4.5 Θέμα 3 Σεπτέμβριος 2010

(α) Δίνεται η ακόλουθη γραμματική:

$\langle X \rangle ::= \langle B \rangle a \langle A \rangle b$

$\langle A \rangle ::= a \langle A \rangle \mid a$

$\langle B \rangle ::= \langle B \rangle b \mid b$

Ποια είναι τα χαρακτηριστικά των μελών της γλώσσας που παράγονται από την παραπάνω γραμματική;

(β) Είναι **αποδεικτά τα ακόλουθα string** από τη γραμματική αυτή;

- i) baab
- ii) bbbaabb
- iii) bbbaab
- iv) bbbbbaa

Δεν χρειάζεται να κατασκευάσετε parse tree για όλα

(γ) Να κάνετε τους **κατάλληλους μετασχηματισμούς** έτσι ώστε να γίνει η γραμματική LL/1

Απάντηση

$X \Rightarrow BaAb \Rightarrow \textcolor{red}{baab}$

$X \Rightarrow BaAb \Rightarrow \underline{B}baab \Rightarrow \textcolor{red}{bbaab}$

$X \Rightarrow BaAb \Rightarrow \underline{baa}Ab \Rightarrow \textcolor{red}{baaab}$

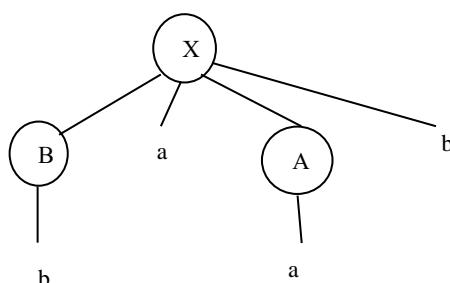
$X \Rightarrow BaAb \Rightarrow \underline{Bbaa}Ab \Rightarrow \underline{Bbb}aaab \Rightarrow \textcolor{red}{bbbbaab}$

(α) Τα χαρακτηριστικά των μελών της γλώσσας που παράγονται από την παραπάνω γραμματική;

- Αρχίζουν και τελειώνουν με b
- Στο τέλος κάθε συμβολοσειράς υπάρχει ακριβώς ένα b ενώ στην αρχή της υπάρχει τουλάχιστον ένα b
- Μεταξύ των a δεν παρεμβάλλονται b
- Το μήκος των συμβολοσειρών είναι τουλάχιστον 4 χαρακτήρες
- Περιέχουν τουλάχιστον 2 χαρακτήρες a και τουλάχιστον 2 χαρακτήρες b

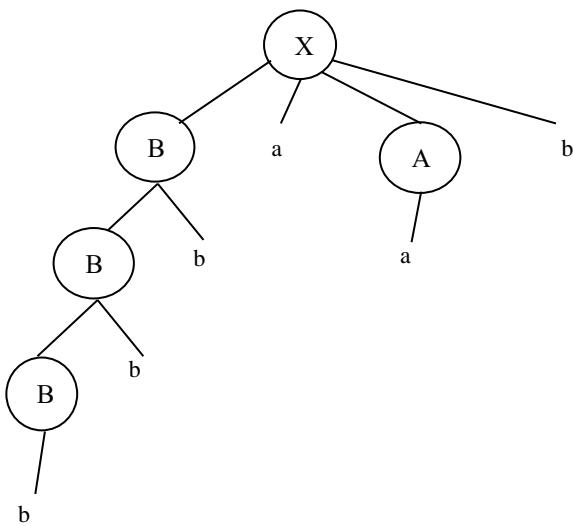
(β)

Η συμβολοσειρά **baab είναι αποδεκτή από τη γλώσσα γιατί μπορεί να κατασκευαστεί parse tree γιαντί**. Στα φύλλα του βρίσκεται η συμβολοσειρά.



Η συμβολοσειρά **bbbbaabb ΔΕΝ είναι αποδεκτή από τη γλώσσα διότι τελειώνει σε 2 b και ΔΕΝ μπορεί να κατασκευαστεί parse tree γιαυτή**

Η συμβολοσειρά **bbbaab είναι αποδεκτή από τη γλώσσα γιατί μπορεί να κατασκευαστεί parse tree γιαυτή** που είναι το ακόλουθο:



Η συμβολοσειρά **bbbbbaaa ΔΕΝ είναι αποδεκτή από τη γλώσσα διότι τερματίσει σε a και ΔΕΝ μπορεί να κατασκευαστεί parse tree γιαυτή.**

(γ) Η ακόλουθη γραμματική:

$$\langle X \rangle ::= \langle B \rangle a \langle A \rangle b$$

$$\langle A \rangle ::= \mathbf{a} \langle A \rangle \mid \mathbf{a}$$

$$\langle B \rangle ::= \langle B \rangle b \mid b$$

ΔΕΝ είναι LL/1 διότι υπάρχουν εναλλακτικοί κανόνες $\langle A \rangle ::= a \langle A \rangle \mid a$ που αρχίζουν με το ίδιο τερματικό σύμβολο a και ο κανόνας $\langle B \rangle ::= \langle B \rangle b \mid b$ παρουσιάζει αριστερή αναδρομή

Μετατροπή Γραμματικής σε LL/1

Στο 2^o κανόνα $\langle A \rangle ::= a \langle A \rangle \mid a$ πρέπει να κάνουμε αριστερή Παραγοντοποίηση ενώ στον 3^o κανόνα $\langle B \rangle ::= \langle B \rangle b \mid b$ πρέπει να κάνουμε απαλοιφή αριστερής αναδρομής.

Ο 2^o κανόνας αναλύεται στους εξής: $\langle A \rangle ::= \mathbf{a} \langle K \rangle$

$$\langle K \rangle ::= \langle A \rangle \mid \varepsilon$$

Ο 3^o κανόνας αναλύεται στους εξής: $\langle B \rangle ::= b \langle L \rangle$

$$\langle L \rangle ::= b \langle L \rangle \mid \varepsilon$$

Η τελική διορθωμένη LL/1 γραμματική είναι:

$\langle X \rangle ::= \langle B \rangle a \langle A \rangle \beta$

$\langle A \rangle ::= a \langle K \rangle$

$\langle K \rangle ::= \langle A \rangle | \varepsilon$

$\langle B \rangle ::= b \langle \Lambda \rangle$

$\langle \Lambda \rangle ::= b \langle \Lambda \rangle | \varepsilon$

3.4.6 Θέμα 3 Ατυπη Φεβρουάριος 2012 και Σεπτέμβριος 2016

- (α) Παρουσιάστε μια BNF γραμματική η οποία να παράγει όλες τις δυνατές ακολουθίες χαρακτήρων που αποτελούνται από παρενθέσεις () ή/και αγκύλες [] σωστά φωλιασμένες π.χ. () [] ()()
- (β) Είναι η γραμματική σας LL/1; Αν όχι παρουσιάστε μια ισοδύναμη γραμματική που είναι LL/1
- (γ) Χρησιμοποιώντας τη γραμματική σας (κατά προτίμηση την LL/1) παρουσιάστε το δέντρο συντακτικής ανάλυσης (parse tree) για την ακολουθία ([] []) [] ()()

Απάντηση

(α) $\langle S \rangle ::= (\langle S \rangle) \langle S \rangle \mid [\langle S \rangle] \langle S \rangle \mid \epsilon$

(β)

Η γραμματική που κατασκευάσαμε:

- Δεν έχει αριστερή αναδρομή
- Δεν έχει εναλλακτικούς κανόνες που ξεκινούν με το ίδιο τερματικό σύμβολο
- **Άρα είναι LL/1 γραμματική**

Παρατήρηση

Μια σωστή γραμματική **που δεν είναι LL/1** είναι η εξής: $\langle S \rangle ::= () \mid [] \mid (\langle S \rangle) \mid [\langle S \rangle] \mid \langle S \rangle \langle S \rangle$

Διόρθωση κάνοντας παραγοντοποίηση:

$\langle S \rangle ::= (\langle B \rangle$

$\langle B \rangle ::=) \mid \langle S \rangle$

$\langle S \rangle ::= [\langle C \rangle$

$\langle C \rangle ::=] \mid \langle S \rangle$

$\langle S \rangle ::= \langle S \rangle \langle S \rangle$

Νέα Διόρθωση με απαλοιφή αναδρομής:

$\langle S \rangle ::= (\langle B \rangle \langle D \rangle \mid [\langle C \rangle \langle D \rangle$

$\langle D \rangle ::= \langle S \rangle \langle D \rangle \mid \epsilon$

Η τελική διορθωμένη LL/1 Γραμματική είναι η ακόλουθη:

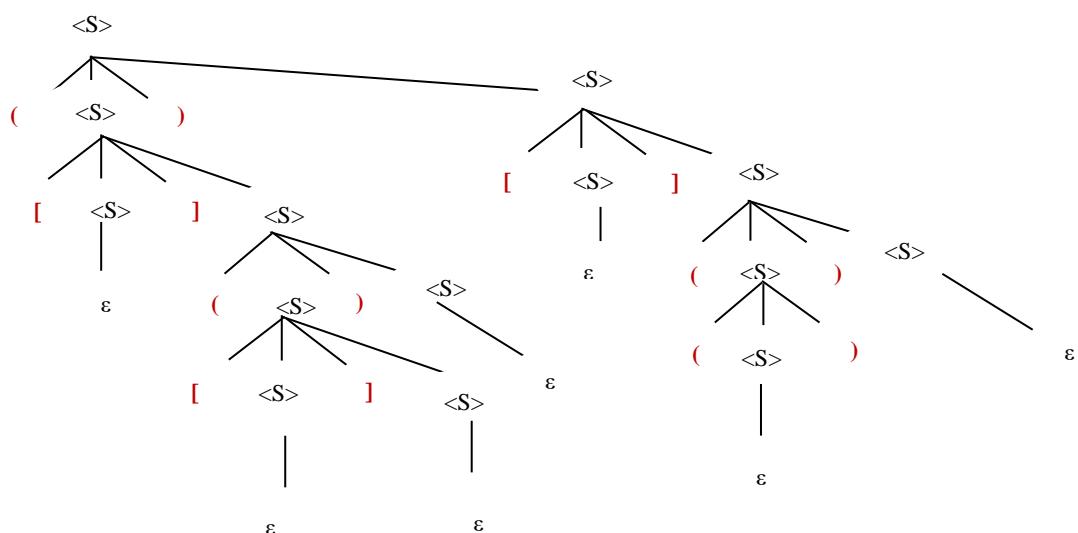
$\langle S \rangle ::= (\langle B \rangle \langle D \rangle \mid [\langle C \rangle \langle D \rangle$

$\langle D \rangle ::= \langle S \rangle \langle D \rangle \mid \epsilon$

$\langle B \rangle ::=) \mid \langle S \rangle$

$\langle C \rangle ::=] \mid \langle S \rangle$

(γ) To parse tree για τη συμβολοσειρά $\{ \Pi (\Pi) \} \Pi (\Omega)$ με βάσει τη γραμματική $\langle S \rangle ::= \langle S \rangle \langle S \rangle \mid [\langle S \rangle] \langle S \rangle \mid \varepsilon$ φαίνεται στην ακόλουθη εικόνα:



3.5 Θέματα με Parsers

3.5.1 Παράδειγμα με Στοίβα Ολίσθησης-Ελάττωσης (Bottom-Up Parser)

Δίνεται η ακόλουθη γραμματική:

$$\begin{aligned} <S> &::= r \\ &::= <D> \mid , <D> \\ <D> &::= a \mid b \end{aligned}$$

Να αναγνωρίσετε τη συμβολοσειρά **ra,b** κατασκευάζοντας τη στοίβα Ολίσθησης-Ελάττωσης (Bottom-Up Parser)

Απάντηση

Παραγωγή: $S \Rightarrow rB \Rightarrow r B, D \Rightarrow rD, D \Rightarrow ra, b$

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

Βασικές Παρατηρήσεις

Βήμα 4: Παρόλο που το $r $ βρίσκεται από μόνο του στο δεξιό μέρος του κανόνα $<S> ::= r $ εντούτοις δεν αντικαθιστούμε το $r $ με το $<S>$ διότι για να βάλουμε στη στοίβα το αρχικό σύμβολο της γραμματικής πρέπει η συμβολοσειρά εισόδου να έχει εξαντληθεί. αλλιώς θα έχουμε σφάλμα Ολίσθησης-Ελάττωσης.

Βήμα 7: Μπορούμε να εφαρμόσουμε δύο κανόνες είτε τον κανόνα $::= <D>$ είτε τον κανόνα $::= , <D>$. Επιλέγουμε τον κανόνα που διώχνει το μεγαλύτερο κομμάτι μέσα από τη στοίβα. Αν δεν εφαρμόσουμε το σωστό κανόνα τότε θα έχουμε σφάλμα Ελάττωσης-Ελάττωσης.

3.5.2 Παράδειγμα με Στοίβα Πρόβλεψης-Ταιριάσματος (Top-Down Parser)

Δίνεται η ακόλουθη γραμματική.

Γραμματική: $<S> ::= (<S>)<S> \mid \epsilon$

Να αναγνωρίσετε τη συμβολοσειρά () κατασκευάζοντας τη στοίβα Πρόβλεψης-Ταιριάσματος (Top-Down Parser)

Απάντηση

$<S> \Rightarrow (<S>) <S> \Rightarrow ()$

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

3.5.3 Θέμα 4 Ιούνιος 2012

Δίνεται η παρακάτω LL(1) BNF γραμματική:

$\langle X \rangle ::= \langle A \rangle | \langle B \rangle$

$\langle A \rangle ::= a \langle A \rangle b \mid \epsilon$

$\langle B \rangle ::= c \langle B \rangle d \mid \epsilon$

(α) Περιγράψτε τα μέλη της γλώσσας που παράγονται από τη γραμματική

(β) Είναι το **cceddd** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας την **bottom-up** στοίβα συντακτικής ανάλυσης για το string

(γ) Είναι το **aabb** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας την **top-down** στοίβα συντακτικής ανάλυσης για το string

Απάντηση

Ενδεικτικές Παραγωγές

$X \Rightarrow A \Rightarrow \epsilon$

$X \Rightarrow A \Rightarrow aAb \Rightarrow ab$

$X \Rightarrow A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aabb$

$X \Rightarrow B \Rightarrow cBd \Rightarrow cd$

$X \Rightarrow B \Rightarrow cBd \Rightarrow ccBdd \Rightarrow cedd$

$X \Rightarrow B \Rightarrow cBd \Rightarrow ccBdd \Rightarrow cccBddd \Rightarrow cccddd$

(α)

Χαρακτηριστικά Συμβολοσειρών

- Έχουν άρτιο μήκος συμβολοσειρών
- Αποτελούνται είτε από a και b είτε από c και d
- Αν η συμβολοσειρά αποτελείται από a και b τα a προηγούνται των b και ο συνολικός αριθμός των a ισούται με το συνολικό αριθμό των b
- Αν η συμβολοσειρά αποτελείται από c και c τα c προηγούνται των d και ο συνολικός αριθμός των c ισούται με το συνολικό αριθμό των d
- Στις συμβολοσειρές της γλώσσας περιλαμβάνεται η κενή συμβολοσειρά

(β) Η bottom-up στοίβα συντακτικής ανάλυσης (Στοίβα Ολίσθησης-Ελάττωσης) για το string cccddd είναι η ακόλουθη: aabb

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	cceddd EOF	Ολίσθηση
1	c	cceddd EOF	Ολίσθηση
2	cc	ccdd EOF	Ολίσθηση
3	ccc	ccdd EOF	Ελάττωση $\langle B \rangle ::= \epsilon$
4	ccc 	ccdd EOF	Ολίσθηση

5	ccd	dd EOF	Ελάττωση $::= cd$
6	cc 	dd EOF	Ολίσθηση
7	ccd	d EOF	Ελάττωση $::= cd$
8	c 	d EOF	Ολίσθηση
9	cd	EOF	Ελάττωση $::= cd$
10		EOF	Ελάττωση $<X> ::= $
11	<X>	EOF	Αναγνώριση

(γ) Η top-down στοίβα συντακτικής ανάλυσης (Στοίβα Πρόβλεψης-Ταιριάσματος) για το string aabb είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<X>	aabb EOF	Πρόβλεψη $<X> ::= <A>$
1	<A>	aabb EOF	Πρόβλεψη $<A> ::= a<A>b$
2	a<A>b_a	aabb EOF	Ταίριασμα συμβόλου a
3	<A>b	abb EOF	Πρόβλεψη $<A> ::= a<A>b$
4	a<A>b_a	abb EOF	Ταίριασμα συμβόλου a
5	<A>b_b	b EOF	Ταίριασμα συμβόλου b
6	<A>b	b EOF	Ταίριασμα συμβόλου b
7	<A>	EOF	Πρόβλεψη $<A> ::= \epsilon$
8	ϵ	EOF	Αναγνώριση

3.5.4 Θέμα 3 Σεπτέμβριος 2013 και Σεπτέμβριος 2022

Δίνεται η παρακάτω γραμματική για το αλφάριθμο a, b, c:

$<A> ::= <A> a \mid <A> b \mid $

$::= c$

1. Περιγράψτε τι συμβολοσειρές παράγει η γραμματική αυτή. Τι ιδιότητες έχουν; Ποιο το ελάχιστο μέγεθος συμβολοσειράς;
2. Να ελεγχθεί αν η συμβολοσειρά **cacab** ανήκει στη γλώσσα αυτή κατασκευάζοντας το δέντρο συντακτικής ανάλυσης (Parse Tree) για αυτή.
3. Είναι η συμβολοσειρά **cacac** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη **Στοίβα Ολίσθησης-Ελάττωσης** για τη συμβολοσειρά.
4. Τι μειονεκτήματα έχει η γραμματική αυτή; Πως μπορούμε να τη βελτιώσουμε;

Απάντηση

1.

Ενδεικτικές Παραγωγές

$A \Rightarrow B \Rightarrow c$

$A \Rightarrow AaA \Rightarrow BaB \Rightarrow cac$

$A \Rightarrow AbA \Rightarrow BbB \Rightarrow cbc$

$A \Rightarrow AaA \Rightarrow AaAaAbA \Rightarrow BaBaBbB \Rightarrow cacacbc$

$A \Rightarrow AaA \Rightarrow AaAaB \Rightarrow BaBaB \Rightarrow cacac$

$A \Rightarrow AbA \Rightarrow AbAbAbA \Rightarrow AaAbAaAbAbAbAaA \Rightarrow BaBbBaBbBbBbBaB \Rightarrow cacbcacbcacbcac$

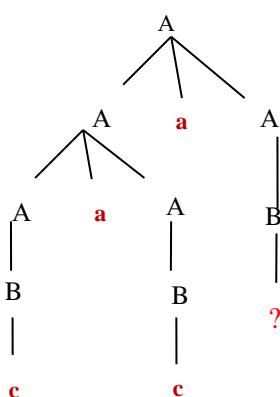
$A \Rightarrow AaA \Rightarrow AaAbA \Rightarrow BaBbA \Rightarrow cacbc$

Χαρακτηριστικά Συμβολοσειρών

- ✓ περιττού μήκους cacbc
- ✓ αρχίζουν και τελειώνουν με c
- ✓ στις περιττές θέσεις κάθε συμβολοσειράς εμφανίζεται ο χαρακτήρας c
- ✓ έχουν διαδοχική εναλλαγή χαρακτήρων
- ✓ ανάμεσα στα c παρεμβάλλονται a ή b

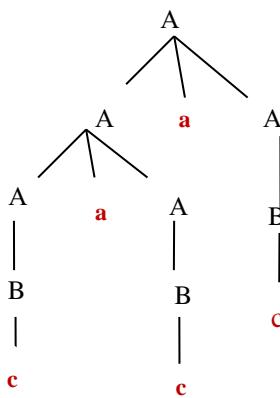
2.

Για να ανήκει η συμβολοσειρά **cacab** στη γλώσσα θα πρέπει να μπορεί να κατασκευαστεί το δέντρο συντακτικής ανάλυσης (Parse Tree) γιαυτή.



Παρατηρούμε πως δε μπορούμε να κατασκευάσουμε parse tree για τη συμβολοσειρά **cacab**, άρα δεν ανήκει στη γλώσσα. Αυτό είναι αναμενόμενο διότι δεν τελειώνει σε c αλλά το parse tree είναι η τυπική απόδειξη γιαυτό.

3. Η Στοίβα Ολίσθησης-Ελάττωσης για τη συμβολοσειρά **cacac** είναι η ακόλουθη:



ΒΗΜΑ	ΣΤΟΙΒΑ	ΕΙΣΟΔΟΣ	ΠΡΑΞΗ
0	ϵ	cacac EOF	Ολίσθηση
1	c	acac EOF	Ελάττωση $\langle B \rangle ::= c$
2	$\langle B \rangle$	acac EOF	Ελάττωση $\langle A \rangle ::= \langle B \rangle$
3	$\langle A \rangle$	acac EOF	Ολίσθηση
4	$\langle A \rangle a$	cac EOF	Ολίσθηση
5	$\langle A \rangle ac$	ac EOF	Ελάττωση $\langle B \rangle ::= c$
6	$\langle A \rangle a \langle B \rangle$	ac EOF	Ελάττωση $\langle A \rangle ::= \langle B \rangle$
7	$\langle A \rangle a \langle A \rangle$	ac EOF	Ελάττωση $\langle A \rangle ::= \langle A \rangle a \langle A \rangle$
8	$\langle A \rangle$	ac EOF	Ολίσθηση
9	$\langle A \rangle a$	c EOF	Ολίσθηση
10	$\langle A \rangle c$	EOF	Ελάττωση $\langle B \rangle ::= c$
11	$\langle A \rangle c \langle B \rangle$	EOF	Ελάττωση $\langle A \rangle ::= \langle B \rangle$
12	$\langle A \rangle c \langle A \rangle$	EOF	Ελάττωση $\langle A \rangle ::= \langle A \rangle a \langle A \rangle$
13	$\langle A \rangle$	EOF	Αναγνώριση

Παρατήρηση

Όταν η γραμματική δεν είναι LL/1 όπως αυτή, μπορούμε να βάλουμε στη στοίβα ολίσθησης το αρχικό σύμβολο της γραμματικής έστω και αν η συμβολοσειρά εισόδου δεν έχει διαβαστεί πλήρως.

4. Η γραμματική που δίνεται είναι η εξής:

$\langle A \rangle ::= \langle A \rangle a \langle A \rangle \mid \langle A \rangle b \langle A \rangle \mid \langle B \rangle$

$\langle B \rangle ::= c$

Παρατηρούμε πως η γραμματική αυτή παρουσιάζει αριστερή αναδρομή συνεπώς δεν είναι LL(1). Μπορούμε να τη διορθώσουμε δηλ. να τη μετατρέψουμε σε LL/1 κάνοντας **απαλοιφή αριστερής αναδρομής**.

Μεθοδολογία για Απαλοιφή Αριστερής Αναδρομής

1. Πηγαίνουμε σε κάθε κανόνα που ΔΕΝ έχει αριστερή αναδρομή και βάζουμε στο τέλος του ένα νέο ΜΗ τερματικό σύμβολο
2. Από το νέο ΜΗ τερματικό σύμβολο πηγαίνουμε σε κάθε κανόνα που είχε αριστερή αναδρομή, γράφουμε ότι υπάρχει μετά την αναδρομή ακολουθούμενο από το νέο σύμβολο και προσθέτουμε ένα κανόνα που οδηγεί σε κενό

Η διορθωμένη LL/1 γραμματική είναι η ακόλουθη:

$\langle A \rangle ::= \langle B \rangle \langle K \rangle$

$\langle K \rangle ::= a \langle A \rangle \langle K \rangle \mid b \langle A \rangle \langle K \rangle \mid \epsilon$

$\langle B \rangle ::= c$

3.5.5 Θέμα 5 Σεπτέμβρης 2010

Δίνεται η ακόλουθη γραμματική:

$<S> ::= b<F> \mid a<T>$

$<T> ::= a<S> \mid a$

$<F> ::= b<S> \mid b$

(α) Περιγράψτε τα χαρακτηριστικά των μελών της γλώσσας που παράγεται από την παραπάνω γραμματική

(β) Είναι το string **bbbaabb** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη στοίβα **Ολίσθησης-Ελάττωσης** (Bottom-Up Στοίβα) για το string

(γ) Η παραπάνω γραμματική δεν είναι LL(1). Εξηγήστε γιατί. Μετασχηματίστε τη σε μια ισοδύναμη LL(1) αναφέροντας τους αντίστοιχους μετασχηματισμούς.

(δ) Είναι το **aaabbba** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη στοίβα συντακτικής ανάλυσης **Ταιριάσματος-Πρόβλεψης** (Top-Down Στοίβα) για το string χρησιμοποιώντας τη γραμματική του ερωτήματος (γ)

Απάντηση

(α)

Εγδεικτικές Παραγωγές

$S \Rightarrow aT \Rightarrow aa$

$S \Rightarrow bF \Rightarrow bb$

$S \Rightarrow aT \Rightarrow aaS \Rightarrow aaaT \Rightarrow aaaa$

$S \Rightarrow bF \Rightarrow bbS \Rightarrow bbbF \Rightarrow bbbb$

$S \Rightarrow aT \Rightarrow aaS \Rightarrow aabF \Rightarrow aabb$

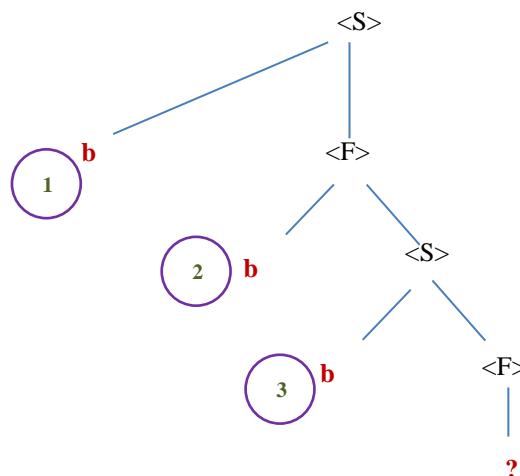
$S \Rightarrow bF \Rightarrow bbS \Rightarrow bbaT \Rightarrow bbaa$

$S \Rightarrow aT \Rightarrow aaS \Rightarrow aabF \Rightarrow aabbS \Rightarrow aabbaT \Rightarrow aabbbaa$

Χαρακτηριστικά μελών ης γλώσσας που παράγεται από την παραπάνω γραμματική

- Κατασκευάζει συμβολοσειρές άρτιου μήκους
- Ελάχιστο μήκος συμβολοσειρών τουλάχιστον 2 χαρακτήρες
- Αν οι συμβολοσειρές έχουν και a και b τότε θα έχουν τουλάχιστον δύο διαδοχικά a και τουλάχιστον δύο διαδοχικά b
- Αν οι συμβολοσειρές έχουν και a και b το πλήθος των a και το πλήθος των b είναι άρτιο

(β) To parse tree για τη συμβολοσειρά **bbbaabb** ΔΕΝ μπορεί να κατασκευαστεί διότι η συμβολοσειρά δεν ανήκει στη γλώσσα:



Η Bottom-up στοίβα συντακτικής ανάλυσης για τη ΜΗ αναγνώριση της συμβολοσειράς **bbbaabb**

Βίμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	bbbaabb EOF	Ολίσθηση 1
1	b	bbaabb EOF	Ολίσθηση 2
2	bb	baabb EOF	Ολίσθηση 3
3	bbb	aabb EOF	ΜΗ ΑΝΑΓΝΩΡΙΣΗ

(γ) Η γραμματική δεν είναι LL/1 διότι ο κανόνας $<T> ::= a<S>$ | a έχει δύο εναλλακτικούς κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο που είναι το a και ομοίως ο κανόνας $<F> ::= b<S>$ | b έχει δύο εναλλακτικούς κανόνες που αρχίζουν επίσης με το ίδιο τερματικό σύμβολο που είναι το b .

Παραγοντοποίηση

1. Βγάζουμε ως κοινό παράγοντα το τερματικό σύμβολο από κάθε κανόνα στον οποίο εμφανίζεται και προσθέτουμε μετά τον κοινό παράγοντα ένα νέο ΜΗ τερματικό σύμβολο

2. Από το νέο ΜΗ τερματικό σύμβολο πηγαίνουμε σε κάθε κανόνα που είχε κοινό παράγοντα και γράφουμε ότι υπάρχει ΜΕΤΑ τον κοινό παράγοντα. Αν δεν υπάρχει κάτι μετά τον κοινό παράγοντα βάζουμε το ϵ

Η διόρθωση του κανόνα $<T> ::= a<S>$ | a είναι η ακόλουθη:

$<T> ::= a<K>$

$<K> ::= <S> \mid \epsilon$

Η διόρθωση του κανόνα $<F> ::= b<S>$ | b είναι η ακόλουθη:

$<F> ::= b<M>$

$<M> ::= <S> \mid \epsilon$

Η τελική διορθωμένη LL/1 γραμματική είναι:

```

<S> ::= b<F> | a<T>
<T> ::= a<K>
<K> ::= <S> | ε
<F> ::= b<M>
<M> ::= <S> | ε
    
```

(δ)

Παρατήρηση: Αν μας ζητούσε να κατασκευάσουμε τον Top-Down Parser για κάποια συμβολοσειρά της αρχικής γραμματικής τότε αυτό θα ήταν αδύνατο διότι στην αρχική γραμματική δεν υπάρχει κανόνας που να οδηγεί στο ϵ (**ΕΙΝΑΙ ΑΠΑΡΑΙΤΗΤΟ ΝΑ ΥΠΑΡΧΕΙ ΣΤΗ ΓΡΑΜΜΑΤΙΚΗ ΕΝΑΣ ΤΟΥΛΑΧΙΣΤΟΝ ΚΑΝΟΝΑΣ ΠΟΥ ΝΑ ΟΔΗΓΕΙ ΣΕ ϵ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΤΟΥ TOP-DOWN PARSER**). Γιαυτό ζητάει να χρησιμοποιήσουμε τη γραμματική του ερωτήματος c η οποία είναι LL/1 και έχει κανόνες που οδηγεί στο ϵ .

Παρακάτω δίνονται τα βήματα του Top-Down Parser για τη ΜΗ αναγνώριση της συμβολοσειράς aaabbba

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	aaabbba EOF	Πρόβλεψη $<S> ::= a<T>$
1	a<T> a	aaabbba EOF	Ταίριασμα Συμβόλου a
2	<T>	aabbba EOF	Πρόβλεψη $<T> ::= a<K>$
3	a<K> a	aabbba EOF	Ταίριασμα Συμβόλου a
4	<K>	abbba EOF	Πρόβλεψη $<K> ::= <S>$
5	<S>	abbba EOF	Πρόβλεψη $<S> ::= a<T>$
6	a <T> a	abbba EOF	Ταίριασμα Συμβόλου a
7	<T>	bbba EOF	Πρόβλεψη $<T> ::= a<K>$
8	a<K>	bbba EOF	Πρόβλεψη $<K> ::= <S>$
9	a<S>	bbba EOF	Πρόβλεψη $<S> ::= b<F>$
10	ab<F>	bbba EOF	Πρόβλεψη $<F> ::= b<M>$
11	abb<M>	bbba EOF	Πρόβλεψη $<M> ::= <S>$
12	abb<S>	bbba EOF	Πρόβλεψη $<S> ::= b<F>$
13	abbb<F>	bbba EOF	Πρόβλεψη $<F> ::= b<M>$
14	abbbb<M>	bbba EOF	Πρόβλεψη $<M> ::= \epsilon$
15	abbb b	bbba EOF	Ταίριασμα Συμβόλου b
16	abb b	bba EOF	Ταίριασμα Συμβόλου b
17	ab b	ba EOF	Ταίριασμα Συμβόλου b
18	aba	a EOF	Ταίριασμα Συμβόλου a
19	b	EOF	ΜΗ ΑΝΑΓΝΩΡΙΣΗ

Δικό μου Παράδειγμα

Να κατασκευαστεί Top-Down Parser για την αναγνώριση της συμβολοσειράς **aabb**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	aabbEOF	Πρόβλεψη $\langle S \rangle ::= a \langle T \rangle$
1	<T>a	abbEOF	Ταίριασμα
2	<T>	abbEOF	Πρόβλεψη $\langle T \rangle ::= a \langle K \rangle$
3	<K>a	abbEOF	Ταίριασμα
4	<K>	bbEOF	Πρόβλεψη $\langle K \rangle ::= \langle S \rangle$
5	<S>	bbEOF	Πρόβλεψη $\langle S \rangle ::= b \langle F \rangle$
6	<F>b	bbEOF	Ταίριασμα
7	<F>	bEOF	Πρόβλεψη $\langle F \rangle ::= b \langle M \rangle$
8	<M>b	bEOF	Ταίριασμα
9	<Γ>	EOF	Πρόβλεψη $\langle Y \rangle ::= \epsilon$
10	ϵ	EOF	Αναγνώριση

3.5.6 Θέμα 5 Ιούνιος 2012

- (a) Παρουσιάστε μια BNF γραμματική η οποία παράγει τους δυαδικούς αριθμούς που αποτελούνται από n 1 ακολουθούμενα από 3n 0, δηλαδή είναι της μορφής 1....10....0 όπου $n \geq 0$ και το πλήθος των 0 είναι τριπλάσιο από το πλήθος των 1
- (b) Με βάση τη γραμματική σας δείξτε ότι ο αριθμός **1000** είναι μέλος της γλώσσας παρουσιάζοντας τη **Στοίβα Ολίσθησης-Ελάττωσης Bottom-Up** συντακτικής ανάλυσης
- (c) Επίσης δείξτε ότι ο αριθμός **11000000** είναι μέλος της γλώσσας παρουσιάζοντας τη **Στοίβα Ταιριάσματος-Πρόβλεψης Top-Down** συντακτικής ανάλυσης (αν η γραμματική σας δεν είναι LL(1) να κάνετε τους αναγκαίους μετασχηματισμούς)

Απάντηση

(a) Η BNF γραμματική η οποία παράγει τους δυαδικούς αριθμούς είναι: $\langle S \rangle ::= 1 \langle S \rangle 000 \mid \epsilon$

(b) **Στοίβα Ολίσθησης-Ελάττωσης** για την αναγνώριση του string **1000**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	1000 EOF	Ολίσθηση
1	1	000 EOF	Ελάττωση $\langle S \rangle ::= \epsilon$
2	$1 \langle S \rangle$	000 EOF	Ολίσθηση
3	$1 \langle S \rangle 0$	00 EOF	Ολίσθηση
4	$1 \langle S \rangle 00$	0 EOF	Ολίσθηση
5	$1 \langle S \rangle 000$	EOF	Ελάττωση $\langle S \rangle ::= 1 \langle S \rangle 000$
6	$\langle S \rangle$	EOF	Αναγνώριση

(c) **Στοίβα Ταιριάσματος-Πρόβλεψης** για την αναγνώριση του string **11000000**. Η Γραμματική μας είναι LL(1)

Βήμα	Στοίβα	Είσοδος	Πράξη
0	$\langle S \rangle$	11000000 EOF	Πρόβλεψη $\langle S \rangle ::= 1 \langle S \rangle 000$
1	$1 \langle S \rangle 000$	11000000 EOF	Πρόβλεψη $\langle S \rangle ::= 1 \langle S \rangle 000$
2	11 $\langle S \rangle 0000001$	11000000 EOF	Ταίριασμα συμβόλου 1
3	11 $\langle S \rangle 0000001$	1000000 EOF	Ταίριασμα συμβόλου 1
4	$\langle S \rangle 000000$ 0	000000 EOF	Ταίριασμα συμβόλου 0
5	$\langle S \rangle 00000$ 0	00000 EOF	Ταίριασμα συμβόλου 0
6	$\langle S \rangle 0000$ 0	0000 EOF	Ταίριασμα συμβόλου 0
7	$\langle S \rangle 000$ 0	000 EOF	Ταίριασμα συμβόλου 0
8	$\langle S \rangle 00$ 0	00 EOF	Ταίριασμα συμβόλου 0
9	$\langle S \rangle 0$ 0	0 EOF	Ταίριασμα συμβόλου 0
10	$\langle S \rangle$	EOF	Πρόβλεψη $\langle S \rangle ::= \epsilon$
11	ϵ	EOF	Αναγνώριση

3.5.7 Θέμα 2 Ιούνιος 2014

Δίνεται η ακόλουθη BNF γραμματική:

$\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle \mid d$

$\langle A \rangle ::= a \langle A \rangle \mid \epsilon$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

$\langle C \rangle ::= c$

α) Ποια γλώσσα περιγράφει; Δώστε παράδειγμα

β) Είναι LL(1); Αν όχι να τη μετατρέψτε σε LL(1)

γ) Να εξετάσετε αν η συμβολοσειρά **bcbabc** ανήκει στη γλώσσα με χρήση του top-down parser

δ) Να εξετάσετε αν η συμβολοσειρά **bacab** ανήκει στη γλώσσα με χρήση του bottom-up parser

Απάντηση

α) Για να βρούμε τη γλώσσα που περιγράφει η γραμματική κάνουμε παραγωγές στη γραμματική

Ενδεικτικές Παραγωγές

$S \Rightarrow d$

$S \Rightarrow ABCS \Rightarrow cd$

$S \Rightarrow ABCS \Rightarrow aAcd \Rightarrow acd$

$S \Rightarrow ABCS \Rightarrow bBcd \Rightarrow bcd$

$S \Rightarrow ABCS \Rightarrow aAbBcd \Rightarrow abcd$

$S \Rightarrow ABCS \Rightarrow aAbBcABCS \Rightarrow abcaAbBcd \Rightarrow abcabcd$

$S \Rightarrow ABCS \Rightarrow ABCABCABC \Rightarrow ABCABCABC \Rightarrow aAbBcaAbBcaAbBcd \Rightarrow abcabcabcd$

$S \Rightarrow ABCS \Rightarrow aAbBcd \Rightarrow aaAbbBcd \Rightarrow aaaAbbcd \Rightarrow aaabbcd$

Από τις προηγούμενες παραγωγές καταλήγουμε στο συμπέρασμα ότι η γραμματική περιγράφει τη γλώσσα με τα εξής χαρακτηριστικά:

Χαρακτηριστικά Συμβολοσειρών Γλώσσας που παράγεται από τη γραμματική

- Ο χαρακτήρας **d** εμφανίζεται μόνο 1 φορά στο τέλος της κάθε συμβολοσειράς δηλ. κάθε συμβολοσειρά τελειώνει σε **d**
- Οι συμβολοσειρές έχουν μήκος τουλάχιστον 1 χαρακτήρα δηλ.. απαγορεύεται η κενή συμβολοσειρά
- Αμέσως (ακριβώς) μετά το χαρακτήρα **b** δεν μπορεί να υπάρχει χαρακτήρας **a**

β) Είναι LL(1) διότι α) ΔΕΝ έχει αριστερή αναδρομή και β) ΔΕΝ χρειάζεται παραγοντοποίηση:

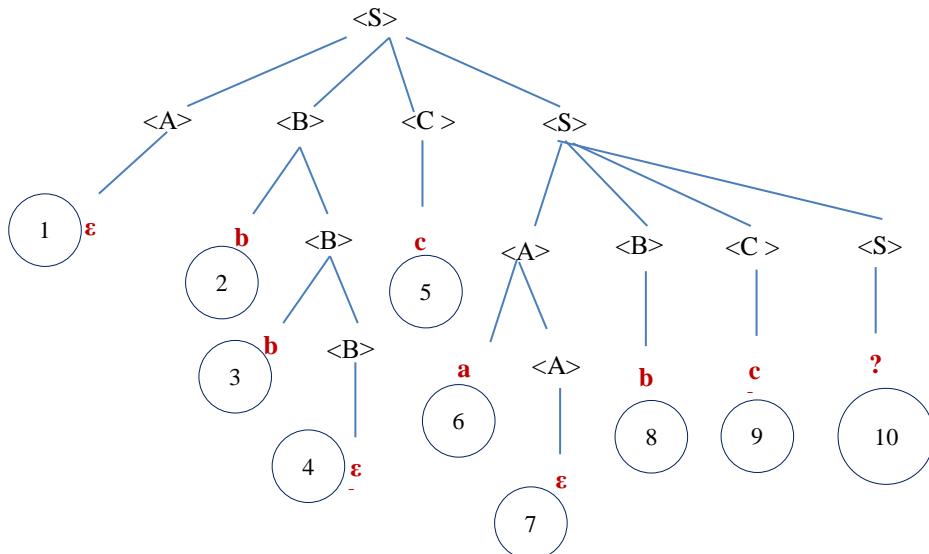
$\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle \mid d$

$\langle A \rangle ::= a \langle A \rangle \mid \epsilon$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

$\langle C \rangle ::= c$

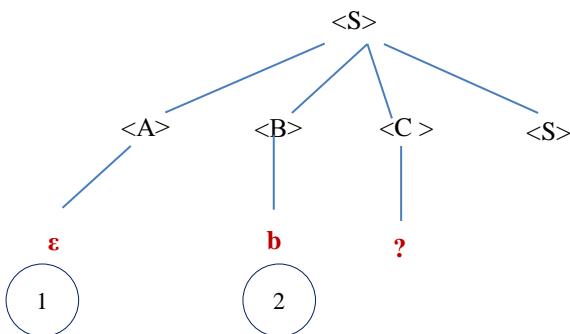
γ) Παρακάτω δίνονται τα βήματα του **Top-Down Parser** (Στοίβα Πρόβλεψης-Ταιριάσματος) για τη ΜΗ αναγνώριση της συμβολοσειράς **babcabc**



Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	babcabc EOF	Πρόβλεψη $\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle$
1	<S><C><A>	babcabc EOF	Πρόβλεψη $\langle A \rangle ::= \epsilon$
2	<S><C>	babcabc EOF	Πρόβλεψη $\langle B \rangle ::= b \langle B \rangle$
3	<S><C>b	babc EOF	Ταίριασμα Συμβόλου b
4	<S><C>	cabc EOF	Πρόβλεψη $\langle B \rangle ::= b \langle B \rangle$
5	<S><C>b	bc EOF	Ταίριασμα Συμβόλου b
6	<S><C>	abc EOF	Πρόβλεψη $\langle B \rangle ::= \epsilon$
7	<S><C>	abc EOF	Πρόβλεψη $\langle C \rangle ::= c$
8	<S>c	cabc EOF	Ταίριασμα Συμβόλου c
9	<S>	abc EOF	Πρόβλεψη $\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle$
10	<S><C><A>	ab EOF	Πρόβλεψη $\langle A \rangle ::= a \langle A \rangle$
11	<S><C><A>a	bc EOF	Ταίριασμα Συμβόλου a
12	<S><C><A>	bc EOF	Πρόβλεψη $\langle A \rangle ::= \epsilon$
13	<S><C>	bc EOF	Πρόβλεψη $\langle B \rangle ::= b$

14	$<S><C>b$	bc EOF	Ταίριασμα Συμβόλου b 8
15	$<S><C>$	c EOF	Πρόβλεψη $<C> ::= c$
16	$<S>c$	c EOF	Ταίριασμα Συμβόλου c 9
17	$<S>$	EOF	ΜΗ ΑΝΑΓΝΩΡΙΣΗ

δ) Παρακάτω δίνονται τα βήματα του Bottom-UP Parser (Στοίβα Ολίσθησης - Ελάττωσης) για τη ΜΗ αναγνώριση της συμβολοσειράς **bacad**



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	bacad EOF	Ελάττωση $<A> ::= \epsilon$ 1
1	$<A>$	bacad EOF	Ολίσθηση
2	$<A>b$	acad EOF	Ελάττωση $::= b$ 2
3	$<A>$	acad EOF	ΜΗ ΑΝΑΓΝΩΡΙΣΗ

3.5.8 Θέμα 3 Ιούνιος 2015

Δίνεται η παρακάτω BNF γραμματική με **τερματικά σύμβολα τα (,), w, x, y, z**

$\langle A \rangle ::= (\langle B \rangle)$

$\langle B \rangle ::= \langle C \rangle \mid \langle B \rangle, \langle C \rangle$

$\langle C \rangle ::= \langle A \rangle \mid \langle D \rangle$

$\langle D \rangle ::= w \mid x \mid y \mid z$

(a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε μερικά βασικά χαρακτηριστικά των συμβολοσειρών αυτών παρουσιάζοντας και παραδείγματα

(b) Ελέγξτε αν μια νόμιμη συμβολοσειρά με δύο ζευγάρια παρενθέσεων και δύο κόμματα (όχι και τα δύο μέσα στο ίδιο ζευγάρι παρενθέσεων) είναι μέλος της γλώσσας που υποστηρίζει η γραμματική, παρουσιάζοντας τη **Στοίβα Ολίσθησης-Ελάττωσης Bottom-Up** συντακτικής ανάλυσης

(c) Είναι η γραμματική LL(1); Αιτιολογήστε την απάντηση σας. Αν η γραμματική δεν είναι LL(1), κάνετε τους αναγκαίους μετασχηματισμούς και παρουσιάστε την ισοδύναμη LL(1) γραμματική.

Απάντηση

(a)

Ενδεικτικές Παραγωγές

Προσπαθούμε να εξαλείψουμε όλα τα μη τερματικά σύμβολα και να κατασκευάσουμε συμβολοσειρές που αποτελούνται μόνο από τερματικά σύμβολα.

$A \Rightarrow (B) \Rightarrow (C) \Rightarrow (D) \Rightarrow (\text{w})$

$A \Rightarrow (B) \Rightarrow (C) \Rightarrow (A) \Rightarrow ((B)) \Rightarrow ((C)) \Rightarrow ((D)) \Rightarrow (\text{(x)})$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (C, C) \Rightarrow (\text{x}, \text{y})$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (C, C) \Rightarrow (A, A) \Rightarrow ((B), (B)) \Rightarrow ((C), (C)) \Rightarrow ((A), (A)) \Rightarrow (\text{(w)}, \text{(x)})$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (C, C) \Rightarrow (A, A) \Rightarrow ((B), (B)) \Rightarrow ((C), (B, C)) \Rightarrow ((C), (C, C)) \Rightarrow ((D), (A, D)) \Rightarrow ((D), ((B), D)) \Rightarrow ((D), ((C), D))$

$\Rightarrow ((D), ((D), D)) \Rightarrow (\text{(w)}, \text{(x)}, \text{y}))$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (C, A) \Rightarrow (D, (B)) \Rightarrow (D, (B, C)) \Rightarrow (D, (C, C)) \Rightarrow (D, (D, D)) \Rightarrow (\text{w}, \text{(x)}, \text{y})$

Χαρακτηριστικά Συμβολοσειρών

- Η γραμματική αυτή παράγει συμβολοσειρές από τα τερματικά σύμβολα w, x, y, z μέσα σε παρενθέσεις σωστά εμφωλευμένες
- Αν υπάρχουν εμφωλευμένες παρενθέσεις διαχωρίζονται μεταξύ τους με κόμμα και μπορεί να υπάρχει οποιοσδήποτε βαθμός εμφώλευσης των παρενθέσεων
- Οι παρενθέσεις περιλαμβάνουν τα τερματικά σύμβολα w, x, y, z διαχωρίζονται με κόμμα είτε με κόμμα και άλλες εμφωλευμένες παρενθέσεις

(b) Επιλέγουμε τη συμβολοσειρά $(w, (x, y))$ η οποία είναι νόμιμη και έχει δύο ζευγάρια παρενθέσεων και δύο κόμματα (δεν είναι και τα δυο μέσα στο ίδιο ζευγάρι παρενθέσεων όπως ζητείται). Για να δούμε αν είναι μέλος της γλώσσας παρουσιάζουμε τη στοίβα Ολίσθησης-Ελάττωσης (Bottom-Up Parser) συντακτικής ανάλυσης.

Στοίβα Ολίσθησης-Ελάττωσης

Βίμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	$(w, (x, y))$ EOF	Ολίσθηση
1	($w, (x, y)$ EOF	Ολίσθηση
2	(w	, (x, y) EOF	Ελάττωση $<D> ::= w$
3	(<D>	, (x, y) EOF	Ολίσθηση
4	(<D>,	(x, y) EOF	Ολίσθηση
6	(<D>, (x,	y) EOF	Ολίσθηση
7	(<D>, (x,	y) EOF	Ολίσθηση
8	(<D>, (x,	, y) EOF	Ελάττωση $<D> ::= x$
9	(<D>, (<D>	,y) EOF	Ολίσθηση
10	(<D>, (<D>,	y) EOF	Ολίσθηση
11	(<D>, (<D>, y) EOF	Ελάττωση $<D> ::= y$
12	(<D>,(<D>, <D>) EOF	Ολίσθηση
13	(<D>, (<D>, <D>)) EOF	Ελάττωση $<C> ::= <D>$
14	(<C>, (<C>, <C>)) EOF	Ελάττωση $::= <C>$
15	(<C>, (, <C>)) EOF	Ελάττωση $::= , <C>$
16	(<C>, ()) EOF	Ελάττωση $<A> ::= ()$
17	(<C>, <A>) EOF	Ολίσθηση
18	(<C>,<A>)	EOF	Ελάττωση $<C> ::= <A>$
19	(<C>, <C>)	EOF	Ελάττωση $::= <C>$
20	(, <C>)	EOF	Ελάττωση $::= , <C>$
21	()	EOF	Ελάττωση $<A> ::= ()$
22	<A>	EOF	Αναγώριση

(c) Η γραμματική

$<A> ::= ()$

$::= <C> | , <C>$

$<C> ::= <A> | <D>$

$<D> ::= w | x | y | z$

δεν είναι LL(1) διότι στον κανόνα $::= , <C>$ υπάρχει αριστερή αναδρομή. Κάνουμε απαλοιφή της αριστερής αναδρομής και η LL(1) που προκύπτει είναι η ακόλουθη:

<A>::=()

::=<C> **<K>**

<K>::= , <C> **<K>** | ε

<C>::= <A> | <D>

<D>::= w | x | y | z

Επεξήγηση Διόρθωσης

1. Πηγαίνουμε σε κάθε κανόνα που ΔΕΝ έχει αριστερή αναδρομή και βάζουμε στο τέλος του ένα νέο ΜΗ τερματικό σύμβολο. Ο κανόνας ::=<C> Δεν έχει αριστερή αναδρομή. Στο τέλος του βάζω ως νέο ΜΗ τερματικό σύμβολο το **<K>**

2. Από το νέο ΜΗ τερματικό σύμβολο δηλ. από το **<K>** πηγαίνουμε σε κάθε κανόνα που έχει αριστερή αναδρομή, γράφουμε ότι υπάρχει μετά την αναδρομή ακολουθούμενο από το νέο σύμβολο ή κενό. Μετά την αναδρομή υπάρχει το , <C> άρα γράφουμε , <C> ακολουθούμενο από το νέο σύμβολο δηλ. το **<K>** και προσθέτουμε και ένα κανόνα με το κενό

3.5.9 Θέμα 3 Σεπτέμβριος 2015 και Αυτονομή 2016

Σας δίνεται η παρακάτω LL(1) BNF γραμματική:

```
<S> ::= int <κενό> id <B> | float <κενό> id <B>
<B> ::= , <κενό> id <B> | ;
<κενό> ::= ένας χαρακτήρας κενού
```

(a) Να περιγράψετε τη γλώσσα που παράγεται από τη γραμματική; Τι θα μπορούσε να περιγράψει στη γλώσσα C η παραπάνω γραμματική ως τμήμα της συνολικής γραμματικής της γλώσσας; Περιγράψτε μερικά παραδείγματα «νόδυμων» εκφράσεων της γλώσσας

(b) Είναι το **int mi, pi;** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη Στοίβα Ολίσθησης-Ελάττωσης Bottom-Up συντακτικής ανάλυσης για το string

(c) Είναι το **float t, xa, b** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη **Στοίβα Ταιριάσματος-Πρόβλεψης Top-Down** συντακτικής ανάλυσης για το string

Απάντηση

(a)

Ενδεικτικές Παραγωγές

$S \Rightarrow \text{int} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{int } | \text{id} ;$

$S \Rightarrow \text{float} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{float } | \text{id} ;$

$S \Rightarrow \text{int} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{int} <\text{κενό}> \text{id} , <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{int } <\text{κενό}> \text{id} , <\text{κενό}> \text{id} ;$

$S \Rightarrow \text{float} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{float } | \text{id} \text{ float} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{float } | \text{id} \text{ float } <\text{κενό}> \text{id} ;$

$S \Rightarrow \text{int} <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{int} <\text{κενό}> \text{mi}, <\text{κενό}> \text{id} <\text{B}> \Rightarrow \text{int } <\text{κενό}> \text{mi}, <\text{κενό}> \text{mi} ;$

Χαρακτηριστικά Συμβολοσειρών

- Η γλώσσα περιλαμβάνει όλες τις συμβολοσειρές που αποτελούν δήλωση μεταβλητών τύπου int ή float.
- Κάθε δήλωση περιλαμβάνει μια ή περισσότερες μεταβλητές (id's) που διαχωρίζονται μεταξύ τους με κόμμα και κενό και τελειώνει με το **χαρακτήρα** ;
- Η γραμματική αυτή θα μπορούσε να περιγράψει το τμήμα δηλώσεων μεταβλητών στη γλώσσα C.
- Βέβαια η γραμματική αυτή περιλαμβάνει δήλωση μεταβλητών μόνο τύπου int ή float και όχι άλλων τύπων όπως στη C

(b) Παρακάτω δίνονται τα βήματα του Bottom-UP Parser για την αναγνώριση της συμβολοσειράς **int <κενό> mi, <κενό> pi;**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	int mi, pi; EOF	Ολίσθηση
1	int	 mi, pi; EOF	Ολίσθηση
2	int	mi, pi; EOF	Ελάττωση $<\text{κενό}> ::= $
3	int <κενό>	mi, pi; EOF	Ολίσθηση
4	int <κενό> mi	 pi; EOF	Ολίσθηση
5	int <κενό> mi,	pi; EOF	Ολίσθηση
6	int <κενό> mi,	pi; EOF	Ελάττωση $<\text{κενό}> ::= $

7	int <κενό> mi, <κενό>	pi; EOF	Ολίσθηση
8	int <κενό> mi, <κενό> pi	; EOF	Ολίσθηση
9	int <κενό> mi, <κενό> pi;	EOF	Ελάττωση ::=;
10	int <κενό> mi , <κενό> pi 	EOF	::=, <κενό> id
11	int <κενό> <id>	EOF	Ελάττωση <S>::=int <κενό> id
12	<S>	EOF	Αναγνώριση

(c) Παρακάτω δίνονται τα βήματα του **Top-Down Parser** για τη ΜΗ αναγνώριση της συμβολοσειράς float |t, |xa, |b

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	float t, xa, b EOF	Πρόβλεψη <S>::=float <κενό> id
1	<κενό> id float	float t, xa, b EOF	Ταίριασμα συμβόλου float
2	id <κενό>	t, <κενό> xa, b EOF	Πρόβλεψη <κενό>::=
3	id 	t, <κενό> xa, b EOF	Ταίριασμα συμβόλου κενό
4	 id	t, <κενό> xa, <κενό>b EOF	Ταίριασμα συμβόλου id
5		, <κενό> xa, <κενό>b EOF	Πρόβλεψη ::=, <κενό> id
6	<κενό> id ,	, xa, b EOF	Ταίριασμα συμβόλου ,
7	id , <κενό>	xa b EOF	Πρόβλεψη <κενό>::=
8	id ,	xa b EOF	Ταίριασμα συμβόλου κενό
9	id	xa, <κενό>b EOF	Ταίριασμα συμβόλου id
10		, b EOF	Πρόβλεψη ::=, <κενό> id
9	, <κενό> id 	, b EOF	Ταίριασμα συμβόλου ,
10	id <κενό>	b EOF	Πρόβλεψη <κενό>::=
10	id 	b EOF	Ταίριασμα συμβόλου κενό
11	 id	b EOF	Ταίριασμα συμβόλου id
12		EOF	ΜΗ Αναγνώριση

3.5.10 Θέμα 3 Ιούνιος 2015 και Θέμα 3 Φεβρουάριος 2018

Δίνεται η παρακάτω BNF γραμματική με τερματικά σύμβολα τα a, b, ε

$\langle S \rangle ::= b \langle S \rangle | a \langle M \rangle$

$\langle M \rangle ::= b \langle S \rangle | a \langle F \rangle$

$\langle F \rangle ::= b \langle F \rangle | a \langle F \rangle | \epsilon$

(a) Τι τύπου είναι η γραμματική **στην ιεραρχία Chomsky**; Αιτιολογήστε την απάντηση σας

(b) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική;

(c) Ελέγξτε αν η συμβολοσειρά **baaabaa** είναι μέλος της γλώσσας, παρουσιάζοντας τη **Στοίβα Ολίσθησης-Ελάττωσης (Bottom-up) συντακτικής ανάλυσης**

Απάντηση

Η ιεραρχία Chomsky είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

**Γραμματικές Χωρίς
Συμφραζόμενα**

**Γραμματικές Με
Συμφραζόμενα**

**Γραμματικές χωρίς
περιορισμούς**

(α) Η γραμματική αυτή είναι **Κανονική γιατί στο δεξί μέρος κάθε κανόνα υπάρχει το πολύ 1 μη τερματικό σύμβολο (δηλ. είτε ένα είτε κανένα)** και αυτό είναι το τελευταίο σύμβολο του κανόνα

Εναλλακτικά η Γραμματική αυτή στην ιεραρχία Chomsky είναι κανονική διότι σε **όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.**

(β)

Ενδεικτικές Παραγωγές

$S \Rightarrow aM \Rightarrow aaF \Rightarrow aa$

$S \Rightarrow aM \Rightarrow aaF \Rightarrow aabF \Rightarrow aab$

$S \Rightarrow aM \Rightarrow aaF \Rightarrow aabF \Rightarrow aabbF \Rightarrow aabbaF \Rightarrow aabba$

$S \Rightarrow bS \Rightarrow bbS \Rightarrow bbaM \Rightarrow bbaaF \Rightarrow bbaabF \Rightarrow bbaabaF \Rightarrow bbaaba$

$S \Rightarrow bS \Rightarrow baaM \Rightarrow baabS \Rightarrow baabaM \Rightarrow baabaaF \Rightarrow baabaabF \Rightarrow baabaabaF \Rightarrow baabaaba$

Χαρακτηριστικά Συμβολοσειρών

- Μη κενές Συμβολοσειρές που περιέχουν τουλάχιστον δύο a
- Αν οι συμβολοσειρές περιέχουν και χαρακτήρα b τότε έχουν μήκος τουλάχιστον 3 χαρακτήρες και τουλάχιστον 2 χαρακτήρες a
- Αν οι συμβολοσειρές περιέχουν μόνο χαρακτήρες a τότε το μήκος τους είναι τουλάχιστον 2 χαρακτήρες

3. Παρακάτω δίνονται τα βήματα του Bottom-up Parser για την αναγνώριση της συμβόλοσειράς **baaaba**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	baaaba EOF	Ολίσθηση
1	b	aaaba EOF	Ολίσθηση
2	ba	aaba EOF	Ολίσθηση
3	baa	aba EOF	Ολίσθηση
4	baaa	ba EOF	Ολίσθηση
5	baaab	a EOF	Ολίσθηση
6	baaaba	EOF	Ελάττωση $<F> ::= \epsilon$
7	baaaba<F>	EOF	Ελάττωση $<F> ::= a <F>$
8	baaab<F>	EOF	Ελάττωση $<F> ::= b <F>$
9	aaa<F>	EOF	Ελάττωση $<F> ::= a <F>$
10	baa<F>	EOF	Ελάττωση $<M> ::= a <F>$
11	ba<M>	EOF	Ελάττωση $<S> ::= a <M>$
12	b<S>	EOF	Ελάττωση $<S> ::= b <S>$
13	<S>	EOF	Αναγνώριση

3.5.11 Θέμα 4 Ιούνιος 2017

Μια συνθηματική γλώσσα απλών προτάσεων περιγράφεται από την παρακάτω BNF γραμματική, με τερματικά σύμβολα τα **a b d S** όπου το **S** συμβολίζει το χαρακτήρα «κενό» (space)

```
<πρόταση> ::= <λέξη> | <πρόταση> S <λέξη>

<λέξη> ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή>

<συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | a <βάση> | a <σύμφωνο>

<βάση> ::= <σύμφωνο> a

<σύμφωνο> ::= b | d
```

- (a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε μερικά βασικά χαρακτηριστικά των συμβολοσειρών αυτών, παρουσιάζοντας και παραδείγματα
- (b) Παρουσιάζοντας τη **Στοίβα Ολίσθησης – Ελάττωσης (Bottom-Up)** Συντακτικής Ανάλυσης να αποφανθείτε αν η πρόταση **daSaba** είναι μέλος της συνθηματικής γλώσσας
- (c) Παρουσιάζοντας τη **Στοίβα Πρόβλεψης – Ταιριάσματος (Top-Down)** Συντακτικής Ανάλυσης να αποφανθείτε αν η πρόταση **baSbab** είναι μέλος της συνθηματικής γλώσσας
- (d) Είναι η γραμματική LL(1); Αιτιολογήστε την απάντηση σας, υποδεικνύοντας και τα σχετικά σημεία της γραμματικής που δικαιολογούν τον ισχυρισμό σας. Αν η γραμματική δεν είναι LL(1), τι μετασχηματισμοί πρέπει να γίνουν ώστε να δημιουργηθεί μια ισόδύναμη LL(1) γραμματική;

Απάντηση

(a)

Παραδείγματα Συμβολοσειρών

πρόταση ⇒ λέξη ⇒ συλλαβή ⇒ βάση ⇒ σύμφωνο **a** ⇒ ba **ń** da

πρόταση ⇒ λέξη ⇒ συλλαβή ⇒ **a** βάση ⇒ **a** σύμφωνο **a** ⇒ aba **ń** ada

πρόταση ⇒ λέξη ⇒ συλλαβή ⇒ βάση σύμφωνο ⇒ σύμφωνο **a** σύμφωνο ⇒ bab **ń** dad **ń** bad **ń** dab

πρόταση ⇒ λέξη ⇒ συλλαβή λέξη συλλαβή ⇒ **a** σύμφωνο συλλαβή **a** βάση ⇒ **ab** **a** σύμφωνο **a** σύμφωνο ⇒ abadaba **ń** abadada

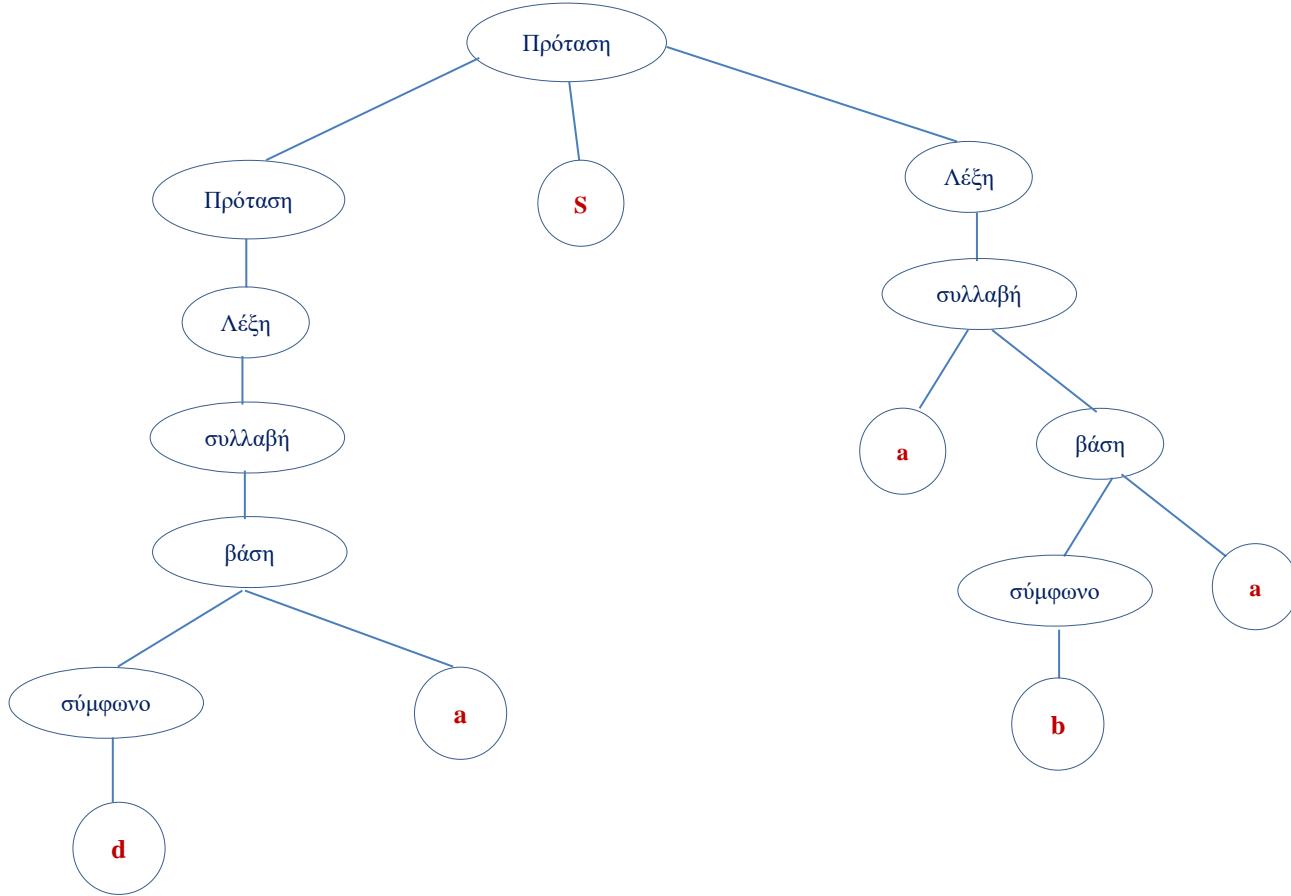
πρόταση ⇒ πρόταση **S** λέξη ⇒ λέξη **S** λέξη ⇒ συλλαβή **S** συλλαβή ⇒ βάση **S** βάση σύμφωνο ⇒ σύμφωνο **a** **S** σύμφωνο **a** **b** ⇒ baSbab

✓ Το βασικό χαρακτηριστικό των συμβολοσειρών της γλώσσας είναι ότι δεν επιτρέπεται να έχουμε συνεχόμενα **b** **ń** **d** **ń** **a** **ń** **S**.

Υπάρχει διαδοχική εναλλαγή χαρακτήρων

✓ Ελάχιστο μήκος συμβολοσειράς είναι οι 2 χαρακτήρες

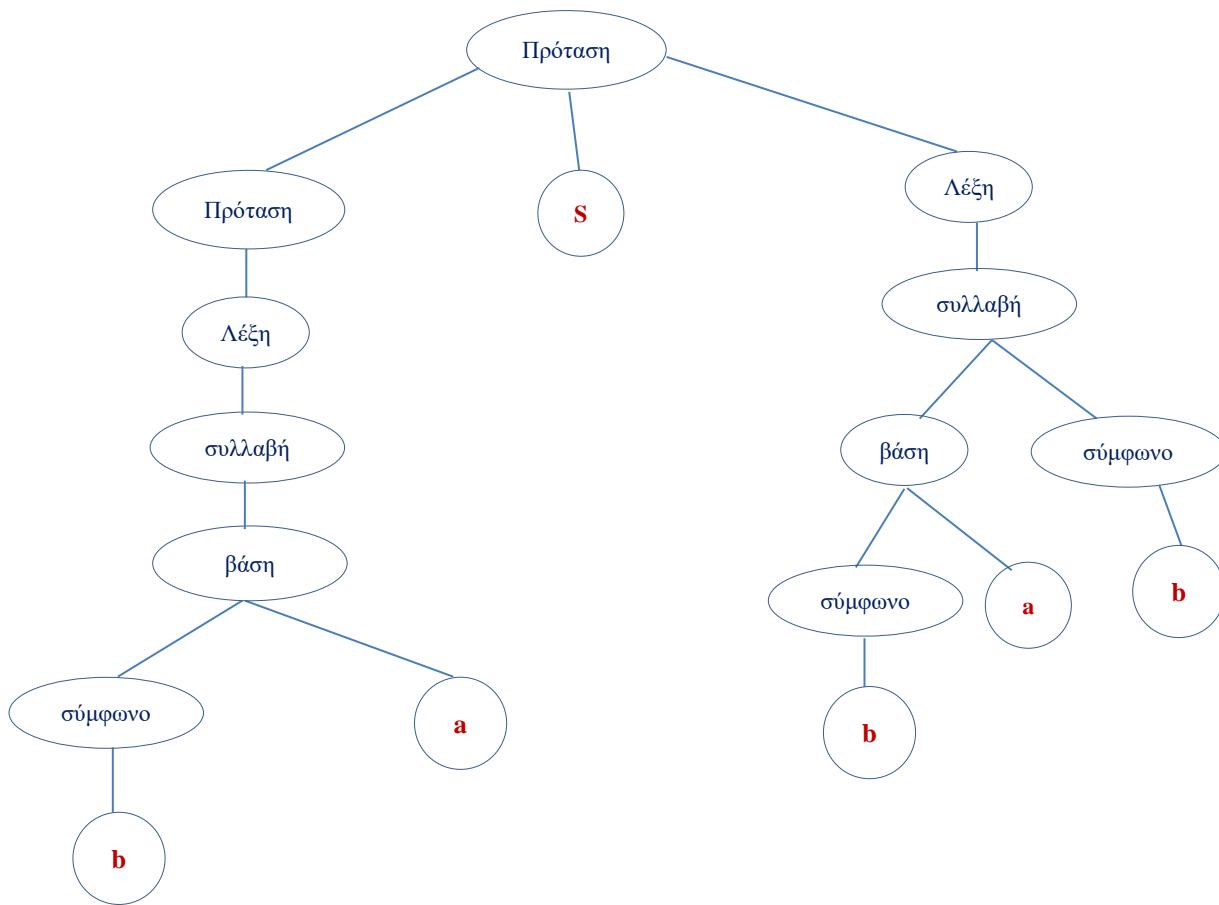
(b) Παρακάτω δίνονται τα βήματα του parse tree για τη συμβολοσειρά **daSaba**



Η Στοίβα Ολίσθησης – Ελάττωσης Bottom-UP Στοίβα Συντακτικής Ανάλυσης για την πρόταση **daSaba** είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ε	daSaba EOF	Ολίσθηση
1	d	aSaba EOF	Ελάττωση $\langle\text{σύμφωνο}\rangle ::= d$
2	$\langle\text{σύμφωνο}\rangle$	daSaba EOF	Ολίσθηση
3	$\langle\text{σύμφωνο}\rangle a$	Saba EOF	Ελάττωση $\langle\text{βάση}\rangle ::= \langle\text{σύμφωνο}\rangle a$
4	$\langle\text{βάση}\rangle$	Saba EOF	Ελάττωση $\langle\text{συλλαβή}\rangle ::= \langle\text{βάση}\rangle$
5	$\langle\text{συλλαβή}\rangle$	Saba EOF	Ελάττωση $\langle\text{λέξη}\rangle ::= \langle\text{συλλαβή}\rangle$
6	$\langle\text{λέξη}\rangle$	Saba EOF	Ελάττωση $\langle\text{πρόταση}\rangle ::= \langle\text{λέξη}\rangle$
7	$\langle\text{πρόταση}\rangle$	Saba EOF	Ολίσθηση
8	$\langle\text{πρόταση}\rangle S$	aba EOF	Ολίσθηση
9	$\langle\text{πρόταση}\rangle Sa$	ba EOF	Ολίσθηση
9	$\langle\text{πρόταση}\rangle Sab$	a EOF	Ελάττωση $\langle\text{σύμφωνο}\rangle ::= b$
11	$\langle\text{πρόταση}\rangle Saba <\text{σύμφωνο}>$	a EOF	Ολίσθηση
12	$\langle\text{πρόταση}\rangle Saba <\text{σύμφωνο}> a$	EOF	Ελάττωση $\langle\text{βάση}\rangle ::= \langle\text{σύμφωνο}\rangle a$
13	$\langle\text{πρόταση}\rangle Saba <\text{βάση}\rangle$	EOF	Ελάττωση $\langle\text{συλλαβή}\rangle ::= a <\text{βάση}\rangle$
14	$\langle\text{πρόταση}\rangle Sab <\text{συλλαβή}\rangle$	EOF	Ελάττωση $\langle\text{λέξη}\rangle ::= \langle\text{συλλαβή}\rangle$
15	$\langle\text{πρόταση}\rangle Sab <\text{λέξη}\rangle$	EOF	Ελάττωση $\langle\text{πρόταση}\rangle ::= \langle\text{πρόταση}\rangle S <\text{λέξη}\rangle$
16	$\langle\text{πρόταση}\rangle$	EOF	Αναγνώριση

(c) Παρακάτω δίνονται τα βήματα του parse tree για τη συμβολοσειρά **baSbab**



Η Στοίβα Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης για τη συμβολοσειρά **baSbab** είναι:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<Πρόταση>	baSbab EOF	Πρόβλεψη <πρόταση> ::= <πρόταση> S <λέξη>
1	<λέξη> S <πρόταση>	baSbab EOF	Πρόβλεψη <πρόταση> ::= <λέξη>
2	<λέξη> S <λέξη>	baSbab EOF	Πρόβλεψη <λέξη> ::= <συλλαβή>
3	<λέξη> S <συλλαβή>	baSbab EOF	Πρόβλεψη <συλλαβή> ::= <βάση>
4	<λέξη> S <βάση>	baSbab EOF	Πρόβλεψη <βάση> ::= <σύμφωνο> a
5	<λέξη> S a <σύμφωνο>	baSbab EOF	Πρόβλεψη <σύμφωνο> ::= b
6	<λέξη> S a <u>b</u>	baSbab EOF	Ταίριασμα Συμβόλου b
7	<λέξη> S <u>a</u>	aSbab EOF	Ταίριασμα Συμβόλου a
8	<λέξη> S	Sbab EOF	Ταίριασμα Συμβόλου S
9	<λέξη>	bab EOF	Πρόβλεψη <λέξη> ::= <συλλαβή>
10	<συλλαβή>	bab EOF	Πρόβλεψη <συλλαβή> ::= <βάση> <σύμφωνο>
11	<σύμφωνο><βάση>	bab EOF	Πρόβλεψη <βάση> ::= <σύμφωνο> a

12	<σύμφωνο> a <σύμφωνο>	bab EOF	Πρόβλεψη < σύμφωνο >::= b
13	<σύμφωνο> a <u>b</u>	bab EOF	Ταίριασμα Συμβόλου b
14	<σύμφωνο> <u>a</u>	<u>a</u>b EOF	Ταίριασμα Συμβόλου a
15	<σύμφωνο>	b EOF	Πρόβλεψη < σύμφωνο >::= b
16	<u>b</u>	b EOF	Ταίριασμα Συμβόλου b
17	<u>ε</u>	EOF	Αναγνώριση

(d) Η γραμματική δεν είναι LL/1 διότι παρουσιάζει τα εξής προβλήματα:

- Ο κανόνας **<πρόταση>** ::= <λέξη> | **<πρόταση>S<λέξη>** παρουσιάζει **αριστερή αναδρομή** και διορθώνεται ως εξής:

<πρόταση> ::= <λέξη> **<K>**

<K> ::= S <λέξη><K> | ε

- Ο κανόνας **<λέξη>** ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή> δεν έχει πρόβλημα και παραμένει όπως είναι
- Ο κανόνας **<συλλαβή>** ::= <βάση> | <βάση> <σύμφωνο> | **a** <βάση> | **a** <σύμφωνο> χρειάζεται **αριστερή παραγοντοποίηση** και διορθώνεται ως εξής:

<συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | **a** <**N**>

<**N**> ::= βάση | <σύμφωνο>

- Οι τελευταίοι 2 κανόνες:

- <βάση> ::= <σύμφωνο> **a**
- <σύμφωνο> ::= **b** | **d**

δεν έχουν πρόβλημα και παραμένουν όπως είναι.

Τελική Διορθωμένη LL/1 Γραμματική

- <πρόταση> ::= <λέξη> **<K>**
- <K> ::= S <λέξη><K> | ε
- <λέξη> ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή>
- <συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | **a** <**N**>
- <**N**> ::= βάση | <σύμφωνο>
- <βάση> ::= <σύμφωνο> **a**
- <σύμφωνο> ::= **b** | **d**

3.5.12 Θέμα 4 Σεπτέμβριος 2017

α) Να κατασκευαστεί γραμματική Χ.Σ. **σε μορφή BNF** που να κατασκευάζει τις ακόλουθες συμβολοσειρές:

01.....101.....10 για $n \geq 1$

Χαρακτηριστικά: Έχει ένα μηδέν στην αρχή, ένα μηδέν στο τέλος και ένα μηδέν στη μέση. Αριστερά από το μεσαίο μηδέν έχει n διαδοχικούς άσσους και δεξιά από το μεσαίο μηδέν έχει n+1 διαδοχικούς άσσους.

Είναι η γραμματική LL/1; Άν όχι να τη μετατρέψετε σε LL/1

β) Να κατασκευάσετε τον bottom-up parser για το string **010110**

γ) Να κατασκευάσετε τον top-down parser για το string 01101110

Απάντηση

α) **01.....1 0 1.....10**

$\langle S \rangle ::= 01 \langle A \rangle 10$

$\langle A \rangle ::= 1 \langle A \rangle 1 \mid 01$

β) Η γραμματική είναι LL/1

Παραγωγές

$S \rightarrow 01A10 \rightarrow 010110$

$S \rightarrow 01A10 \rightarrow 01 1A1 10 \rightarrow 01 1011 10$

01101110

(b) Η **Στοίβα Ολίσθησης – Ελάττωσης Bottom-UP Στοίβα Συντακτικής Ανάλυσης** για την πρόταση **010110** είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη	
0	ϵ	010110 EOF	Ολίσθηση	
1	0	10110 EOF	Ολίσθηση	
2	01	10110 EOF	Ολίσθηση	
3	010	10110 EOF	Ολίσθηση	
4	<u>0101</u>	10 EOF	Ελάττωση $\langle A \rangle ::= 01$	$\langle S \rangle ::= 01 \langle A \rangle 10$
5	01 $\langle A \rangle$	10 EOF	Ολίσθηση	
6	01 $\langle A \rangle 1$	0 EOF	Ολίσθηση	
7	01 $\langle A \rangle 10$	EOF	Ελάττωση $\langle S \rangle ::= 01 \langle A \rangle 10$	
8	<S>	EOF	Αναγνώριση	

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	<u>Q1101110 EOF</u>	Ολίσθηση
1	0	<u>1101110 EOF</u>	Ολίσθηση
2	01	<u>101110 EOF</u>	Ολίσθηση
3	011	<u>01110 EOF</u>	Ολίσθηση
4	0110	<u>110 EOF</u>	Ολίσθηση
5	01101	<u>110 EOF</u>	Ελάττωση $<A> ::= 01$
6	011 $<A>$	<u>110 EOF</u>	Ολίσθηση
7	011 $<A>1$	<u>10 EOF</u>	Ελάττωση $<A> ::= 1 <A> 1$
8	01 $<A>$	<u>10 EOF</u>	Ολίσθηση
9	01 $<A>1$	<u>0 EOF</u>	Ολίσθηση
10	01 $<A>10$	<u>EOF</u>	Ελάττωση $<S> ::= 01 <A> 10$
11	<u><S></u>	<u>EOF</u>	ΑΝΑΓΝΩΡΙΣΗ

(c) Η Στοίβα Πρόβλεψης– Ταιριάσματος Top-Down Στοίβα Συντακτικής Ανάλυσης για την πρόταση 01101110 είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	01101110 EOF	Πρόβλεψη <S>::=01<A>10
1	01 <A> 10	01101110 EOF	Ταίριασμα συμβόλου 0
2	01<A>1	1101110 EOF	Ταίριασμα συμβόλου 1
3	01<A>	101110 EOF	Πρόβλεψη <A>::=1<A>1
4	01 1<A>1	101110 EOF	Ταίριασμα συμβόλου 1
5	0 1 1<A>0	01110 EOF	Ταίριασμα συμβόλου 0
6	† 1<A> 1	1110 EOF	Ταίριασμα συμβόλου 1
7	† <A> 1	110 EOF	Ταίριασμα συμβόλου 1
8	<A>	10 EOF	Πρόβλεψη <A>::=01
9	0 1	10 EOF	Ταίριασμα συμβόλου 1
10	0	0 EOF	Ταίριασμα συμβόλου 0
11	ε	EOF	ΑΝΑΓΝΩΡΙΣΗ

3.5.13 Θέμα 4 Φεβρουάριος 2019

Δίνονται οι ακόλουθες τρεις γραμματικές Χωρίς Συμφραζόμενα σε συμβολισμό BNF:

- 1^η Γραμματική $<A> ::= 0 <A> \mid 1 <A> \mid 0$
- 2^η Γραμματική $<\Sigma> ::= <\Sigma> <\Sigma> \mid 1 \mid 0$
- 3^η Γραμματική $<\Lambda> ::= 0 <\Lambda> 0 \mid 1 <\Lambda> 1 \mid 0 \mid 1$

Τι είδους συμβολοσειρές παράγει η κάθε γραμματική;

- Κάνουμε πρώτα παραγωγές

$A \Rightarrow 0$

$A \Rightarrow 0A \Rightarrow 00$

$A \Rightarrow 1A \Rightarrow 10$

$A \Rightarrow 1A \Rightarrow 11A \Rightarrow 110$

$A \Rightarrow 0A \Rightarrow 00A \Rightarrow 001A \Rightarrow 0010$

$A \Rightarrow 1A \Rightarrow 10A \Rightarrow 101A \Rightarrow 1010A \Rightarrow 10100$

Η 1^η Γραμματική παράγει συμβολοσειρές που τελειώνουν σε 0

- Κάνουμε πρώτα παραγωγές

$\Sigma \Rightarrow 1$

$\Sigma \Rightarrow 0$

$\Sigma \Rightarrow \Sigma\Sigma \Rightarrow 01$

$\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \Sigma\Sigma\Sigma \Rightarrow 111$

$\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \underline{\Sigma\Sigma} \Rightarrow \Sigma\Sigma\Sigma \Rightarrow 1110$

$\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \Sigma\Sigma\Sigma \Rightarrow \Sigma\Sigma\Sigma\Sigma \Rightarrow 01100$

Η 2η Γραμματική παράγει όλες τις μη κενές συμβολοσειρές από 0 και 1

- Κάνουμε πρώτα παραγωγές

$\Lambda \Rightarrow 0$

$\Lambda \Rightarrow 1$

$\Lambda \Rightarrow 0\Lambda 0 \Rightarrow 010$

$\Lambda \Rightarrow 1\Lambda 1 \Rightarrow 10\Lambda 01 \Rightarrow 10001$

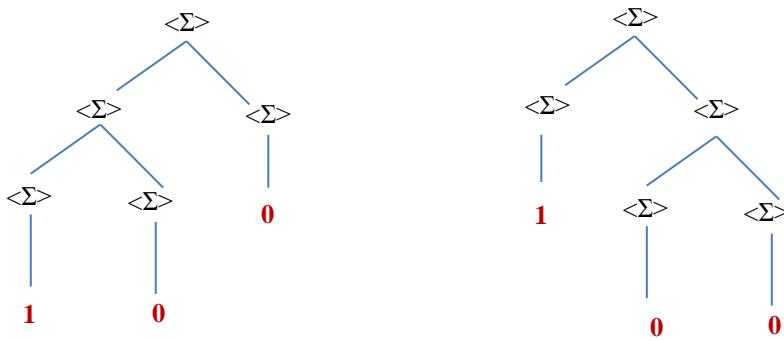
$\Lambda \Rightarrow 1\Lambda 1 \Rightarrow 10\Lambda 01 \Rightarrow 101\Lambda 101 \Rightarrow 1010101$

$\Lambda \Rightarrow 0\Lambda 0 \Rightarrow 00\Lambda 00 \Rightarrow 001\Lambda 100 \Rightarrow 0010\Lambda 0100 \Rightarrow 001010100$

Η 3η Γραμματική παράγει παλινδρομικές συμβολοσειρές περιττού μήκους

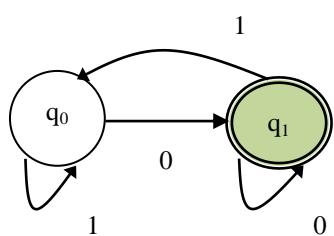
- Ποια(ες) Γραμματική (ες) είναι διφορούμενη (ες);

- Η 1^η Γραμματική ΔΕΝ είναι διφορούμενη
- Η 2^η Γραμματική είναι διφορούμενη διότι π.χ. η συμβολοσειρά 101 έχει 2 παραγωγές: $\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \underline{\Sigma\Sigma} \Rightarrow 101$ και $\Sigma \Rightarrow \Sigma\Sigma \Rightarrow \Sigma\Sigma\Sigma \Rightarrow 101$
- Η 3^η Γραμματική ΔΕΝ είναι διφορούμενη



- Ποια(ες) Γραμματική (ες) είναι κανονική (ες);
 - Η 1^η Γραμματική είναι κανονική διότι σε κάθε κανόνα $<A> ::= 0<A> \mid 1<A> \mid 0$ υπάρχει το πολύ 1 μη τερματικό σύμβολο και είναι το τελευταίο σύμβολο του κανόνα. Στο κανόνα $<A> ::= 0$ δεν υπάρχει κανένα ΜΗ τερματικό σύμβολο και αυτό είναι επιτρεπτό
 - Η 2^η Γραμματική ΔΕΝ είναι κανονική διότι στον κανόνα $<\Sigma> ::= <\Sigma><\Sigma>$ υπάρχουν 2 ΜΗ τερματικά σύμβολα στο δεξί μέρος του κανόνα
 - Η 3^η Γραμματική ΔΕΝ είναι κανονική διότι στον κανόνα $<\Lambda> ::= 0<\Lambda>0$ το ΜΗ τερματικό σύμβολο στο δεξί μέρος του κανόνα δεν βρίσκεται στο τέλος του κανόνα δηλ. ΔΕΝ είναι το τελευταίο σύμβολο του κανόνα
- Για τη γραμματική που είναι κανονική να κατασκευάσετε DFA

Για την 1η γραμματική που είναι κανονική μπορούμε να κατασκευάσουμε DFA που να αναγνωρίζει τις συμβολοσειρές της γλώσσας που τελειώνουν σε 0.



Να εξετάστε ποιες είναι LL/1 και να τις διορθώσετε

- 1^η Γραμματική $<A> ::= 0<A> \mid 1<A> \mid 0$
- 2^η Γραμματική $<\Sigma> ::= <\Sigma><\Sigma> \mid 1 \mid 0$
- 3^η Γραμματική $<\Lambda> ::= 0<\Lambda>0 \mid 1<\Lambda>1 \mid 0 \mid 1$

Απάντηση

Η 1^η Γραμματική $<A> ::= \underline{0<A>} \mid 1<A> \mid \underline{0}$ Δεν είναι LL/1 διότι έχει 2 εναλλακτικούς κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο δηλ. με 0

Διόρθωση

$<A> ::= \underline{0<K>} \mid 1<A>$

<K> ::= <A> | ε

Η διορθωμένη LL/1 Γραμματική είναι

<A> ::= 0<K> | 1<A>

<K> ::= <A> | ε

Η 2^η Γραμματική <Σ> ::= <Σ><Σ> | 1 | 0 ΔΕΝ είναι LL/1 διότι έχει αριστερή αναδρομή

Διόρθωση

<Σ> ::= 1<M> | 0<M>

<M> ::= <Σ><M> | ε

Η διορθωμένη LL/1 Γραμματική είναι

<Σ> ::= 1<M> | 0<M>

<M> ::= <Σ><M> | ε

Η 3^η Γραμματική <Λ> ::= 0<Λ>0** | 1<Λ>**1** | 0 | 1 Δεν είναι LL/1 διότι έχει 2 εναλλακτικά ζεύγη κανόνων που αρχίζουν με το ίδιο τερματικό σύμβολο**

<Λ> ::= 0<N> | 1

<N> ::= <Λ>0** | ε**

** ::= <Λ>**1** | ε**

Η διορθωμένη LL/1 Γραμματική είναι

<Λ> ::= 0<N> | 1

<N> ::= <Λ>0** | ε**

** ::= <Λ>**1** | ε**

3.5.14 Θέμα 1 Ιούνιος 2019-Ομάδα Α

Δίνεται η παρακάτω BNF γραμματική, τμήμα της περιγραφής του συντακτικού μιας γλώσσας προγραμματισμού. Τα τερματικά σύμβολα της γραμματικής είναι το: **id**, **=**, **(**, **σύμβολο εντολής ανάθεσης**, **)**, **+**, *****

```
<S> ::= id = <S> | <B> <A>  
<A> ::= + <B> <A> | ∈  
<B> ::= <D> <C>  
<C> ::= * <D> <C> | ∈  
<D> ::= (<S>) | id
```

(α) Τι είδους κατασκευές της γλώσσας δέχεται η παραπάνω γραμματική; **Ποια ιδιομορφία της γλώσσας C είναι αποδεκτή από τη γραμματική;**

(b) Η συμβολοσειρά $N=P * (D = T) + Y$ είναι νόμιμη με βάση την παραπάνω γραμματική; Δικαιολογήστε την απάντησή σας κατασκευάζοντας το **Δέντρο Συντακτικής Ανάλυσης** της συμβολοσειράς.

(c) Ελέγξτε, επίσης, αν η συμβολοσειρά **U+F** είναι μέλος της γλώσσας, παρουσιάζοντας τη **Στοίβα Ταιριάσματος-Πρόβλεψης Top – Down συντακτικής ανάλυσης**.

(δ) Ελέγξτε, επίσης, αν η συμβολοσειρά **(U+F)** είναι μέλος της γλώσσας, παρουσιάζοντας τη **Στοίβα Ταιριάσματος-Πρόβλεψης Top – Down συντακτικής ανάλυσης**

Απάντηση

a)

$S \Rightarrow id = S \Rightarrow id = SBA \Rightarrow id = DC \Rightarrow id = id$

$S \Rightarrow id = S \Rightarrow id = id = S \Rightarrow id = id = BA \Rightarrow id = id = DC + BA \Rightarrow id = id = id * DC + DC \Rightarrow id = id = id * id + id$

$S \Rightarrow BA \Rightarrow DC \Rightarrow id$

$S \Rightarrow BA \Rightarrow DC + BA \Rightarrow id * DC + DC \Rightarrow id * id + id$

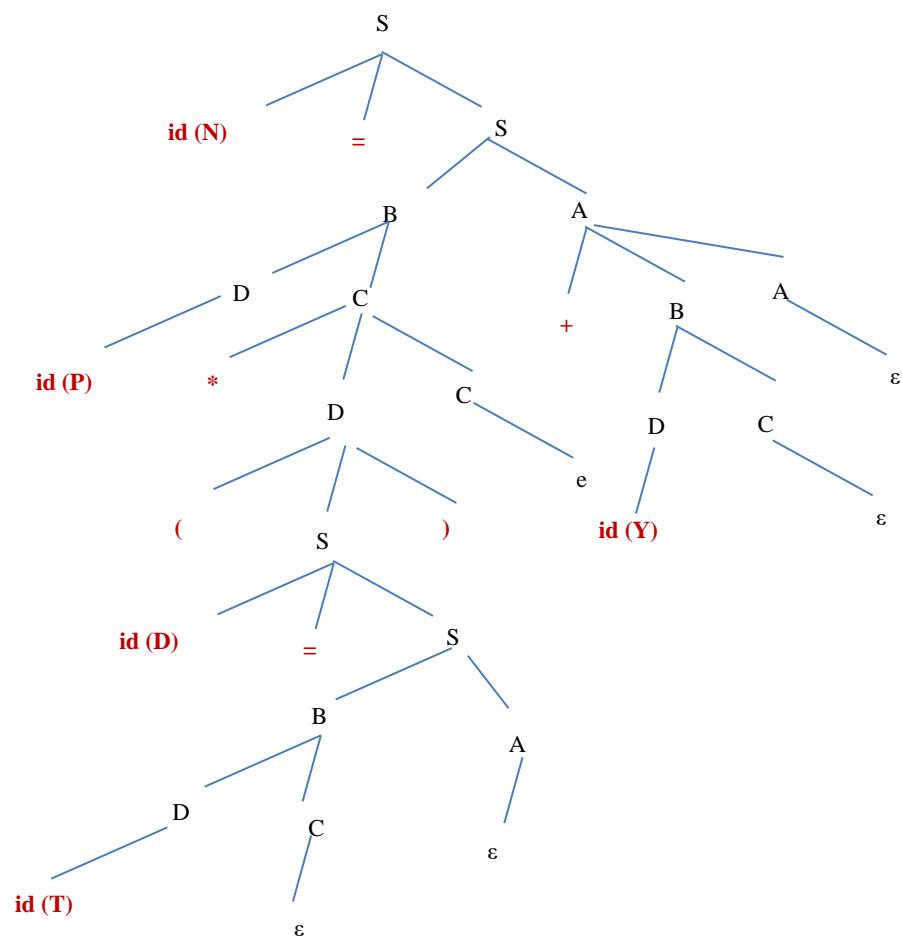
$S \Rightarrow id = S \Rightarrow id = BA \Rightarrow id = DC + BA \Rightarrow id = (S) * D + DC \Rightarrow id = (BA) * id + id \Rightarrow id = (id) * id + id$

$S \Rightarrow BA \Rightarrow DC \Rightarrow (S) * DC \Rightarrow (id = S) * id \Rightarrow (id = BA) * id \Rightarrow (id = DC) * id \Rightarrow (id = id) * id$

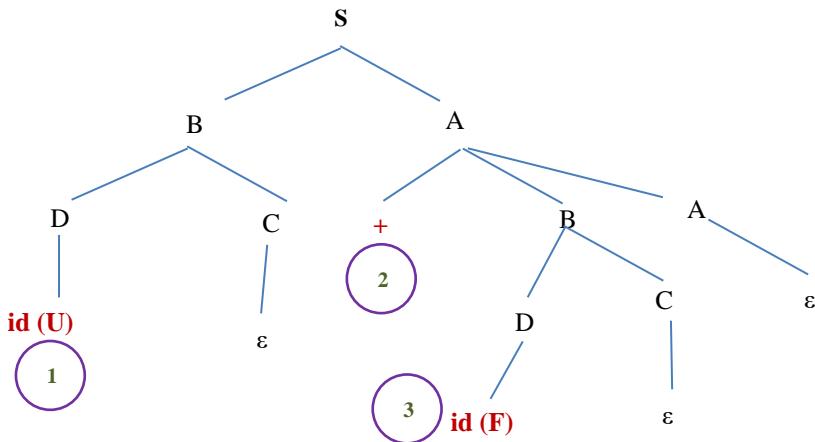
- Η γραμματική αυτή κατασκευάζει συμβολοσειρές που περιλαμβάνουν εντολές καταχώρισης και παραστάσεις με τους τελεστές *, + και ()
- Η ιδιομορφία της γλώσσας C που είναι αποδεκτή από τη γραμματική είναι ότι υπολογίζεται μια παράσταση μέσα σε παρενθέσεις και το αποτέλεσμα της παράστασης **συγκρίνεται αυτόματα με το μηδέν με την εξορισμού συνθήκη !=0**. Π.χ. στη συμβολοσειρά $(id = id) * id$ αφού γίνει η καταχώριση ελέγχεται η νέα τιμή του id αν είναι διάφορη από το μηδέν. Αν είναι ίση με μηδέν παρένθεση επιστρέφει 1 ενώ αν δεν είναι επιστρέφει 0.

β)

$N = P^*(D=T) + Y$



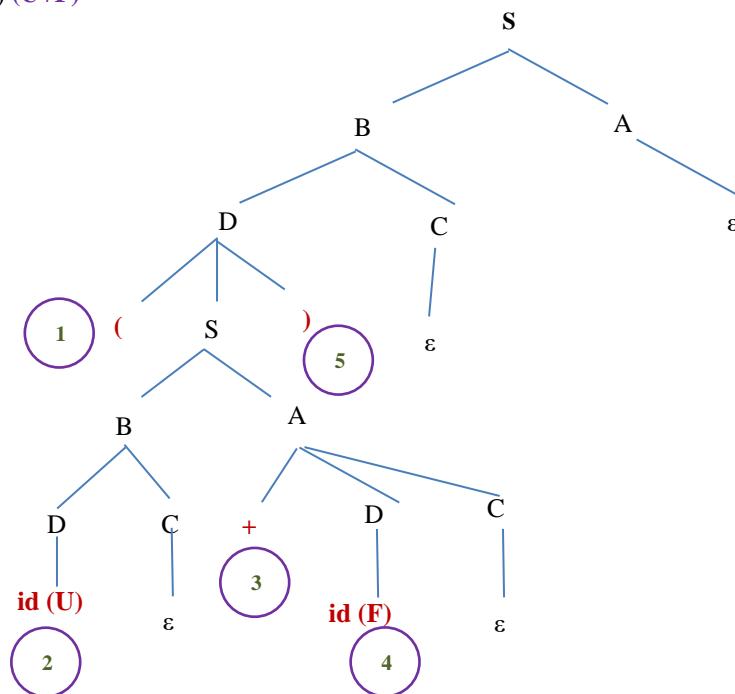
γ) **U+F**



Η Στοίβα Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης γίνεται με βάση το Συντακτικό Δέντρο

Βήμα	Στοίβα	Είσοδος	Πράξη
0	< S >	U+F EOF	Πρόβλεψη < S > ::= < B >< A >
1	< A >< B >	U+F EOF	Πρόβλεψη < B > ::= < D >< C >
2	< A >< C >< D >	U+F EOF	Πρόβλεψη < D > ::= id (U)
3	< A >< C > U	U+F EOF	Ταίριασμα συμβόλου U 1
4	< A >< C >	+F EOF	Πρόβλεψη < C > ::= ε
5	< A >	+F EOF	Πρόβλεψη < A > ::= +< B >< A >
6	< A >< B > +	+F EOF	Ταίριασμα συμβόλου + 2
7	< A >< B >	F EOF	Πρόβλεψη < B > ::= < D >< C >
8	< A >< C >< D >	F EOF	Πρόβλεψη < D > ::= id (F)
9	< A >< C > F	F EOF	Ταίριασμα συμβόλου F 3
10	< A >< C >	EOF	Πρόβλεψη < C > ::= ε
11	< A >	EOF	Πρόβλεψη < A > ::= ε
12	ε	EOF	Αναγνώριση

δ) (U+F)



Η Στοίβα Ολίσθησης Ελάττωσης Bottom-UP συντακτικής ανάλυσης γίνεται με βάση το Συντακτικό Δέντρο

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ε	(U+F) EOF	Ολίσθηση 1
1	(U+F EOF	Ολίσθηση 2
2	(U	+F EOF	Ελάττωση <D> ::= id (U)
3	<D>	+F EOF	Ελάττωση <C> ::= ε
4	<D><C>	+F EOF	Ελάττωση ::= <D><C>
5		+F EOF	Ολίσθηση 3
6	+	F EOF	Ολίσθηση 4
7	+F) EOF	Ελάττωση <D> ::= id (F)
8	+<D>) EOF	Ελάττωση <C> ::= ε
9	+<D><C>) EOF	Ελάττωση <A> ::= +<D><C>
10	<A>) EOF	Ελάττωση <S> ::= <A>
11	<S>) EOF	Ολίσθηση 5
12	<S>	EOF	Ελάττωση <D> ::= (<S>)
13	<D>	EOF	Ελάττωση <C> ::= ε

14	<D><C>	EOF	Ελάττωση ::= <D><C>
15		EOF	Ελάττωση <A> ::= ε
16	<A>	EOF	Ελάττωση <S> ::= <A>
17	<S>	EOF	Αναγνώριση

3.5.14.1 Θέμα 1 Ιούνιος 2019-Ομάδα Β και Φεβρουάριος 2023

Δίνεται η παρακάτω BNF γραμματική, τμήμα της περιγραφής του συντακτικού μιας γλώσσας προγραμματισμού. Τα τερματικά σύμβολα της γραμματικής είναι το: if then else id = (σύμβολο εντολής ανάθεσης) > <

<S> ::= if <A> then

 ::= <E> | else | <S>

<E> ::= id := id

<A> ::= id <op> id

<op> ::= > | < | >= | <= | == | !=

(a) Τι είδους κατασκευές της γλώσσας δέχεται η παραπάνω γραμματική;

(b) Η συμβολοσειρά if x<=y then if y>z then y:=x else x:=z είναι νόμιμη με βάση την παραπάνω γραμματική; Δικαιολογήστε την απάντησή σας κατασκευάζοντας το Δέντρο Συντακτικής Ανάλυσης της συμβολοσειράς.

(c) Ελέγξτε, επίσης, αν η συμβολοσειρά if x==y then y:=z else x:=z είναι μέλος της γλώσσας; Δικαιολογήστε παρουσιάζοντας τη **Στοίβα Ολίσθησης-Ελάττωσης Bottom-Up** συντακτικής ανάλυσης

(d) Είναι η **Γραμματική LL/1**; Αν όχι διορθώστε τη

Απάντηση

(a)

S ⇒ if A then B ⇒ if id op id then E ⇒ if id == id then id := id

S ⇒ if A then B ⇒ if id op id then E ⇒ if id <= id then id := id

S ⇒ if A then B ⇒ if id op id then B else B ⇒ if id >= id then E else E ⇒ if id >= id then id := id else id := id

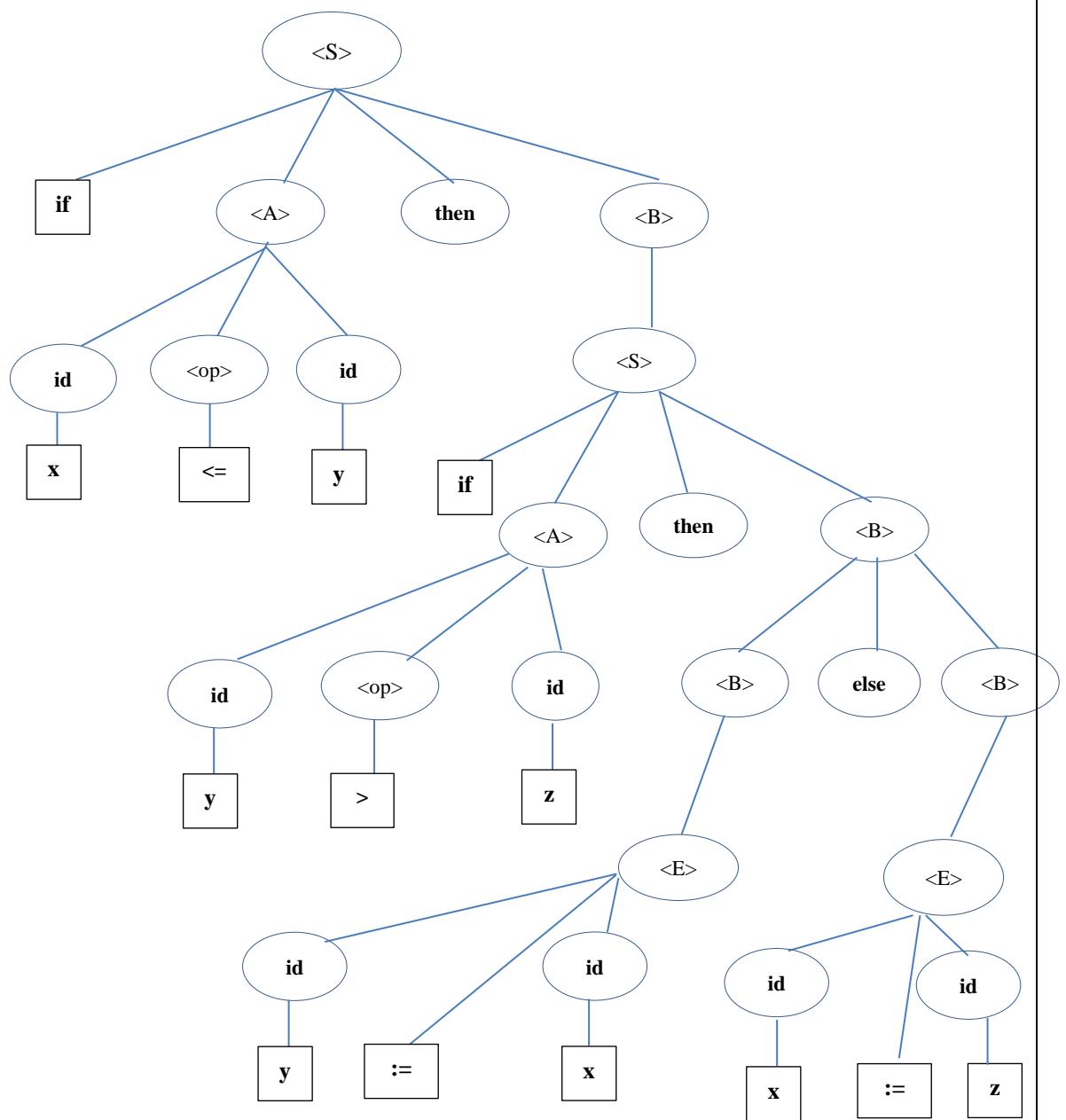
S ⇒ if A then B ⇒ if A then S ⇒ if A then if A then B ⇒ if id op id then if id op id then E ⇒ if id != id then if id == id then id := id

S ⇒ if A then B ⇒ if A then B else B ⇒ if A then S else S ⇒ if A then if A then B else if A then B ⇒ if id <op> id then <E> else if id <op> id then B ⇒ if id <= id then id := id else if id != id then id := id

Η συγκεκριμένη γραμματική κατασκευάζει συμβολοσειρές οι οποίες αποτελούν υλοποίηση της εντολής ελέγχου if σε κάποια γλώσσα προγραμματισμού. Πιο συγκεκριμένα οι συμβολοσειρές αυτές αφορούν:

- είτε μια απλή if με μια συνθήκη που αν ικανοποιείται τότε γίνεται μια καταχώριση της μορφής id := id αλλιώς τίποτα
- είτε μια if..else που αν ικανοποιείται μια συνθήκη τότε γίνεται μια καταχώριση αλλιώς μια διαφορετική καταχώριση όπως π.χ. if id = id then id := id else id := id
- είτε μια εμφωλευμένη if όπου ελέγχονται πολλές συνθήκες όπως if id > id then id := id else if id < id then id := id else id := id

(b) if $x \leq y$ then if $y > z$ then $y := x$ else $x := z$



(c) Στοίβα Ολίσθησης-Ελάττωσης Bottom-Up συντακτικής ανάλυσης

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	if $x==y$ then $y:=z$ else $x:=z$ EOF	Ολίσθηση
1	if	$x==y$ then $y:=z$ else $x:=z$ EOF	Ολίσθηση
2	if x	$== y$ then $y:=z$ else $x:=z$ EOF	Ολίσθηση
3	if x ==	y then $y:=z$ else $x:=z$ EOF	Ελάττωση $<\text{op}> ::= ==$
4	if x $<\text{op}>$	y then $y:=z$ else $x:=z$ EOF	Ολίσθηση
5	if x $<\text{op}>$ y	then $y:=z$ else $x:=z$ EOF	Ελάττωση $<\text{A}> ::= \text{id} <\text{op}> \text{id}$
6	if $<\text{A}>$	then $y:=z$ else $x:=z$ EOF	Ολίσθηση
7	if $<\text{A}>$ then	y := z else $x:=z$ EOF	Ολίσθηση
8	if $<\text{A}>$ then y	:= z else $x:=z$ EOF	Ολίσθηση
9	if $<\text{A}>$ then y :=	z else $x:=z$ EOF	Ολίσθηση
10	if $<\text{A}>$ then y := z	else $x:=z$ EOF	Ελάττωση $<\text{E}> ::= \text{id} := \text{id}$
11	if $<\text{A}>$ then $<\text{E}>$	else $x:=z$ EOF	Ολίσθηση
12	if $<\text{A}>$ then $<\text{E}>$ else	x := z EOF	Ολίσθηση
13	if $<\text{A}>$ then $<\text{E}>$ else x	:= z EOF	Ολίσθηση
14	if $<\text{A}>$ then $<\text{E}>$ else x :=	z EOF	Ολίσθηση
15	if $<\text{A}>$ then $<\text{E}>$ else x := z	EOF	Ελάττωση $<\text{E}> ::= \text{id} := \text{id}$
16	if $<\text{A}>$ then $<\text{E}>$ else $<\text{E}>$	EOF	Ελάττωση $<\text{B}> ::= <\text{E}>$
17	if $<\text{A}>$ then $<\text{E}>$ else $<\text{B}>$	EOF	Ελάττωση $<\text{B}> ::= <\text{E}>$
18	if $<\text{A}>$ then $<\text{B}>$ else $<\text{B}>$	EOF	Ελάττωση $<\text{B}> ::= <\text{B}>$ else $<\text{B}>$
19	if $<\text{A}>$ then $<\text{B}>$	EOF	Ελάττωση $<\text{S}> ::= \text{if } <\text{A}> \text{ then } <\text{B}>$
20	$<\text{S}>$	EOF	ΑΝΑΓΝΩΡΙΣΗ

(d)

Η γραμματική ΔΕΝ είναι LL/1 διότι παρουσιάζει αριστερή αναδρομή στον κανόνα $<\text{B}> ::= <\text{B}>$ else $<\text{B}>$

Επίσης υπάρχουν εναλλακτικοί κανόνες της γραμματικής που αρχίζουν με το ίδιο μη τερματικό σύμβολο όπως είναι οι κανόνες $<\text{op}> ::= > | >=$ και $<\text{op}> ::= < | <=$

Η διορθωμένη LL/1 Γραμματική είναι η ακόλουθη:

$<\text{S}> ::= \text{if } <\text{A}> \text{ then } <\text{B}>$

$<\text{B}> ::= <\text{E}> <\text{K}> | <\text{S}> <\text{K}>$

$<\text{K}> ::= \text{else } <\text{B}> <\text{K}> | \epsilon$

<E> ::= **id** := **id**

<A> ::= **id** <op> **id**

<op> ::= > <**M**> | <**N**> | == | !=

<**M**> ::= ε | =

<**N**> ::= ε | =

3.5.15 Φροντιστήριο 2019

Δίνεται η παρακάτω BNF γραμματική, με τερματικά σύμβολα τα: a b c.

$\langle A \rangle ::= \langle A \rangle a \mid \langle A \rangle b \mid \langle A \rangle \mid \langle B \rangle$

$\langle B \rangle ::= c$

(α) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε τα βασικά χαρακτηριστικά των συμβολοσειρών αυτών. Δώστε παραδείγματα.

(β) Η συμβολοσειρά **cacab** είναι νόμιμη με βάση την παραπάνω γραμματική; Δικαιολογήστε την απάντησή σας κατασκευάζοντας το Δέντρο Συντακτικής Ανάλυσης της συμβολοσειράς.

(γ) Ελέγξτε αν η συμβολοσειρά **cbcac** είναι μέλος της γλώσσας, παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης.

(δ) Ποιες **αρνητικές ιδιότητες** έχει η γραμματική; Περιγράψτε σύντομα πως θα τις εξαλείφατε.

Απάντηση

(α)

Παραγωγές

$A \Rightarrow B \Rightarrow c$

$A \Rightarrow AaA \Rightarrow BaB \Rightarrow \underline{cac}$

$A \Rightarrow AbA \Rightarrow BaB \Rightarrow \underline{cbc}$

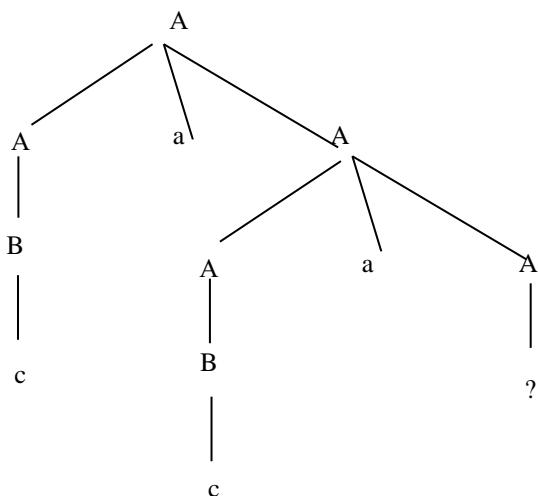
$A \Rightarrow AaA \Rightarrow AaAaAaA \Rightarrow BaBaBaB \Rightarrow \underline{cacacac}$

$A \Rightarrow AbA \Rightarrow BbB \Rightarrow AbA \Rightarrow AbAbAaA \Rightarrow BbBbBaB \Rightarrow \underline{cbcacac}$

Χαρακτηριστικά

- συμβολοσειρές περιττού μήκους
- αρχίζουν και τελειώνουν με c
- τα c είναι περισσότερα κατά ένα από το συνολικό πλήθος των a και b
- Οι χαρακτήρες c βρίσκονται μόνο στις περιττές θέσεις κάθε συμβολοσειράς
- Υπάρχει διαδοχική εναλλαγή χαρακτήρων
- Μεταξύ των a και b παρεμβάλλονται c

(β)



(γ)

Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	cbcac EOF	Ολίσθηση
1	c	bcac EOF	Ελάττωση $::= c$
2	$$	bcac EOF	Ολίσθηση
3	b	cac EOF	Ολίσθηση
4	bc	ac EOF	Ελάττωση $::= c$
5	b	ac EOF	Ολίσθηση
6	ba	c EOF	Ολίσθηση
7	bac	EOF	Ελάττωση $::= c$
8	ba	EOF	Ελάττωση $<A> ::= $
9	$ba<A>$	EOF	Ελάττωση $<A> ::= $
10	$b<A>a<A>$	EOF	Ελάττωση $<A> ::= $
11	$<A>b<A>a<A>$	EOF	Ελάττωση $<A> ::= <A> a <A>$
12	$<A>b<A>$	EOF	Ελάττωση $<A> ::= <A> b <A>$
13	$<A>$	EOF	Αναγνώριση

(δ)

$<A> ::= <A> a <A> \mid <A> b <A> \mid $

$::= c$

Οι 2 πρώτοι κανόνες παρουσιάζουν αριστερή αναδρομή οπότε κάνουμε απαλοιφή της αριστερής αναδρομής για να την μετατρέψουμε σε LL/1

$<A> ::= <K>$

$<K> ::= a <A> <K> \mid b <A> <K> \mid \epsilon$

$::= c$

3.5.16 Θέμα 3 Σεπτέμβριος 2016

Δίνεται η παρακάτω BNF γραμματική, με τερματικά σύμβολα τα: a b c d.

$\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle \mid d$

$\langle A \rangle ::= a \langle A \rangle \mid \epsilon$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

$\langle C \rangle ::= c$

(α) Είναι η γραμματική LL (1); Αιτιολογήστε την απάντησή σας.

(β) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε μερικά βασικά χαρακτηριστικά των συμβολοσειρών αυτών.

(γ) Ελέγξτε αν η συμβολοσειρά **acb** είναι μέλος της γλώσσας, παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης.

Απάντηση

(α) Η Γραμματική είναι LL(1) διότι ΔΕΝ έχει αριστερή αναδρομή ούτε και χρειάζεται παραγοντοποίηση

$\langle S \rangle ::= \langle A \rangle \langle B \rangle \langle C \rangle \langle S \rangle \mid d$

$\langle A \rangle ::= a \langle A \rangle \mid \epsilon$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

$\langle C \rangle ::= c$

(β)

$S \Rightarrow d$

$S \Rightarrow ABCS \Rightarrow aAbBcd \Rightarrow \text{abcd}$

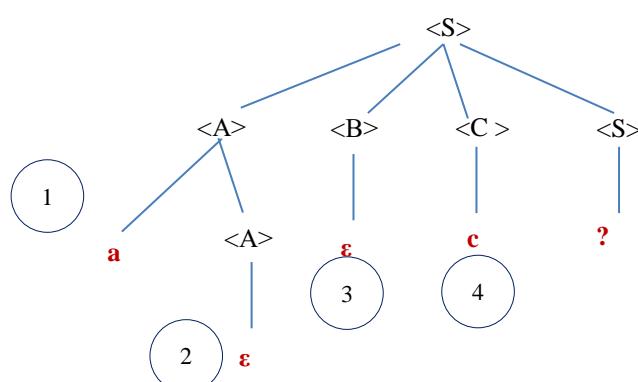
$S \Rightarrow ABCS \Rightarrow aAbBcd \Rightarrow aaAbcd \Rightarrow \text{aabcd}$

$S \Rightarrow ABCS \Rightarrow bBcd \Rightarrow bbBcd \Rightarrow bbbBcd \Rightarrow \text{bbbcd}$

$S \Rightarrow ABCS \Rightarrow ABCABC \Rightarrow caAbBcd \Rightarrow \text{cabcd}$

- Περιγράφει ΜΗ κενές συμβολοσειρές που τελειώνουν σε d

(γ)



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ε	acb EOF	Ολίσθηση  1
1	a	cb EOF	Ελάττωση $<A> ::= \varepsilon$  2
2	a<A>	cb EOF	Ελάττωση $::= \varepsilon$  3
3	<A>	cb EOF	Ολίσθηση  4
4	<A>c	b EOF	Ελάττωση $<C> ::= c$
5	<A><C>	b EOF	ΜΗ ΑΝΑΓΝΩΡΙΣΗ

3.5.17 Θέμα 3 Σεπτέμβριος 2018

Δίνεται η παρακάτω BNF γραμματική, με τερματικά σύμβολα τα: a, b.

$\langle S \rangle ::= b\langle F \rangle \mid a\langle T \rangle$

$\langle T \rangle ::= a\langle S \rangle \mid a$

$\langle F \rangle ::= b\langle S \rangle \mid b$

(a) Περιγράψτε τα χαρακτηριστικά των μελών της γλώσσας που παράγει η γραμματική

(b) Είναι το **bbbbbaabb** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης για το string

(c) Η παραπάνω γραμματική δεν είναι LL (1). Εξηγείστε γιατί. Μετασχηματίστε τη γραμματική σε LL/1 εφαρμόζοντας και αναφέροντας τους απαραίτητους μετασχηματισμούς

(d) Είναι το **aaabba** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη Στοίβα Πρόβλεψης-Ταιριάσματος Top-Down συντακτικής ανάλυσης για το string χρησιμοποιώντας τη γραμματική του ερωτήματος (c)

Απάντηση

(α)

Παραδείγματα Παραγόμενων Συμβολοσειρών

$S \Rightarrow b\langle F \rangle \Rightarrow \text{bb}$

$S \Rightarrow b\langle F \rangle \Rightarrow bb\langle S \rangle \Rightarrow bbb\langle F \rangle \Rightarrow \text{bbbb}$

$S \Rightarrow b\langle F \rangle \Rightarrow bb\langle S \rangle \Rightarrow bba\langle T \rangle \Rightarrow \text{bbaa}$

$S \Rightarrow a\langle T \rangle \Rightarrow \text{aa}$

$S \Rightarrow a\langle T \rangle \Rightarrow aa\langle S \rangle \Rightarrow aaa\langle T \rangle \Rightarrow \text{aaaa}$

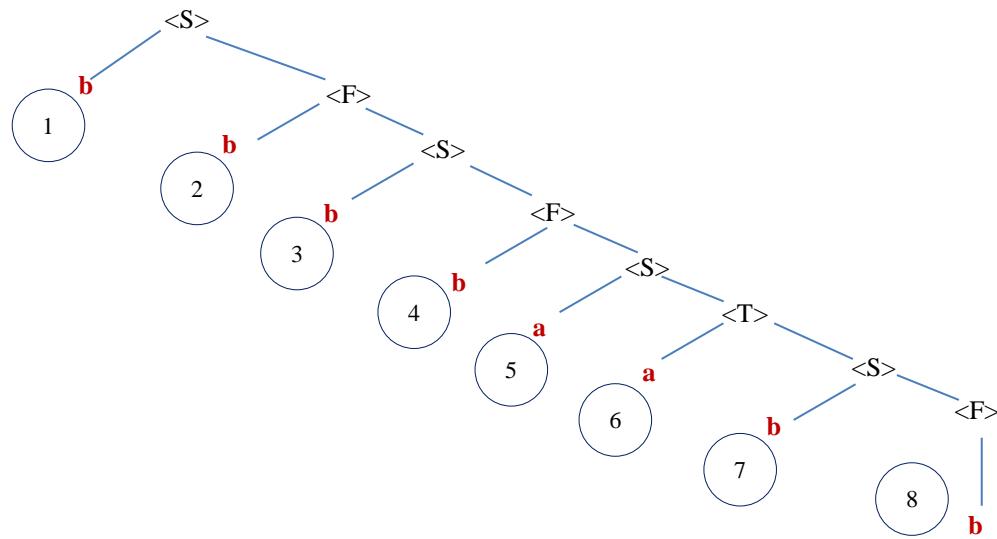
$S \Rightarrow a\langle T \rangle \Rightarrow aa\langle S \rangle \Rightarrow aab\langle F \rangle \Rightarrow \text{aab}$

$S \Rightarrow a\langle T \rangle \Rightarrow aa\langle S \rangle \Rightarrow aab\langle F \rangle \Rightarrow aabb\langle S \rangle \Rightarrow aabba\langle T \rangle \Rightarrow aabbaa\langle S \rangle \Rightarrow aabbaaa\langle T \rangle \Rightarrow \text{aabbaaaa}$

Χαρακτηριστικά Παραγόμενων Συμβολοσειρών

- Το ελάχιστο μήκος συμβολοσειράς είναι 2 και οι μικρότερες συμβολοσειρές που σχηματίζονται είναι aa ή bb
- Οι συμβολοσειρές που σχηματίζονται έχουν συνολικά άρτιο μήκος
- Οι συμβολοσειρές που σχηματίζονται έχουν άρτιο πλήθος a που ακολουθείται από άρτιο πλήθος b και το αντίθετο

β) Το συντακτικό δέντρο για το string **bbbaabb** είναι



Η bottom-up στοίβα συντακτικής ανάλυσης για το string **bbbbaabb** είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	bbbbaabb EOF	Ολίσθηση 1
1	b	bbbbaabb EOF	Ολίσθηση 2
2	bb	bbaabb EOF	Ολίσθηση 3
3	bbb	baabb EOF	Ολίσθηση 4
4	bbbb	aabb EOF	Ολίσθηση 5
5	bbbba	abb EOF	Ολίσθηση 6
6	bbbbaa	bb EOF	Ολίσθηση 7
7	bbbbaab	b EOF	Ολίσθηση 8
8	bbbbaabb	EOF	Ελάττωση $<F> ::= b$
9	bbbbaab $<F>$	EOF	Ελάττωση $<S> ::= b <F>$
10	bbbbaa $<S>$	EOF	Ελάττωση $<T> ::= a <S>$
11	bbbba $<T>$	EOF	Ελάττωση $<S> ::= a <T>$
12	bbba $<S>$	EOF	Ελάττωση $<F> ::= b <S>$
13	bbb $<F>$	EOF	Ελάττωση $<S> ::= b <F>$
14	bb $<S>$	EOF	Ελάττωση $<F> ::= b <S>$
15	b $<F>$	EOF	Ελάττωση $<S> ::= b <F>$
16	$<S>$	EOF	Αναγνώριση

(c)

$$\begin{aligned} <S> ::= & b<F> \mid a<T> \\ <T> ::= & \mathbf{a}<S> \mid \mathbf{a} \\ <F> ::= & \mathbf{b}<S> \mid \mathbf{b} \end{aligned}$$

- Οι κανόνες $<S> ::= b<F> \mid a<T>$ δεν παραβιάζουν την LL/1 γραμματική
- Από τους κανόνες $<T> ::= a<S> \mid a$ που αρχίζουν με το ίδιο τερματικό σύμβολο βγάζουμε το a ως κοινό παράγοντα και έχουμε:

$<T> ::= a<\mathbf{K}>$ (βγάζουμε το a από κάθε κανόνα που αρχίζει με a ως κοινό παράγοντα και μετά βάζουμε ένα νέο μη τερματικό σύμβολο, εδώ βάζουμε το \mathbf{K})

$<\mathbf{K}> ::= <S> \mid \epsilon$ (από το νέο σύμβολο πηγαίνουμε σε κάθε κανόνα με κοινό παράγοντα το a και γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα. Στον 1^o κανόνα μετά το a υπάρχει το $<S>$. Στο 2^o δεν υπάρχει τίποτα μετά το a)
- Από τους κανόνες $<F> ::= b<S> \mid b$ βγάζουμε το b ως κοινό παράγοντα και έχουμε:

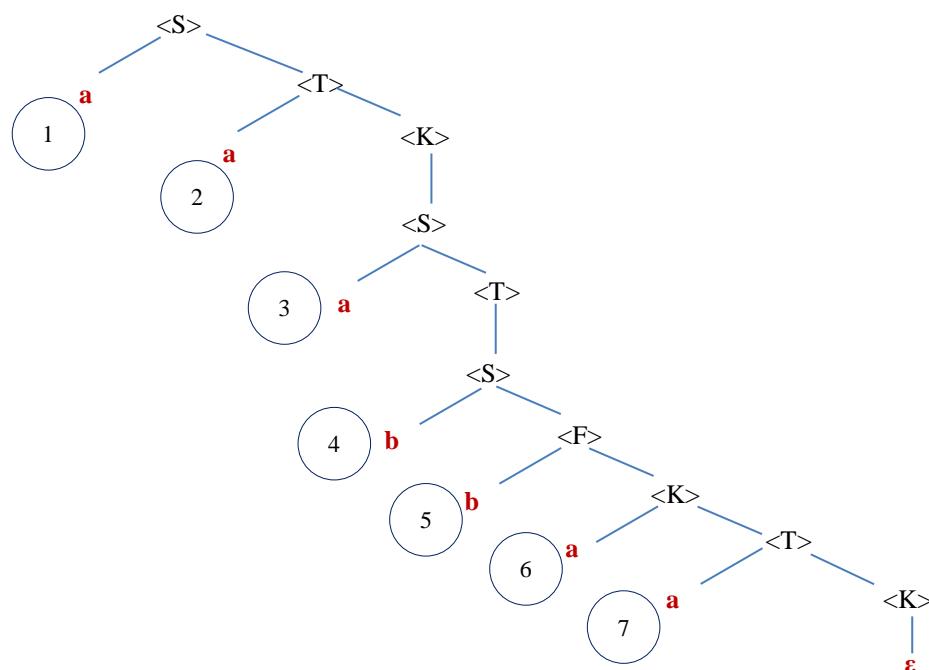
$<F> ::= b<\mathbf{M}>$ (βγάζουμε το b από κάθε κανόνα που αρχίζει με b ως κοινό παράγοντα και μετά βάζουμε ένα νέο μη τερματικό σύμβολο, εδώ βάζουμε το \mathbf{M})

$<\mathbf{M}> ::= <S> \mid \epsilon$ (από το νέο σύμβολο πηγαίνουμε σε κάθε κανόνα με κοινό παράγοντα το b και γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα. Στον 1^o κανόνα μετά το ba υπάρχει το $<S>$. Στο 2^o δεν υπάρχει τίποτα μετά το b)

Η τελική LL/1 γραμματική είναι:

$$\begin{aligned} <S> ::= & b<F> \mid a<T> \\ <T> ::= & a<\mathbf{K}> \\ <\mathbf{K}> ::= & <S> \mid \epsilon \\ <F> ::= & b<\mathbf{M}> \\ <\mathbf{M}> ::= & <S> \mid \epsilon \end{aligned}$$

(d) Το συντακτικό δέντρο για το string **aaabba** είναι



Θα αποδείξουμε ότι η συμβολοσειρά aaabba είναι έγκυρη βασιζόμενοι στην LL/1 Γραμματική που κατασκευάσαμε.

H top-down στοίβα συντακτικής ανάλυσης για το string **aaabbbaa** είναι η ακόλουθη:

Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	aaabba EOF	Πρόβλεψη <S>::=a<T>
1	<T> a	a aabba EOF	Ταίριασμα συμβόλου 1
2	<T>	aabba	Πρόβλεψη <T>::=a<K>
3	<T><K> a	a abba	Ταίριασμα συμβόλου a 2
4	<T><K>	abba	Πρόβλεψη <K>::=<S>
5	<T><S>	abba	Πρόβλεψη <S>::=a<T>
6	<T><T> a	a bba	Ταίριασμα συμβόλου a 3
7	<T><T>	bba	Πρόβλεψη <S>::=b<F>
8	<T><T><F> b	b baa	Ταίριασμα συμβόλου b 4
9	<T><T><F>	baa	Πρόβλεψη <S>::=b<K>
10	<T><T><K> b	b aa	Ταίριασμα συμβόλου b 5
11	<T><T><K>	aa	Πρόβλεψη <S>::=a<T>
12	<T><T><T> a	a a	Ταίριασμα συμβόλου a 6
13	<T><T><T>	a	Πρόβλεψη <T>::=a<K>
14	<T><T><T><K> a	a	Ταίριασμα συμβόλου a 7
15	<K>		Πρόβλεψη <K>::=e

3.5.18 Θέμα 3 Σεπτέμβριος 2019

Δίνεται η γραμματική:

$$\begin{aligned} <\text{A}> &::= (\text{B}) \\ <\text{B}> &::= \text{C} | \text{B}, \text{C} \\ <\text{C}> &::= \text{A} | \text{D} \\ <\text{D}> &::= w | x | y | z \end{aligned}$$

- (a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε τα βασικά χαρακτηριστικά των συμβολοσειρών αυτών. Δώστε και σχετικά παραδείγματα
- (b) Ελέγξτε αν η συμβολοσειρά $((w, x), y)$ είναι μέλος της γλώσσας παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης
- (c) Είναι η γραμματική LL(1); Αιτιολογήστε την απάντηση σας. Αν η γραμματική δεν είναι LL(1), κάντε τους κατάλληλους μετασχηματισμούς και παρουσιάστε την ισοδύναμη LL(1) γραμματική

Απάντηση

(a)

Παραδείγματα Παραγόμενων Συμβολοσειρών

$A \Rightarrow (B) \Rightarrow (C) \Rightarrow (D) \Rightarrow (w)$

$A \Rightarrow (B) \Rightarrow (C) \Rightarrow (A) \Rightarrow ((B)) \Rightarrow ((C)) \Rightarrow ((D)) \Rightarrow ((x))$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (C, C) \Rightarrow (A, C) \Rightarrow ((B), C) \Rightarrow ((C), C) \Rightarrow ((D), C) \Rightarrow ((D), x) \Rightarrow ((w, x), y)$

$A \Rightarrow (B) \Rightarrow (B, C) \Rightarrow (B, C, A) \Rightarrow (B, C, (B)) \Rightarrow (C, C, (C)) \Rightarrow (D, D, (D)) \Rightarrow (x, (y, z))$

Χαρακτηριστικά Παραγόμενων Συμβολοσειρών

Η γραμματική αυτή παράγει συμβολοσειρές από w, x, y, z μέσα σε παρενθέσεις σωστά εμφωλευμένες. Οι παρενθέσεις μπορεί να διαχωρίζονται μεταξύ τους με κόμματα και μπορεί να υπάρχει οποιοσδήποτε βαθμός εμφώλευσης τους. Οι παρενθέσεις μπορεί να περιλαμβάνουν τερματικά σύμβολα w, x, y, z διαχωριζόμενα με κόμμα είτε άλλες εμφωλευμένες παρενθέσεις.

(b) Παρακάτω δίνονται τα βήματα του **Bottom-UP Parser** για την αναγνώριση της συμβολοσειράς $(w, (x, y))$

Η συμβολοσειρά $(w, (x, y))$ που έχουμε κατασκευάσει από την τελευταία παραγωγή είναι νόμιμη και έχει δύο ζευγάρια παρενθέσεων και δύο κόμματα. Για να δούμε αν είναι μέλος της γλώσσας παρουσιάζουμε τη στοίβα Ολίσθησης-Ελάττωσης (Bottom-Up Parser) συντακτικής ανάλυσης.

Στοίβα Ολίσθησης-Ελάττωσης

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	$((w, x), y)$ EOF	Ολίσθηση
1	($(w, x), y$ EOF	Ολίσθηση
2	(($w, x), y$ EOF	Ολίσθηση
3	((w	, x), y) EOF	Ελάττωση $<\text{D}> ::= w$
4	((D)	, x), y) EOF	Ολίσθηση
5	((D),	x), y) EOF	Ολίσθηση

6	((<D>,x),y) EOF	Ελάττωση $<D> ::= x$
7	((<D>, <D>),y) EOF	Ολίσθηση
8	((<D>, <D>)	,y) EOF	Ελάττωση $<C> ::= <D>$
9	((<C>, <C>)	,y) EOF	Ελάττωση $::= <C>$
10	((, <C>)	,y) EOF	Ελάττωση $::= , <C>$
11	(()	,y) EOF	Ολίσθηση
12	((),	y) EOF	Ολίσθηση
13	((),y) EOF	Ολίσθηση
14	((),y)	EOF	Ελάττωση $<D> ::= y$
15	((),<D>)	EOF	Ελάττωση $<C> ::= (<D>)$
16	((),<C>)	EOF	Ελάττωση $<D> ::= <C>$
17	(<A>,<C>)	EOF	Ελάττωση $<C> ::= <A>$
18	(<C>,<C>)	EOF	Ελάττωση $::= <C>$
19	(,<C>)	EOF	Ελάττωση $::= , <C>$
20	()	EOF	Ελάττωση $<A> ::= ()$
21	<A>	EOF	Αναγνώριση

(c) Η γραμματική

$<A> ::= ()$

$::= <C> | , <C>$

$<C> ::= <A> | <D>$

$<D> ::= w | x | y | z$

δεν είναι LL(1) διότι στον κανόνα $::= , <C>$ υπάρχει αριστερή αναδρομή. Κάνουμε απαλοιφή της αριστερής αναδρομής και η LL(1) που προκύπτει είναι η ακόλουθη:

$<A> ::= ()$

$::= <C> <K>$ (πηγαίνουμε σε κάθε κανόνα που δεν έχει αναδρομή και βάζουμε στο τέλος του ένα νέο μη τερματικό σύμβολο. Εδώ βάζουμε το $<K>$ και γράφουμε $<C> <K>$)

$<K> ::= , <C> <K> | \epsilon$ (από το νέο μη τερματικό σύμβολο δηλ. από το $<K>$ πηγαίνουμε σε κάθε κανόνα που έχει αναδρομή, γράφουμε ότι υπάρχει μετα την αναδρομή, $<C>$ ακολουθούμενο από το νέο μη τερματικό σύμβολο δηλ. το $<K>$ άρα γράφουμε $, <C> <K>$)

$<C> ::= <A> | <D>$

$<D> ::= w | x | y | z$

3.5.19 Θέμα 3 Φεβρουάριος 2020

Δίνεται η παρακάτω BNF γραμματική με τερματικά σύμβολα τα a, b, c:

$$\langle S \rangle ::= \langle X \rangle \langle X \rangle \mid \langle Y \rangle$$
$$\langle X \rangle ::= a \langle X \rangle c \mid a \langle Y \rangle c$$
$$\langle Y \rangle ::= \langle Y \rangle b \mid \varepsilon$$

(a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε τα βασικά χαρακτηριστικά των συμβολοσειρών αυτών. Δώστε και σχετικά παραδείγματα

(b) Ελέγξτε αν η συμβολοσειρά **aabbccac** είναι μέλος της γλώσσας παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης

(c) Είναι η γραμματική LL (1); Δικαιολογήστε την απάντηση σας

(d) Αν δεν είναι LL/1 μετασχηματίστε τη ώστε να έχετε μια ισοδύναμη LL(1) γραμματική

Απάντηση

(a)

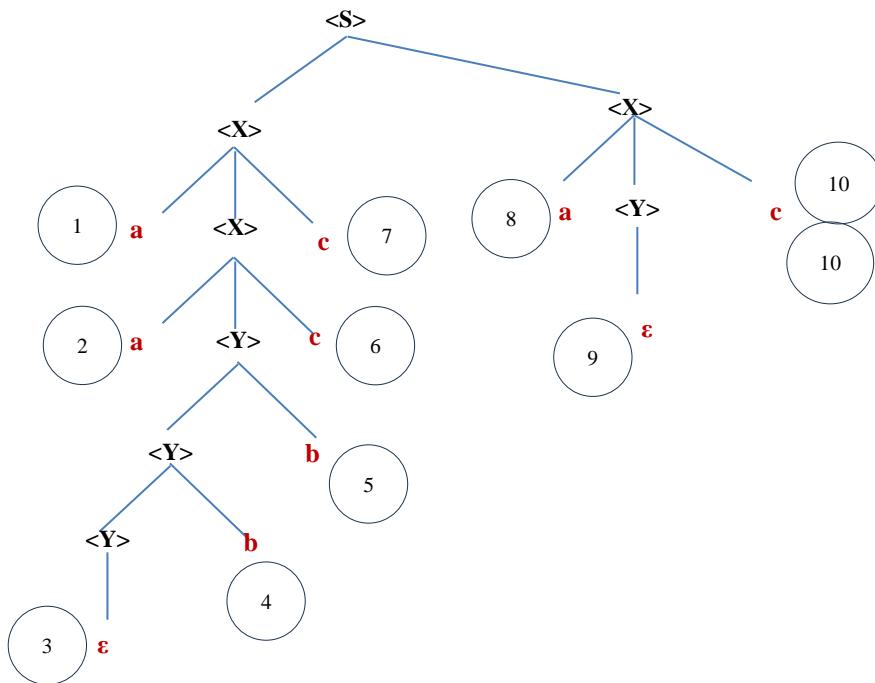
Παραδείγματα Παραγόμενων Συμβολοσειρών

$$\langle S \rangle \Rightarrow \langle Y \rangle \Rightarrow \varepsilon$$
$$\langle S \rangle \Rightarrow \langle Y \rangle \Rightarrow \langle Y \rangle b \Rightarrow \underline{b}$$
$$\langle S \rangle \Rightarrow \langle Y \rangle \Rightarrow \langle Y \rangle b \Rightarrow \langle Y \rangle bb \Rightarrow \underline{bb}$$
$$\langle S \rangle \Rightarrow \langle X \rangle \langle X \rangle \Rightarrow a \langle X \rangle ca \langle Y \rangle c \Rightarrow aa \langle Y \rangle ccac \Rightarrow \underline{aaccac}$$
$$\langle S \rangle \Rightarrow \langle X \rangle \langle X \rangle \Rightarrow a \langle Y \rangle ca \langle Y \rangle c \Rightarrow \underline{acac}$$
$$\langle S \rangle \Rightarrow \langle X \rangle \langle X \rangle \Rightarrow a \langle Y \rangle ca \langle Y \rangle c \Rightarrow \underline{abcabc}$$
$$\langle S \rangle \Rightarrow \langle X \rangle \langle X \rangle \Rightarrow a \langle X \rangle ca \langle Y \rangle c \Rightarrow a \langle X \rangle cac \Rightarrow aa \langle Y \rangle ccac \Rightarrow aa \langle Y \rangle bccac \Rightarrow aa \langle Y \rangle bbccac \Rightarrow \underline{aabbccac}$$

Χαρακτηριστικά Παραγόμενων Συμβολοσειρών

- Αν τερματίζει σε b τότε περιέχει μόνο b
- Εφόσον έχει a, c τα a είναι ίσα με τα c
- Οι συμβολοσειρές που παράγονται δεν τελειώνουν σε a
- Οι συμβολοσειρές που παράγονται δεν ξεκινούν με c

(b) aabbccac

Στοίβα Ολίσθησης-Ελάττωσης

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	aabbccac EOF	Ολίσθηση 1
1	a	abbccac EOF	Ολίσθηση 2
2	aa	bccac EOF	Ελάττωση $<Y> ::= \epsilon$ 3
3	aa<Y>	bccac EOF	Ολίσθηση 4
4	aa<Y>b	bccac EOF	Ελάττωση $<Y> ::= <Y>b$
5	aa<Y>	bccac EOF	Ολίσθηση 5
6	aa<Y>b	ccac EOF	Ελάττωση $<Y> ::= <Y>b$
7	aa<Y>	ccac EOF	Ολίσθηση 6
8	aa<Y>c	cac EOF	Ελάττωση $<X> ::= a<Y>c$
9	a<X>	cac EOF	Ολίσθηση 7
10	a<X>c	ac EOF	Ελάττωση $<X> ::= a<X>c$
11	<X>	ac EOF	Ολίσθηση 8
12	<X>a	c EOF	Ελάττωση $<Y> ::= \epsilon$ 9
13	<X>a<Y>	c EOF	Ολίσθηση 10
14	<X>a<Y>c	EOF	Ελάττωση $<X> ::= a<Y>c$

15	$\langle X \rangle \langle X \rangle$	EOF	Ελάττωση $\langle S \rangle ::= \langle X \rangle \langle X \rangle$
16	$\langle S \rangle$	EOF	Αναγνώριση

(c)

Η Γραμματική ΔΕΝ είναι LL/1 διότι

- Υπάρχουν 2 εναλλακτικοί κανόνες $\langle X \rangle ::= a \langle X \rangle c \mid a \langle Y \rangle c$ που ξεκινούν με το ίδιο τερματικό σύμβολο a
- Στον κανόνα $\langle Y \rangle ::= \langle Y \rangle b \mid \epsilon$ εμφανίζεται αριστερή αναδρομή

(d)

Κάνουμε παραγοντοποίηση στους κανόνες $\langle X \rangle ::= a \langle X \rangle c \mid a \langle Y \rangle c$ και προκύπτει ότι:

$\langle X \rangle ::= a \langle K \rangle$

$\langle K \rangle ::= \langle X \rangle c \mid \langle Y \rangle c$

Κάνουμε απαλοιφή της αριστερής αναδρομής $\langle Y \rangle ::= \langle Y \rangle b$ στο κανόνα και προκύπτει ότι:

$\langle Y \rangle ::= \langle B \rangle$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

Η τελική LL/1 γραμματική είναι:

$\langle S \rangle ::= \langle X \rangle \langle X \rangle \mid \langle Y \rangle$

$\langle X \rangle ::= a \langle K \rangle$

$\langle K \rangle ::= \langle X \rangle c \mid \langle Y \rangle c$

$\langle Y \rangle ::= \langle B \rangle$

$\langle B \rangle ::= b \langle B \rangle \mid \epsilon$

3.5.20 Θέμα 1 Ιούνιος 2020

Δίνεται η BNF γραμματική με επτά κανόνες R1-R7 και τερματικά σύμβολα τα 0 1

R1, R2: $<S> ::= 0 <S> \mid 1 <A>$

R3, R4: $<A> ::= 0 <S> \mid 1 $

R5, R6, R7: $::= 0 \mid 1 \mid \epsilon$

- a. Τι τύπου είναι η γραμματική στην ιεραρχία Chomsky; Αιτιολογήστε την απάντηση σας
- b. Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική;
- c. Μπορείτε να φτιάξετε μια κανονική έκφραση αντίστοιχη της γραμματικής και γιατί; Αν ναι παρουσιάστε τη
- d. Ελέγξτε αν η συμβολοσειρά **010110** είναι μέλος της γλώσσας παρουσιάζοντας τη δημιουργία (derivation) της top-down συντακτικής ανάλυσης της συμβολοσειράς με τον παρακάτω τρόπο (από το παράδειγμα των σελίδων 22-23 των διαφανειών)

$$S_0 \rightarrow [R_1] 0 S_1 \rightarrow [R_2] 0. S_2 \rightarrow [R3] 0.0 S_3 \rightarrow [R6] 0.01$$

όπου S_n είναι συντακτική κατηγορία και $R1, R2, \dots$ είναι παραγωγές (κανόνες)

Απάντηση

- a. Η ιεραρχία Chomsky είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς Συμφραζόμενα

Γραμματικές Με Συμφραζόμενα

Γραμματικές χωρίς περιορισμούς

Η Γραμματική αυτή στην ιεραρχία Chomsky είναι κανονική διότι σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.

b.

Παραδείγματα Συμβολοσειρών που παράγονται από τη γραμματική

$$S \Rightarrow 1 A \Rightarrow 11 B \Rightarrow \underline{11}$$

$$S \Rightarrow 1 A \Rightarrow 11 B \Rightarrow 110 B \Rightarrow \underline{110}$$

$$S \Rightarrow 1 A \Rightarrow 11 B \Rightarrow 111 B \Rightarrow \underline{111}$$

$$S \Rightarrow 0 S \Rightarrow 01 A \Rightarrow 010 S \Rightarrow 0101 A \Rightarrow 01011 B \Rightarrow 010110 B \Rightarrow \underline{010110}$$

$$S \Rightarrow 0 S \Rightarrow 01 A \Rightarrow 010 S \Rightarrow 0101 S \Rightarrow 0101 A \Rightarrow 01011 B \Rightarrow \underline{01011}$$

$$S \Rightarrow 0 S \Rightarrow 00 S \Rightarrow 001 A \Rightarrow 0011 B \Rightarrow \underline{0011}$$

$$S \Rightarrow 0 S \Rightarrow 01 A \Rightarrow 010 S \Rightarrow 0101 A \Rightarrow 01011 B \Rightarrow 010110 B \Rightarrow \underline{010110}$$

Χαρακτηριστικά Συμβολοσειρών που παράγονται από τη γραμματική

Οι συμβολοσειρές που περιγράφει η παραπάνω γραμματική είναι αυτές που:

- περιέχουν **τουλάχιστον δύο διαδοχικούς 1**
- το μήκος τους είναι τουλάχιστον δύο χαρακτήρες με μικρότερη συμβολοσειρά την 11

c. Αφού έχουμε κανονική γραμματική η γλώσσα της μπορεί να περιγραφεί **είτε με NFA είτε με DFA είτε με RE** (Κανονική Έκφραση). Η αντίστοιχη κανονική έκφραση που περιγράφει τη γραμματική είναι: **(0|1)* 11 (0|1)***

d. Η συμβολοσειρά **010110** είναι μέλος της γλώσσας διότι προκύπτει από την ακόλουθη παραγωγή:

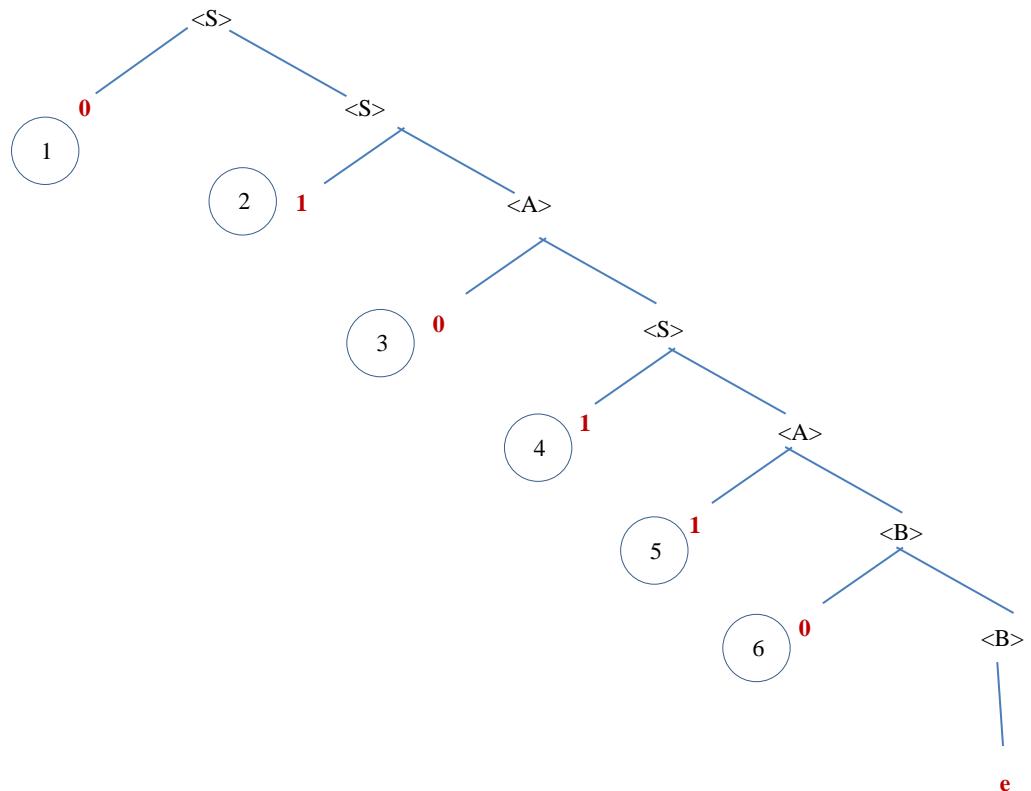
$$R1, R2: <S> ::= 0<S> \mid 1<A>$$

$$R3, R4: <A> ::= 0<S> \mid 1$$

$$R5, R6, R7: ::= 0 \mid 1 \mid \epsilon$$

$S \rightarrow [R1] \underline{0} S \rightarrow [R2] \underline{0} 1 A \rightarrow [R3] 01 \underline{0} S \rightarrow [R2] 010 \underline{1} A \rightarrow [R4] 0101 \underline{1} B \rightarrow [R5] 01011 \underline{0} B \rightarrow [R7] 010110$

Η **Στοίβα Πρόβλεψης Ταιριάσματος –Top-Down** συντακτικής ανάλυσης γίνεται με βάση το Συντακτικό Δέντρο για το **010110**



Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	010110 EOF	Πρόβλεψη <S> ::= <0><S>
1	<S>0	010110 EOF	Ταίριασμα συμβόλου 0
2	<S>	10110 EOF	Πρόβλεψη <S> ::= <1><A>
3	<A>1	10110 EOF	Ταίριασμα συμβόλου 1

4	<A>	0110 EOF	Πρόβλεψη <A> ::= 0 <S>
5	<S>0	0110 EOF	Ταίριασμα συμβόλου 0 3
6	<S>	110 EOF	Πρόβλεψη <S> ::= 1 <A>
7	<A>1	110 EOF	Ταίριασμα συμβόλου 1 4
8	<A>	10 EOF	Πρόβλεψη <A> ::= 1
9	1	10 EOF	Ταίριασμα συμβόλου 1 5
10		0 EOF	Πρόβλεψη ::= 0
11	0	0 EOF	Ταίριασμα συμβόλου 0 6
12		EOF	Πρόβλεψη ::= ε
13	ε	EOF	Αναγνώριση

3.5.21 Θέμα 1 Σεπτέμβριος 2020

Δίνεται η BNF γραμματική με πέντε κανόνες R1-R5 και τερματικά σύμβολα τα x y

R1: $\langle S \rangle ::= \langle A \rangle \langle B \rangle$

R2, R3: $\langle A \rangle ::= x \langle A \rangle y \mid \varepsilon$

R4, R5: $\langle B \rangle ::= y \langle B \rangle \mid y$

- a. Τι τύπου είναι η γραμματική στην **ιεραρχία Chomsky**; Αιτιολογήστε την απάντηση σας
- b. Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Δώστε παραδείγματα αποδεκτών συμβολοσειρών καθώς και τη μικρότερη σε μήκος
- c. Ελέγξτε αν η συμβολοσειρά **xxyyyy** είναι μέλος της γλώσσας παρουσιάζοντας τη **στοίβα Ταιριάσματος-Πρόβλεψης Top Down** συντακτικής ανάλυσης της συμβολοσειράς

Απάντηση

- a. Η **ιεραρχία Chomsky** είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς
Συμφραζόμενα

Γραμματικές Με
Συμφραζόμενα

Γραμματικές χωρίς
περιορισμούς

Μια Γραμματική είναι κανονική αν σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.

Στην **ιεραρχία Chomsky** η **Γραμματική είναι Χωρίς Συμφραζόμενα** διότι στην 1^η παραγωγή, το δεξιά μέλος αρχίζει με μη-τερματικό σύμβολο **άρα η γραμματική δεν είναι κανονική αλλά Χωρίς Συμφραζόμενα**

b.

Παραδείγματα συμβολοσειρών που παράγονται από τη γραμματική

$S \Rightarrow AB \Rightarrow y$. Αυτή είναι και η μικρότερη συμβολοσειρά της γλώσσας

$S \Rightarrow AB \Rightarrow x \langle A \rangle yy \Rightarrow xy$

$S \Rightarrow AB \Rightarrow x \langle A \rangle yy \langle B \rangle \Rightarrow xy$

$S \Rightarrow AB \Rightarrow x \langle A \rangle yy \langle B \rangle \Rightarrow xx \langle A \rangle yyyy \Rightarrow xxxx$

$S \Rightarrow AB \Rightarrow xAy yB \Rightarrow xxAyy yyB \Rightarrow xxxx yyyy \Rightarrow xxxxxxxx$

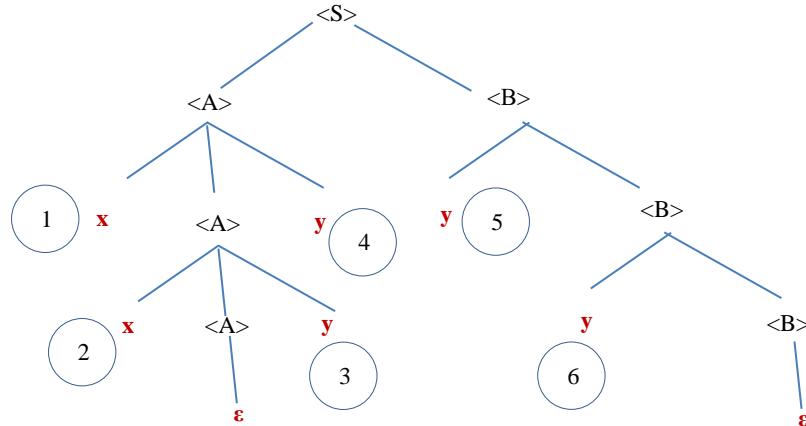
$S \Rightarrow AB \Rightarrow xAy yB \Rightarrow xxAyy yyB \Rightarrow xxxAyyy yyyyB \Rightarrow xxxxxxxx$

Χαρακτηριστικά των συμβολοσειρών που παράγονται από τη γραμματική

- Οι συμβολοσειρές τερματίζουν σε γ
- Έχουν μήκος τουλάχιστον 1 χαρακτήρα και η μικρότερη συμβολοσειρά είναι η γ
- Αν υπάρχουν x και y τα x προηγούνται των y
- Τα y είναι περισσότερα από τα x

c.

Η Στοίβα Πρόβλεψης Ταιριάσματος –Top-Down συντακτικής ανάλυσης γίνεται με βάση το Συντακτικό Δέντρο για το **xxyyyy**



Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	xxyyyy EOF	Πρόβλεψη $\text{<S>} ::= \text{<A>}$
1	<A>	xxyyyy EOF	Πρόβλεψη $\text{<A>} ::= \text{x <A> y}$
2	y<A> <u>x</u>	<u>xx</u> yyyy EOF	Ταίριασμα συμβόλου x 1
3	y<A>	<u>xy</u> yyy EOF	Πρόβλεψη $\text{<A>} ::= \text{x <A> y}$
4	y<A> y<A> <u>x</u>	<u>xy</u> yyy EOF	Ταίριασμα συμβόλου x 2
5	y<A> y<A>	<u>yy</u> yy EOF	Πρόβλεψη $\text{<A>} ::= \epsilon$
6	y<A> y	<u>yy</u> y EOF	Ταίριασμα συμβόλου y 3
7	y<A>	<u>yy</u> EOF	Πρόβλεψη $\text{<A>} ::= \epsilon$
8	 <u>y</u>	<u>y</u> y EOF	Ταίριασμα συμβόλου y 4
9		<u>y</u> y EOF	Πρόβλεψη $\text{} ::= \text{y }$
10	y	<u>y</u> y EOF	Ταίριασμα συμβόλου y 5
11		<u>y</u> EOF	Πρόβλεψη $\text{} ::= \text{y }$
12	 <u>y</u>	<u>y</u> EOF	Ταίριασμα συμβόλου y 6
13		EOF	Πρόβλεψη $\text{} ::= \epsilon$
14	ϵ	EOF	Αναγνώριση

Πιθανή Ερώτηση

Να εξετάσετε αν η γραμματική είναι LL/1 και αν όχι να τη μετατρέψετε σε LL/1

R1: $\langle S \rangle ::= \langle A \rangle \langle B \rangle$

R2, R3: $\langle A \rangle ::= x \langle A \rangle y \mid \epsilon$

R4, R5: $\langle B \rangle ::= \textcolor{red}{y} \langle B \rangle \mid \textcolor{red}{y}$

Απάντηση

Η γραμματική αυτή δεν είναι LL(1) διότι έχουμε 2 εναλλακτικούς κανόνες R4, R5 που αρχίζουν με το ίδιο τερματικό σύμβολο y.

Η διόρθωση είναι η εξής:

$\langle B \rangle ::= y \langle M \rangle$

$\langle M \rangle ::= \langle B \rangle \mid \epsilon$

Η Τελική LL(1) γραμματική είναι:

R1: $\langle S \rangle ::= \langle A \rangle \langle B \rangle$

R2, R3: $\langle A \rangle ::= x \langle A \rangle y \mid \epsilon$

R4: $\langle B \rangle ::= y \langle M \rangle$

R5, R6: $\langle M \rangle ::= \langle B \rangle \mid \epsilon$

3.5.22 Θέμα 1 Φεβρουάριος 2021

Δίνεται η BNF γραμματική που φαίνεται στο τέλος της εκφώνησης με τερματικά σύμβολα τα a b c d

- a. Τι τύπου είναι η γραμματική στην ιεραρχία Chomsky; Είναι LL(1); Αιτιολογήστε την απάντηση σας
- b. Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Δώστε παραδείγματα αποδεκτών συμβολοσειρών καθώς και τη μικρότερη σε μήκος
- c. Είναι το **aaabbb** είναι μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη στοίβα Ολίσθησης-Ελάττωσης συντακτικής ανάλυσης για το string

$\langle X \rangle ::= \langle A \rangle \mid \langle B \rangle$

$\langle A \rangle ::= a \langle A \rangle b \mid \epsilon$

$\langle B \rangle ::= c \langle B \rangle d \mid \epsilon$

Απάντηση

a.

Η ιεραρχία Chomsky είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς Συμφραζόμενα

Γραμματικές Με Συμφραζόμενα

Γραμματικές χωρίς περιορισμούς

Η Γραμματική είναι κανονική διότι στο δεξί μέρος κάθε κανόνα να υπάρχει το πολύ 1 μη τερματικό σύμβολο και αυτό να είναι το τελευταίο σύμβολο του κανόνα

Η Γραμματική είναι κανονική διότι σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.

a. Η γραμματική αυτή ΔΕΝ είναι Κανονική γιατί π.χ.. στον κανόνα $\langle A \rangle ::= a \langle A \rangle b$ το ΜΗ τερματικό σύμβολο $\langle A \rangle$ ΔΕΝ είναι στο τέλος του κανόνα.

Η δοθείσα γραμματική στην ιεραρχία Chomsky είναι Γραμματική Χωρίς Συμφραζόμενα

Η Γραμματική αυτή είναι LL/1 διότι:

α) δεν υπάρχει σε κανένα κανόνα αριστερή αναδρομή

β) δεν υπάρχουν κανόνες που να αρχίζουν με τον ίδιο τερματικό χαρακτήρα

b.

Παραδείγματα Συμβολοσειρών που παράγονται από τη γραμματική αυτή

X⇒A⇒ ϵ

Y⇒B⇒ ϵ

X⇒A⇒aAb⇒**ab**

X⇒A⇒aAb⇒aaAbb⇒**aabb**

Y⇒B⇒cBd⇒**cd**

Y⇒B⇒cBd⇒ccBdd⇒**ccdd**

Η μικρότερη σε μήκος συμβολοσειρά είναι η κενή.

Χαρακτηριστικά Συμβολοσειρών που παράγονται από τη γραμματική αυτή

- 1) Έχουν άρτιο μήκος
- 2) Περιέχουν είτε a, b είτε c, d είτε είναι η κενή συμβολοσειρά
- 3) Αν η συμβολοσειρά περιέχει a, b τα a προηγούνται των b
- 4) Αν η συμβολοσειρά περιέχει c, d τα c προηγούνται των d
- 5) Το πλήθος των a είναι ίσο με το πλήθος των b
- 6) Το πλήθος των c είναι ίσο με το πλήθος των d

c)

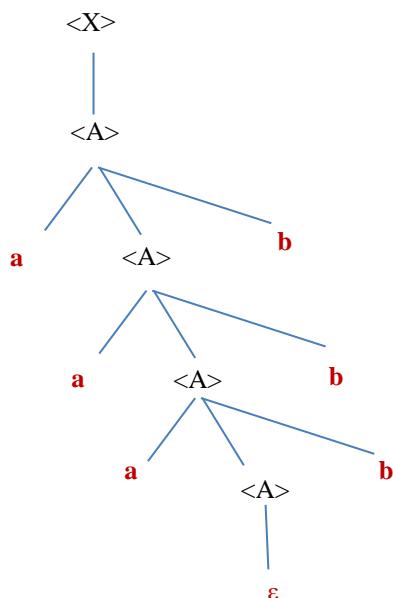
$<X> ::= <A> \mid $

$<A> ::= a <A> b \mid \epsilon$

$::= c d \mid \epsilon$

Στοίβα Ολίσθησης Ελάττωσης

c. aaabbb



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	aaabbb EOF	Ολίσθηση
1	a	aabbb EOF	Ολίσθηση
2	aa	bbb EOF	Ολίσθηση
3	aaa	bb EOF	Ολίσθηση
4	aaa	bb EOF	Ελάττωση $\langle A \rangle ::= \epsilon$
5	aaa $\langle A \rangle$	bb EOF	Ολίσθηση
6	aaa $\langle A \rangleb$	bb EOF	Ελάττωση $\langle A \rangle ::= a \langle A \rangle b$
7	aa $\langle A \rangle$	bb EOF	Ολίσθηση
8	aa $\langle A \rangleb$	bb EOF	Ελάττωση $\langle A \rangle ::= a \langle A \rangle b$
9	a $\langle A \rangle$	bb EOF	Ολίσθηση
10	a $\langle A \rangleb$	EOF	Ελάττωση $\langle A \rangle ::= a \langle A \rangle b$
11	$\langle A \rangle$	EOF	Ελάττωση $\langle X \rangle ::= \langle A \rangle$
12	$\langle X \rangle$	EOF	Αναγνώριση

3.5.23 Θέμα 1 Σεπτέμβριος 2021

Σας δίνονται οι παρακάτω τρεις απλές γραμματικές σε συμβολισμό BNF.

$<A> ::= 0 <A> \mid 1 <A> \mid 0$

$::= \mid 1 \mid 0$

$<C> ::= 0 <C> 0 \mid 1 <C> 1 \mid 0 \mid 1$

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

Απάντηση

a.

Παραδείγματα συμβολοσειρών από την 1^η γραμματική

$A \Rightarrow 0, A \Rightarrow 0A \Rightarrow 00, A \Rightarrow 1A \Rightarrow 10, A \Rightarrow 0A \Rightarrow 01A \Rightarrow 010, A \Rightarrow 1A \Rightarrow 11A \Rightarrow 110A \Rightarrow 1101A \Rightarrow 11010$

Η 1^η γραμματική παράγει συμβολοσειρές που τελειώνουν σε 0

Παραδείγματα συμβολοσειρών από τη 2^η γραμματική

$B \Rightarrow 0, B \Rightarrow 1, B \Rightarrow BB \Rightarrow 00, B \Rightarrow BB \Rightarrow 11, B \Rightarrow BB \Rightarrow 10, B \Rightarrow BB \Rightarrow 01, B \Rightarrow BB \Rightarrow BBB \Rightarrow 101, B \Rightarrow BB \Rightarrow BBB \Rightarrow BBB \Rightarrow 0110$

Η 2^η γραμματική παράγει οποιαδήποτε συμβολοσειρά από 0, 1

Παραδείγματα συμβολοσειρών από την 3^η γραμματική

$C \Rightarrow 0, C \Rightarrow 1, C \Rightarrow 0C0 \Rightarrow 010, C \Rightarrow 0C0 \Rightarrow 000, C \Rightarrow 0C0 \Rightarrow 010, C \Rightarrow 0C0 \Rightarrow 01C10 \Rightarrow 01010, C \Rightarrow 0C0 \Rightarrow 01C10 \Rightarrow 01110,$

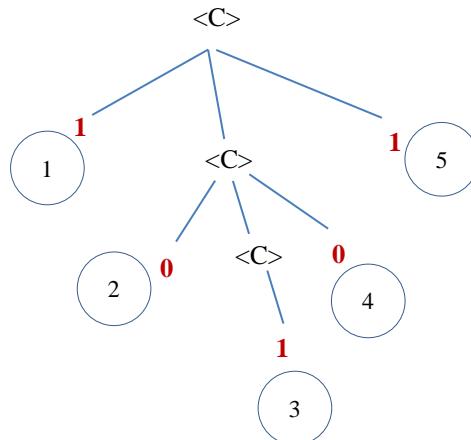
$C \Rightarrow 1C1 \Rightarrow 11C11 \Rightarrow 11111$

Η 3^η Γραμματική παράγει παλινδρομικές συμβολοσειρές περιττού μήκους

b.

- Από την 1^η γραμματική παράγονται όσες τελειώνουν σε 0 και συγκεκριμένα οι **110110, 0100010, 110110110**
- Από τη 2^η γραμματική παράγονται ΟΛΕΣ οι συμβολοσειρές άρα στη 2^η γραμματική αντιστοιχίζονται όλες οι συμβολοσειρές
- Από την 3^η γραμματική παράγονται οι παλινδρομικές συμβολοσειρές περιττού μήκους δηλ οι **0100010, 11111**

c. **10101**



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	10101 EOF	Ολίσθηση 1
1	1	0101 EOF	Ολίσθηση 2
2	10	101 EOF	Ολίσθηση 3
3	101	01 EOF	Ελάττωση $<C> ::= 1$
4	10<C>	01 EOF	Ολίσθηση 4
5	10<C>0	1 EOF	Ελάττωση $<C> ::= 0 <C> 0$
6	1<C>	1 EOF	Ολίσθηση 5
7	1<C>1	EOF	Ελάττωση $<C> ::= 1 <C> 1$
8	<C>	EOF	Αναγνώριση

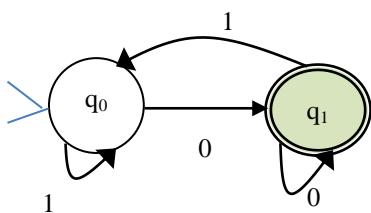
δ.

Μια Γραμματική είναι κανονική αν σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο

- Η 1^η Γραμματική είναι κανονική διότι σε όλες τις παραγωγές τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.
- Η 2^η Γραμματική ΔΕΝ είναι κανονική διότι ΔΕΝ αρχίζουν με τερματικό σύμβολο όλες τις παραγωγές της. Π.χ. ο κανόνας $\langle B \rangle ::= \langle B \rangle \langle B \rangle$ ΔΕΝ αρχίζει με τερματικό σύμβολο
- Η 3^η Γραμματική είναι κανονική διότι σε όλες τις παραγωγές τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.

ε.

- Η πρώτη κατά σειρά κανονική γραμματική είναι η $\langle A \rangle ::= 0\langle A \rangle \mid 1\langle A \rangle \mid 0$ και περιγράφεται από την ακόλουθη κανονική έκφραση: $(0|1)^*0$
- Το DFA που περιγράφει τη γλώσσα αυτή



- Το DFA είναι το διάγραμμα **καταστάσεων-μεταβάσεων**. Από το διάγραμμα **καταστάσεων-μεταβάσεων** προκύπτει πάντα ο πίνακας **καταστάσεων-μεταβάσεων**
- Ο πίνακας **καταστάσεων-μεταβάσεων** για το DFA που περιγράφει τη γλώσσα αυτή είναι ο ακόλουθος:

Τρέχουσα Κατάσταση	Χαρακτήρας που διαβάζεται	Νέα Κατάσταση	Αποδοχή token
q_0	1	q_0	OXI
q_0	0	q_1	NAI
q_1	0	q_1	NAI
q_1	1	q_0	OXI

στ.

Ελεγγος 1^{ης} Γραμματικής αν είναι LL/1

$\langle A \rangle ::= 0\langle A \rangle \mid 1\langle A \rangle \mid 0$

Η 1^η γραμματική ΔΕΝ είναι LL/1 γιατί έχει εναλλακτικούς κανόνες $\langle A \rangle ::= 0\langle A \rangle$ και $\langle A \rangle ::= 0$ που αρχίζουν με το ίδιο τερματικό σύμβολο 0. Η διόρθωση της σε LL/1 είναι η ακόλουθη:

$\langle A \rangle ::= 0\langle K \rangle \mid 1\langle A \rangle$
 $\langle K \rangle ::= \langle A \rangle \mid \epsilon$
Αυτή είναι η τελική LL/1 γραμματική

Περιγραφή Διόρθωσης: Βγάζουμε το 0 ως κοινό παράγοντα από κάθε κανόνα που εμφανίζεται και προσθέτουμε μετά τον κοινό παράγοντα ένα νέο MH Τερματικό Σύμβολο π.χ. το <K>. Στη συνέχεια από το νέο MH Τερματικό Σύμβολο <K> πηγαίνουμε σε κάθε κανόνα που αρχίζει με τον κοινό παράγοντα 0, γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα δηλ. μετά το 0. Στον 1^ο κανόνα μετά το 0 υπάρχει το <A>, στον 3^ο κανόνα μετά το 0 δεν υπάρχει τίποτα και βάζουμε ε.

Παρατήρηση

Αν ζητούσε να μετατρέψουμε σε LL/1 όλες τις γραμματικές που δεν είναι θα κάναμε τα εξής:

Έλεγχος 2^{ης} Γραμματικής αν είναι LL/1

 ::= | 1| 0

Η 2^η κατά σειρά γραμματική ::= | 1| 0 έχει αριστερή αναδρομή που απαλείφεται ως εξής:

 ::= 1 <M> | 0 <M>

<M> ::= <M> | ε

Αυτή είναι η τελική LL/1 γραμματική

Περιγραφή Διόρθωσης: Πηγαίνουμε σε κάθε κανόνα που ΔΕΝ έχει αριστερή αναδρομή και βάζουμε στο τέλος του ένα νέο MH Τερματικό Σύμβολο π.χ. το <M>. Στη συνέχεια από το νέο MH Τερματικό Σύμβολο <M> πηγαίνουμε σε κάθε κανόνα που έχει αριστερή αναδρομή και γράφουμε ότι υπάρχει μετά την αναδρομή ακολουθόμενο από το νέο σύμβολο ή κενό

Έλεγχος 3^{ης} Γραμματικής αν είναι LL/1

<C> ::= 0 <C> 0 | 1 <C> 1 | 0 | 1

Η 3^η κατά σειρά γραμματική <C> ::= 0 <C> 0 | 1 <C> 1 | 0 | 1 έχει εναλλακτικούς κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο και η διόρθωση τους είναι η ακόλουθη:

<C> ::= 0 <X> | 1 <Y>

<X> ::= <C> 0 | ε

<Y> ::= <C> 1 | ε

Αυτή είναι η τελική LL/1 γραμματική

Περιγραφή Διόρθωσης: Βγάζουμε το 0 και το 1 ως κοινό παράγοντα από κάθε κανόνα που εμφανίζεται και προσθέτουμε μετά τον κοινό παράγοντα ένα νέο MH Τερματικό Σύμβολο π.χ. το <X> και το <Y>. Στη συνέχεια από το νέο MH Τερματικό Σύμβολο <X> πηγαίνουμε σε κάθε κανόνα που αρχίζει με τον κοινό παράγοντα 0, γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα δηλ. μετά το 0. Στη συνέχεια από το νέο MH Τερματικό Σύμβολο <Y> πηγαίνουμε σε κάθε κανόνα που αρχίζει με τον κοινό παράγοντα 1, γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα δηλ. μετά το 1.

Ελεγγος Β Γραμματικής αν είναι LL/1

$::= \mid 1 \mid 0$

Η 2^η γραμματική ΔΕΝ είναι LL/1 γιατί παρουσιάζει **αριστερή αναδρομή στον κανόνα** $::= $. Η διόρθωση της είναι η ακόλουθη:

$::= 1 <N> \mid 0 <N>$

$<N> ::= <N> \mid \epsilon$

Αυτή είναι η τελική LL/1 γραμματική

Διόρθωση: πηγαίνουμε σε κάθε κανόνα που δεν έχει αναδρομή και στο τέλος του προσθέτουμε ένα νέο Μη Τερματικό Σύμβολο π.χ. το $<N>$. Από το νέο Μη Τερματικό Σύμβολο $<N>$ πηγαίνουμε σε κάθε κανόνα που είχε αναδρομή και γράφουμε ότι υπάρχει μετά την αναδρομή ακόλουθούμενο από το νέο Μη Τερματικό Σύμβολο $<N>$. Προσθέτουμε και ένα κανόνα που οδηγεί στο ϵ .

Ελεγγος Γ Γραμματικής αν είναι LL/1

$<C> ::= 0 <C> 0 \mid 1 <C> 1 \mid 0 \mid 1$

Η 3^η γραμματική ΔΕΝ είναι LL/1 γιατί υπάρχουν εναλλακτικοί κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο. Αυτοί είναι οι κανόνες $<C> ::= 0 <C> 0$ και $<C> ::= 0$ που αρχίζουν με 0 και οι κανόνες $<C> ::= 1 <C> 1$ και $<C> ::= 1$ που αρχίζουν με 1

$<C> ::= 0 <T> \mid 1 <X>$

$<T> ::= <C> 0 \mid \epsilon$

$<X> ::= <C> 1 \mid \epsilon$

Αυτή είναι η τελική LL/1 γραμματική

Διόρθωση: Βγάζουμε το 0 κοινό παράγοντα από κάθε κανόνα που εμφανίζεται και προσθέτουμε μετά από αυτό ένα νέο ΜΗ Τερματικό Σύμβολο π.χ. το $<T>$. Βγάζουμε και το 1 κοινό παράγοντα από κάθε κανόνα που εμφανίζεται και προσθέτουμε μετά από αυτό ένα νέο ΜΗ Τερματικό Σύμβολο π.χ. το $<X>$.

Στη συνέχεια από το νέο ΜΗ Τερματικό Σύμβολο $<T>$ πηγαίνουμε σε κάθε κανόνα που αρχίζει με τον κοινό παράγοντα 0, γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα δηλ. μετά το 0. Στον 1^ο κανόνα μετά το 0 υπάρχει το $<C> 0$, στον 3^ο κανόνα μετά το 0 δεν υπάρχει τίποτα. Στη συνέχεια από το νέο ΜΗ Τερματικό Σύμβολο $<X>$ πηγαίνουμε σε κάθε κανόνα που αρχίζει με τον κοινό παράγοντα 1, γράφουμε ότι υπάρχει μετά τον κοινό παράγοντα δηλ. μετά το 1. Στο 2^ο κανόνα μετά το 0 υπάρχει το $<C> 1$, στον 4^ο κανόνα μετά το 0 δεν υπάρχει τίποτα

Υπενθύμιση

Αριστερή Αναδρομή

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$

Η απαλοιφή της Αριστερής Αναδρομής γίνεται ως εξής:

$A \rightarrow \beta_1 B \mid \beta_2 B \mid \dots \mid \beta_m B$

$B \rightarrow \alpha_1 B \mid \alpha_2 B \mid \dots \mid \alpha_n B \mid \epsilon$

Κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο

$A \rightarrow \alpha \beta_1 | \alpha \beta_2 | \dots | \alpha \beta_n$

Η Αριστερή Παραγοντοποίηση γίνεται ως εξής:

$A \rightarrow \alpha B$

$B \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$

3.5.24 Θέμα 4 Ιούνιος 2021

Μια συνθηματική γλώσσα απλών προτάσεων περιγράφεται από την παρακάτω BNF γραμματική με τερματικά σύμβολα τα **a b d S** όπου το **S** συμβολίζει το χαρακτήρα «κενό» (space)

```
<πρόταση> ::= <λέξη> | <πρόταση> S <λέξη>  
<λέξη> ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή>  
<συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | a <βάση> | a <σύμφωνο>  
<βάση> ::= <σύμφωνο> a  
<σύμφωνο> ::= b | d
```

(a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε μερικά βασικά χαρακτηριστικά των συμβολοσειρών αυτών, παρουσιάζοντας και παραδείγματα

(b) Παρουσιάζοντας τη Στοίβα Ολίσθησης – Ελάττωσης (Bottom-Up) να αποφανθείτε αν η πρόταση **da|_ad** (ή αλλιώς **daSad**) είναι μέλος της συνθηματικής γλώσσας

(c) Είναι η γραμματική LL(1); Αιτιολογήστε την απάντηση σας, υποδεικνύοντας και τα σχετικά σημεία της γραμματικής που δικαιολογούν τον ισχυρισμό σας. Αν η γραμματική δεν είναι LL(1), τι μετασχηματισμοί πρέπει να γίνουν ώστε να δημιουργηθεί μια ισοδύναμη LL(1) γραμματική;

Απάντηση

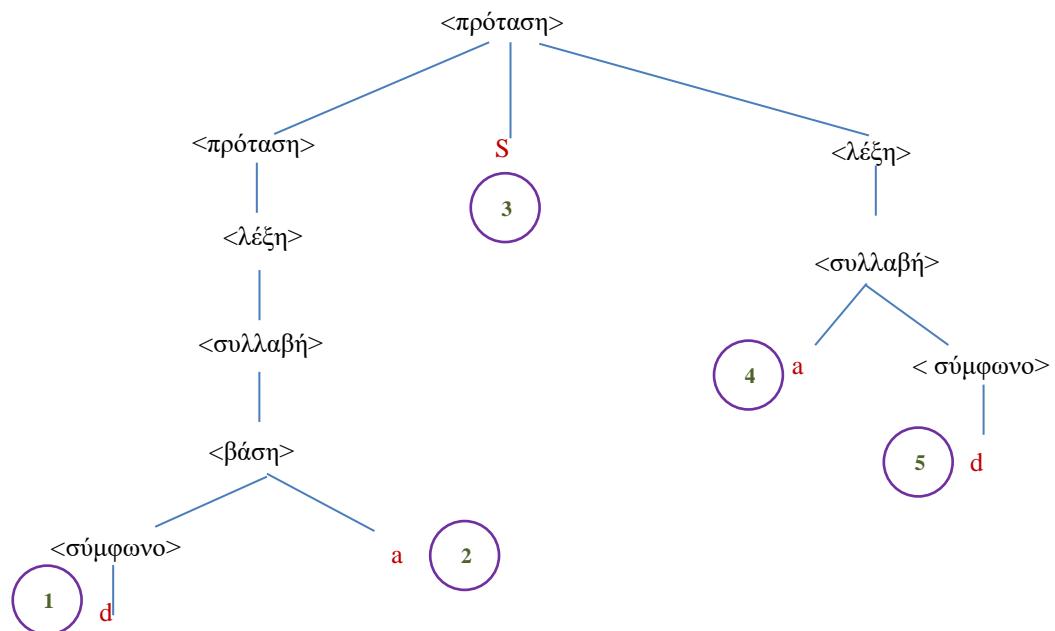
Παραδείγματα Συμβολοσειρών είναι

```
<πρόταση> → <λέξη> → <συλλαβή> → <βάση> → <σύμφωνο> a → ba ń da  
<πρόταση> → <πρόταση> S <λέξη> → <λέξη> S <λέξη> → <συλλαβή> <λέξη> <συλλαβή> S <λέξη> → a <βάση> <συλλαβή>  
<συλλαβή> S <λέξη> → a <σύμφωνο> a <βάση> <βάση> S <συλλαβή> → adabadaSba
```

Επιμέρους παραγωγές είναι οι ακόλουθες: **da, ba|_ba, ba|_da, da|_da, da|_ad, ad|_abadaba, ad, ab.**

- Το βασικό χαρακτηριστικό των συμβολοσειρών της γλώσσας είναι ότι δεν επιτρέπεται να έχουμε συνεχόμενα b ή d ή a ή S. δηλ. υπάρχει διαδοχική εναλλαγή χαρακτήρων
- Επίσης στις συμβολοσειρές της γραμματικής δεν ανήκει η κενή συμβολοσειρά και το ελάχιστο μήκος λέξης είναι 2

(b) Η Στοίβα Ολίσθησης – Ελάττωσης Bottom-UP Στοίβα Συντακτικής Ανάλυσης για την πρόταση **da** | **ad** είναι η ακόλουθη:



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ε	da ad EOF	Ολίσθηση 1
1	d	a ad EOF	Ελάττωση <σύμφωνο> ::= d
2	<σύμφωνο>	a ad EOF	Ολίσθηση 2
3	<σύμφωνο> a	ad EOF	Ελάττωση <βάση> ::= <σύμφωνο> a
4	<βάση>	ad EOF	Ελάττωση <συλλαβή> ::= <βάση>
5	<συλλαβή>	ad EOF	Ελάττωση <λέξη> ::= <συλλαβή>
6	<λέξη>	ad EOF	Ελάττωση <πρόταση> ::= <λέξη>
7	<πρόταση>	ad EOF	Ολίσθηση 3
8	<πρόταση> _	ad EOF	Ολίσθηση 4
9	<πρόταση> _ a	d EOF	Ολίσθηση 5
10	<πρόταση> _ ad	EOF	Ελάττωση <σύμφωνο> ::= d
11	<πρόταση> _ a <σύμφωνο>	EOF	Ελάττωση <συλλαβή> ::= a <σύμφωνο>
12	<πρόταση> _ <συλλαβή>	EOF	Ελάττωση <λέξη> ::= <συλλαβή>
13	<πρόταση> _ <λέξη>	EOF	Ελάττωση <πρόταση> ::= <πρόταση> _ <λέξη>
14	<πρόταση>	EOF	Αναγνώριση

(c)

Αριστερή Αναδρομή	Κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο
$A \rightarrow A\alpha_1 A\alpha_2 \dots A\alpha_n \beta_1 \beta_2 \dots \beta_m$	$A \rightarrow a\beta_1 a\beta_2 \dots a\beta_n$
Η απαλοιφή της Αριστερής Αναδρομής γίνεται ως εξής:	Η Αριστερή Παραγοντοποίηση γίνεται ως εξής:
$A \rightarrow \beta_1 B \beta_2 B \dots \beta_m B$	$A \rightarrow aB$
$B \rightarrow \alpha_1 B \alpha_2 B \dots \alpha_n B \varepsilon$	$B \rightarrow \beta_1 \beta_2 \dots \beta_n$

Η γραμματική δεν είναι LL/1 διότι:

- Ο κανόνας **<πρόταση>** ::= <λέξη> | **<πρόταση> S <λέξη>** παρουσιάζει **αριστερή αναδρομή** και διορθώνεται ως εξής:
<πρόταση> ::= <λέξη> **<Κ>**
<Κ> ::= S <λέξη> **<Κ>** | ε
- Ο κανόνας <λέξη> ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή> δεν έχει πρόβλημα και παραμένει όπως είναι
- Ο κανόνας <συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | a <βάση> | a <σύμφωνο> χρειάζεται **αριστερή παραγοντοποίηση** και διορθώνεται ως εξής:
<συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | a **<Ν>**
<Ν> ::= βάση | <σύμφωνο>
- Οι τελευταίοι 2 κανόνες:
 - <βάση> ::= <σύμφωνο> a
 - <σύμφωνο> ::= b | d

δεν έχουν πρόβλημα και παραμένουν όπως είναι.

Τελική Διορθωμένη LL/1 Γραμματική

- <πρόταση> ::= <λέξη> **<Κ>**
- <Κ>** ::= S <λέξη> **<Κ>** | ε
- <λέξη> ::= <συλλαβή> | <συλλαβή> <λέξη> <συλλαβή>
- <συλλαβή> ::= <βάση> | <βάση> <σύμφωνο> | a <Ν>
- <Ν> ::= βάση | <σύμφωνο>
- <βάση> ::= <σύμφωνο> a
- <σύμφωνο> ::= b | d

3.5.25 Θέμα 1 Ιούνιος 2022

Δίνεται η παρακάτω BNF με αλφάβητο a b

$\langle S \rangle ::= a \langle A \rangle \mid b \langle B \rangle$

$\langle A \rangle ::= a \langle S \rangle \mid a$

$\langle B \rangle ::= b \langle S \rangle \mid b$

(a) Περιγράψτε όλα τα χαρακτηριστικά των μελών της γλώσσας που παράγεται από την παραπάνω γραμματική. Παρουσιάστε παραδείγματα καθώς και το/τα μικρότερο/a σε μήκος μέλος/η

(b) Είναι το **aabbaa** μέλος της γλώσσας; Απαντήστε παρουσιάζοντας τη **Στοίβα Ολίσθησης – Ελάττωσης Bottom – Up συντακτικής ανάλυσης για το string**

(c) Τι τύπου είναι η γραμματική στην **ιεραρχία Chomsky**; Αιτιολογήστε την απάντηση σας

(d) Μπορείτε να φτιάξετε μια κανονική έκφραση αντίστοιχη της γραμματικής και γιατί; Αν ναι παρουσιάστε τη

(e) Επιπλέον υπάρχει ντερμινιστικό πεπερασμένο αυτόματο αντίστοιχο της κανονικής έκφρασης; Αν ναι παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων του αυτού μηχανής

(f) Είναι η γραμματική **LL(1)**; Αιτιολογήστε την απάντηση σας. Αν δεν είναι LL(1) μετασχηματίστε τη σε μια LL(1) γραμματική

Απάντηση

(a)

Ενδεικτικές Παραγωγές

Παραδείγματα συμβολοσειρών που παράγει η συγκεκριμένη γραμματική είναι τα ακόλουθα:

$S \Rightarrow aA \Rightarrow aa$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aaaA \Rightarrow aaaa$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aaaA \Rightarrow aaaaS \Rightarrow aaaaaA \Rightarrow aaaaaaa$

$S \Rightarrow bB \Rightarrow bb$

$S \Rightarrow bB \Rightarrow bbS \Rightarrow bbbB \Rightarrow bbbb$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabb$

$S \Rightarrow bB \Rightarrow bbS \Rightarrow bbaA \Rightarrow bbaa$

$S \Rightarrow bB \Rightarrow bbS \Rightarrow bbbB \Rightarrow bbbbS \Rightarrow bbbbA \Rightarrow bbbbbaS \Rightarrow bbbbbaaA \Rightarrow bbbbaaaa$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaA \Rightarrow aabbaa$

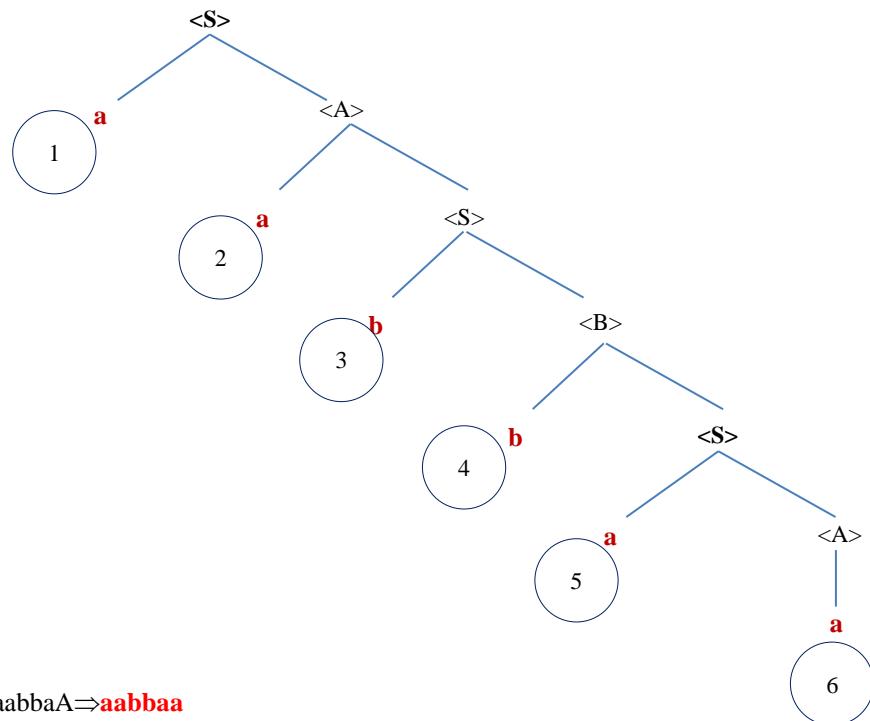
Η μικρότερη συμβολοσειρά της γραμματικής έχει μήκος 2 χαρακτήρες και οι 2 μικρότερες συμβολοσειρές είναι οι aa και bb

Χαρακτηριστικά Συμβολοσειρών

Η γραμματική αυτή δημιουργεί συμβολοσειρές με τα ακόλουθα χαρακτηριστικά:

- Συμβολοσειρές από a και/ή b άρτιου μήκους
- Το πλήθος των a και το πλήθος των b είναι άρτιο σε κάθε συμβολοσειρά
- Το μικρότερο μήκος συμβολοσειράς είναι 2

(b)



$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaA \Rightarrow \text{aabbaa}$

Η Στοίβα Ολίσθησης-Ελάττωσης ή εναλλακτικά **bottom-up parser** ξεκινά με τη συμβολοσειρά που πρέπει να αναγνωρίσει στην είσοδο του και με μια κενή στοίβα.

Αν η συμβολοσειρά εισόδου είναι έγκυρη δηλ. ανήκει στη γλώσσα **καταλήγει πάντα σε αναγνώριση** και έχει **διαβάσει όλη τη συμβολοσειρά εισόδου** και έχει τοποθετήσει στη στοίβα το αρχικό σύμβολο της γραμματικής, αλλιώς όχι

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	aabbaa EOF	Ολίσθηση
1	a	abbaa EOF	Ολίσθηση
2	aa	bbaa EOF	Ολίσθηση
3	aab	baa EOF	Ολίσθηση
4	aabb	aa EOF	Ολίσθηση
5	aabba	a EOF	Ολίσθηση
6	aabbaa	EOF	Ελάττωση $\langle A \rangle ::= a$
7	aabba<A>	EOF	Ελάττωση $\langle S \rangle ::= a \langle A \rangle$
8	aabb<S>	EOF	Ελάττωση $\langle B \rangle ::= b \langle S \rangle$
9	aab	EOF	Ελάττωση $\langle S \rangle ::= b \langle B \rangle$
10	aa<S>	EOF	Ελάττωση $\langle A \rangle ::= a \langle S \rangle$
11	a<A>	EOF	Ελάττωση $\langle S \rangle ::= a \langle A \rangle$
12	<S>	EOF	Αναγνώριση

(c) Η **ιεραρχία Chomsky** είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς
Συμφραζόμενα

Γραμματικές Με
Συμφραζόμενα

Γραμματικές χωρίς
περιορισμούς

Η Γραμματική αυτή στην ιεραρχία Chomsky είναι **κανονική** διότι σε **όλες** τις παραγωγές, τα δεξιά μέλη **αρχίζουν με τερματικό σύμβολο**

Εναλλακτικά:

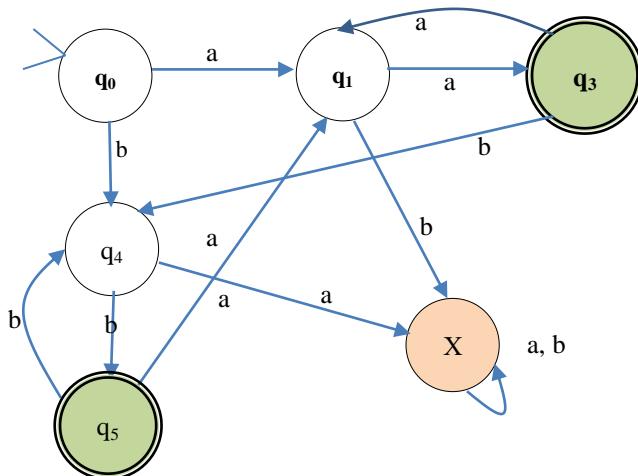
Η Γραμματική αυτή στην ιεραρχία Chomsky είναι **κανονική** διότι:

- στο δεξί του μέρος κάθε κανόνα υπάρχει το πολύ 1 μη τερματικό σύμβολο (δηλ. είτε ένα είτε κανένα)
- αυτό το μη τερματικό σύμβολο βρίσκεται στο τέλος του κανόνα

(d) Η γραμματική είναι **κανονική συνεπώς κατασκευάζεται κανονική έκφραση που να την περιγράφει που είναι η εξής:**

$$((aa)^+ | (bb)^+)^+$$

(e) Το **ντερμινιστικό πεπερασμένο αυτόματο (DFA)** που είναι αντίστοιχο της κανονικής έκφρασης είναι το ακόλουθο:



Παρατήρηση: Όταν μια γραμματική είναι κανονική τότε μπορούμε να κατασκευάσουμε κανονική έκφραση που να την περιγράφει καθώς και DFA και NFA που να την περιγράφουν

f) Η γραμματική δεν είναι LL/1 διότι χρειάζεται αριστερή παραγοντοποίηση στους κανόνες. $\langle A \rangle ::= a \langle S \rangle \mid a$ και $\langle B \rangle ::= b \langle S \rangle \mid b$

Αριστερή παραγοντοποίηση στον κανόνα $\langle A \rangle ::= a \langle S \rangle \mid a$

$\langle A \rangle ::= a \langle X \rangle$

$\langle X \rangle ::= \langle S \rangle \mid \varepsilon$

Αριστερή παραγοντοποίηση στον κανόνα $\langle B \rangle ::= b \langle S \rangle \mid b$

$\langle B \rangle ::= b \langle Y \rangle$

$\langle Y \rangle ::= \langle S \rangle \mid \varepsilon$

Η τελική διορθωμένη LL/1 γραμματική είναι:

$\langle S \rangle ::= a \langle A \rangle \mid b \langle B \rangle$

$\langle A \rangle ::= a \langle X \rangle$

$\langle X \rangle ::= \langle S \rangle \mid \varepsilon$

$\langle B \rangle ::= b \langle Y \rangle$

$\langle Y \rangle ::= \langle S \rangle \mid \varepsilon$

Υπενθύμιση

Κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο

$A \rightarrow \alpha \beta_1 | \alpha \beta_2 | \dots | \alpha \beta_n$

Η **Αριστερή Παραγοντοποίηση** γίνεται ως εξής:

$A \rightarrow \alpha B$

$B \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$

3.5.26 Φροντιστήριο 2023 – Άσκηση 1

Δίνεται η παρακάτω BNF γραμματική με τερματικά σύμβολα τα 0 1

$\langle S \rangle ::= 0 \langle S \rangle \mid 1 \langle A \rangle$

$\langle A \rangle ::= 0 \langle S \rangle \mid 1 \langle B \rangle$

$\langle B \rangle ::= 0 \langle B \rangle \mid 1 \langle B \rangle \mid \epsilon$

1. Τι τύπου είναι η γραμματική στην ιεραρχία Chomsky; Αιτιολογήστε την απάντησή σας.
2. Τι είδους συμβολοσειρές περιγράφει η γραμματική; Δώστε παραδείγματα αποδεκτών συμβολοσειρών καθώς και τη μικρότερη σε μήκος συμβολοσειρά.
3. Μπορείτε να φτιάξετε μια κανονική έκφραση αντίστοιχη της γραμματικής και γιατί; Αν ναι, παρουσιάστε την, καθώς και το αντίστοιχο πεπερασμένο ντετερμινιστικό αυτόματο.
4. Ελέγξτε αν η συμβολοσειρά **011010** είναι μέλος της γλώσσας, παρουσιάζοντας τη **Στοίβα Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης**.

Απάντηση

1. Η ιεραρχία Chomsky είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς Συμφραζόμενα

Γραμματικές Με Συμφραζόμενα

Γραμματικές χωρίς περιορισμούς

Η Γραμματική αυτή στην ιεραρχία Chomsky είναι κανονική διότι σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο. Επιτρέπεται και παραγωγή στο ε

2. Παραδείγματα με Συμβολοσειρές που περιγράφει η γραμματική

$S \Rightarrow 1A \Rightarrow 11B \Rightarrow 11$

$S \Rightarrow 1A \Rightarrow 11B \Rightarrow 110B \Rightarrow 110$

$S \Rightarrow 1A \Rightarrow 11B \Rightarrow 111B \Rightarrow 111$

$S \Rightarrow 1A \Rightarrow 11B \Rightarrow 110B \Rightarrow 1100B \Rightarrow 11001B \Rightarrow 11001$

$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 011$

$S \Rightarrow 0S \Rightarrow 00S \Rightarrow 001A \Rightarrow 0010S \Rightarrow 00101A \Rightarrow 001011B \Rightarrow 001011$

$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011A \Rightarrow 0110S \Rightarrow 01101A \Rightarrow 011011B \Rightarrow 011011$

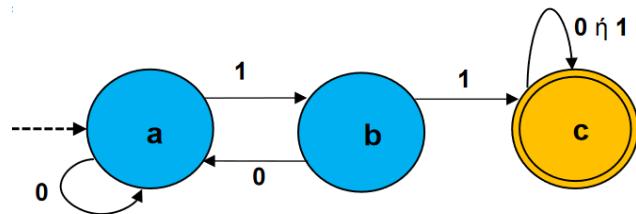
Η γραμματική περιγράφει λέξεις (συμβολοσειρές) με 0, 1 στις οποίες περιέχεται το 11

Ελάχιστη συμβολοσειρά: **11**

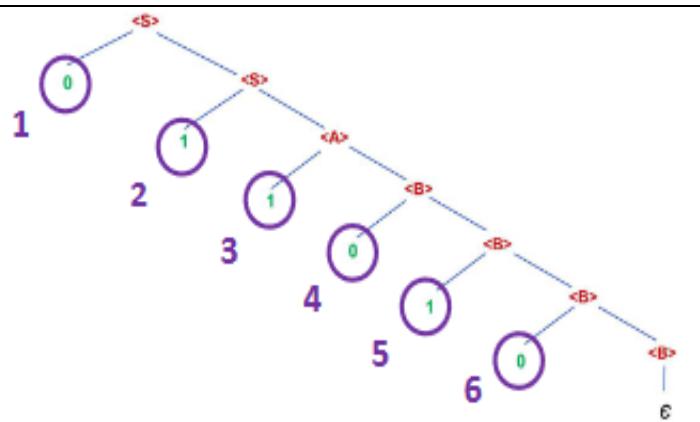
3. Εφόσον είναι κανονική γραμματική, υπάρχει αντίστοιχη κανονική έκφραση που περιγράφει τη γλώσσα της και είναι η ακόλουθη:

$$(0|1)^* \ 11 \ (0|1)^*$$

Το ντετερμινιστικό πεπερασμένο αυτόματο (DFA) που περιγράφει τη γλώσσα είναι το ακόλουθο:



d) Το Δέντρο Συντακτικής Ανάλυσης για τη **011010**



011010

$\langle S \rangle ::= 0\langle S \rangle \mid 1\langle A \rangle$
 $\langle A \rangle ::= 0\langle S \rangle \mid 1\langle B \rangle$
 $\langle B \rangle ::= 0\langle B \rangle \mid 1\langle B \rangle \mid \epsilon$

Η Στοίβα Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης γίνεται με βάση το Συντακτικό Δέντρο

Βήμα	Στοίβα	Είσοδος	Πράξη
0	$\langle S \rangle$	011010 EOF	Πρόβλεψη $\langle S \rangle ::= 0\langle S \rangle$
1	$\langle S \rangle 0$	011010 EOF	Ταίριασμα συμβόλου 0
2	$\langle S \rangle$	11010 EOF	Πρόβλεψη $\langle S \rangle ::= 1\langle A \rangle$
3	$\langle A \rangle 1$	11010 EOF	Ταίριασμα συμβόλου 1
4	$\langle A \rangle$	1010 EOF	Πρόβλεψη $\langle A \rangle ::= 1\langle B \rangle$
5	$\langle B \rangle 1$	1010 EOF	Ταίριασμα συμβόλου 1
6	$\langle B \rangle$	010 EOF	Πρόβλεψη $\langle B \rangle ::= 0\langle B \rangle$
7	$\langle B \rangle 0$	010 EOF	Ταίριασμα συμβόλου 0
8	$\langle B \rangle$	10 EOF	Πρόβλεψη $\langle B \rangle ::= 1\langle B \rangle$
9	$\langle B \rangle 1$	10 EOF	Ταίριασμα συμβόλου 1
10	$\langle B \rangle$	0 EOF	Πρόβλεψη $\langle B \rangle ::= 0\langle B \rangle$
11	$\langle B \rangle 0$	0 EOF	Ταίριασμα συμβόλου 0
12	$\langle B \rangle$	EOF	Πρόβλεψη $\langle B \rangle ::= \epsilon$
13	ϵ	EOF	ΑΝΑΓΝΩΡΙΣΗ

3.5.27 Φροντιστήριο 2023 – Άσκηση 2

Δίνεται η παρακάτω BNF γραμματική, με τερματικά σύμβολα τα x y

$\langle S \rangle ::= \langle A \rangle \langle B \rangle$

$\langle A \rangle ::= \textcolor{red}{x} \langle A \rangle \textcolor{red}{y} \mid \epsilon$

$\langle B \rangle ::= \textcolor{red}{y} \langle B \rangle \mid \textcolor{red}{y}$

1. Τι τύπου είναι η γραμματική στην ιεραρχία Chomsky; Είναι LL(1) η γραμματική; Αιτιολογήστε τις απαντήσεις σας.
2. Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Δώστε παραδείγματα αποδεκτών συμβολοσειρών καθώς και τη μικρότερη σε μήκος.
3. Ελέγξτε αν η συμβολοσειρά **xxxxyy** είναι μέλος της γλώσσας, παρουσιάζοντας τη **Στοίβα Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης**.

Απαντήστε στην ερώτηση παρουσιάζοντας τη Στοίβα Ολίσθησης–Ελάττωσης Bottom-Up συντακτικής ανάλυσης της συμβολοσειράς.

Απάντηση

1. Η ιεραρχία Chomsky είναι η ακόλουθη:

Γραμματικές

Κανονικές Γραμματικές

Γραμματικές Χωρίς
Συμφραζόμενα

Γραμματικές Με
Συμφραζόμενα

Γραμματικές χωρίς
περιορισμούς

Στην ιεραρχία Chomsky **η Γραμματική είναι Χωρίς Συμφραζόμενα**:

- ✓ Αφού στην 1^η παραγωγή, το δεξί μέλος αρχίζει με μη-τερματικό σύμβολο η **γραμματική δεν είναι κανονική**
- ✓ Τα αριστερά μέλη όλων των παραγωγών έχουν μόνο ένα μη-τερματικό σύμβολο **οπότε δεν είναι γραμματική με συμφραζόμενα**

Η γραμματική δεν είναι LL(1) διότι υπάρχουν δύο παραγωγές για το ** που το δεξί μέλος τους αρχίζει με το ίδιο σύμβολο (y).**

Παρατήρηση-Διόρθωση σε LL/1

Ο κανόνας $\langle B \rangle ::= \textcolor{red}{y} \langle B \rangle \mid \textcolor{red}{y}$ διορθώνεται σε:

$\langle B \rangle ::= \textcolor{red}{y} \langle K \rangle$

$\langle K \rangle ::= \langle B \rangle \mid \epsilon$

Η τελική LL/1 διορθωμένη γραμματική είναι η ακόλουθη:

$\langle S \rangle ::= \langle A \rangle \langle B \rangle$

$\langle A \rangle ::= \textcolor{red}{x} \langle A \rangle \textcolor{red}{y} \mid \epsilon$

$\langle B \rangle ::= \textcolor{red}{y} \langle K \rangle$

$\langle K \rangle ::= \langle B \rangle \mid \epsilon$

b) Η γραμματική περιγράφει συμβολοσειρές από x και y όπου:

- ο αριθμός των y είναι μεγαλύτερος από τον αριθμό των x
- και τα x (αν υπάρχουν) προηγούνται των y.

Παραδείγματα Συμβολοσειρών : xxxyyyyy, xyy, yyy

Ελάχιστη συμβολοσειρά: y

Λογική της Top-Down Συντακτικής Ανάλυσης ή Λογική της Στοίβας Πρόβλεψης-Ταιριάσματος

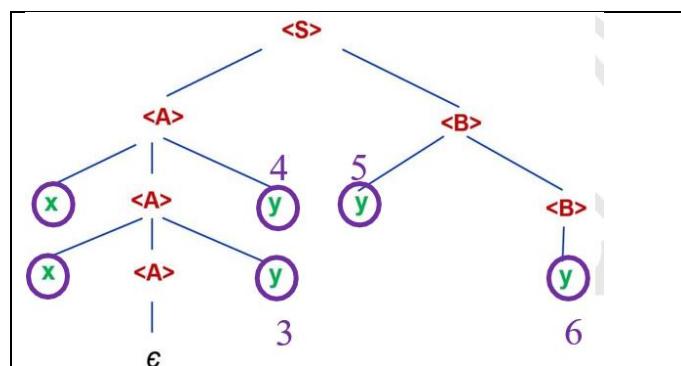
Χρησιμοποιείται μια Στοίβα και δύο Πράξεις:

Ταιριασμα 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.

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.

Αν καμία από τις δύο πράξεις δεν μπορεί να εφαρμοστεί, τότε υπάρχει συντακτικό σφάλμα.

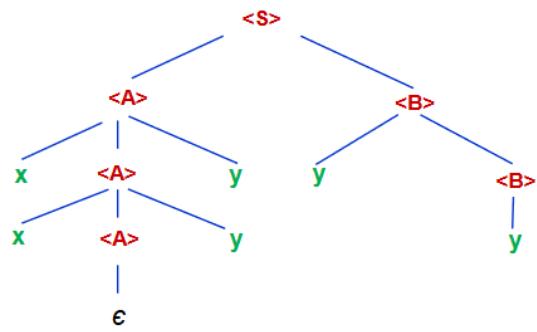
c) Το Δέντρο Συντακτικής Ανάλυσης για τη συμβολοσειρά **xxyyyy** είναι το ακόλουθο:



Βήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	<u>xx</u> yyyy EOF	Πρόβλεψη <S> ::= <A>
1	<A>	xx <u>y</u> yyy EOF	Πρόβλεψη <A> ::= x<A>y
2	 <u>y</u> <u><A>x</u>	xx <u>y</u> yyy EOF	Ταιριασμα συμβόλου x 1
3	y<A>	xyyy EOF	Πρόβλεψη <A> ::= x<A>y
4	yy <u><A>x</u>	xyyy EOF	Ταιριασμα συμβόλου x 2
5	yy<A>	yyy EOF	Πρόβλεψη <A> ::= ε
6	yy	yyy EOF	Ταιριασμα συμβόλου y 3
7	y	yy EOF	Ταιριασμα συμβόλου y 4
8		yy EOF	Πρόβλεψη ::= y

9	<u>y</u>	yy EOF	Ταίριασμα συμβόλου y 
10		y EOF	Πρόβλεψη ::= y
11	y	y EOF	Ταίριασμα συμβόλου y 
12	ε	EOF	ΑΝΑΓΝΩΡΙΣΗ

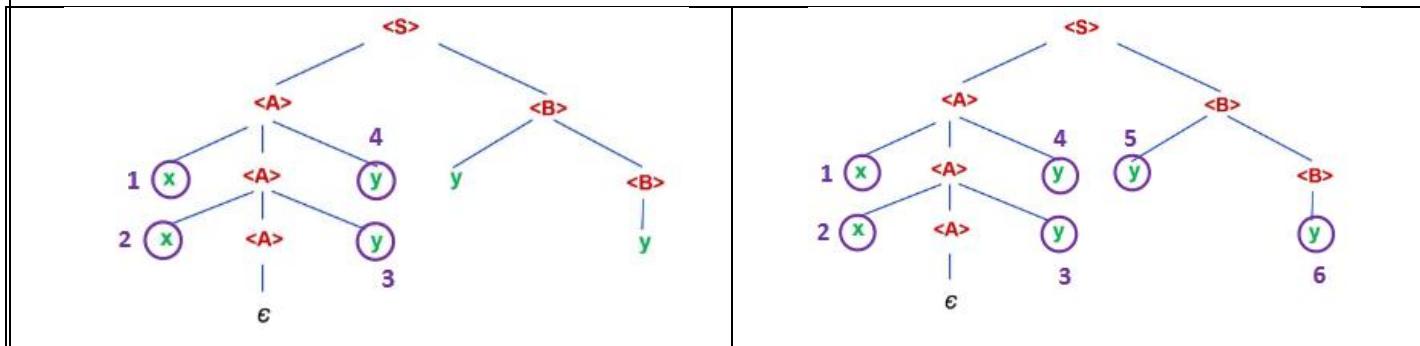
d) Το Δέντρο Συντακτικής Ανάλυσης



Συντακτική Ανάλυση Bottom-Up - Συντακτικοί Αναλυτές ολίσθησης-ελάττωσης (shift-reduce parsers).

Χρησιμοποιούν μια Στοίβα και δύο Πράξεις:

- ✓ **Ολίσθηση (shift):** Αφαιρεί ένα σύμβολο από την αρχή του string και το βάζει στην κορυφή της στοίβας.
- ✓ **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.



Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	xxyyyy EOF	Ολίσθηση 1
1	x	xyyy EOF	Ολίσθηση 2
2	xx	yyy EOF	Ελάττωση $\langle A \rangle ::= \epsilon$
3	xx<A>	yyy EOF	Ολίσθηση 3
4	<u>xx</u> <A> <u>y</u>	yy EOF	Ελάττωση $\langle A \rangle ::= x \langle A \rangle y$
5	x<A>	yy EOF	Ολίσθηση 4
6	x<A>y	yy EOF	Ελάττωση $\langle A \rangle ::= x \langle A \rangle y$
7	<A>	yy EOF	Ολίσθηση 5
8	<A>y	y EOF	Ολίσθηση 6
9	<A>yy	EOF	Ελάττωση $\langle B \rangle ::= y$
10	<A>y	EOF	Ελάττωση $\langle B \rangle ::= y \langle B \rangle$
11	<A>	EOF	Ελάττωση $\langle S \rangle ::= \langle A \rangle \langle B \rangle$
12	<S>	EOF	ΑΝΑΓΝΩΡΙΣΗ

3.5.28 Φροντιστήριο 2023 – Άσκηση 3

Δίνεται η παρακάτω BNF γραμματική με τερματικά σύμβολα τα: (), w x y z

```
<A> ::= (<B>)
<B> ::= <C> | <B>, <C>
<C> ::= <A> | <D>
<D> ::= w | x | y | z
```

a) Τι είδους συμβολοσειρές περιγράφει η παραπάνω γραμματική; Περιγράψτε **μερικά βασικά χαρακτηριστικά** των συμβολοσειρών αυτών παρουσιάζοντας και παραδείγματα.

b) Ελέγξτε αν η συμβολοσειρά **((w, x), y)** είναι μέλος της γλώσσας που υποστηρίζει η γραμματική, παρουσιάζοντας τη **Στοίβα Ολίσθησης – Ελάττωσης Bottom-Up συντακτικής ανάλυσης**

c) Είναι η γραμματική **LL(1)**; Αιτιολογήστε την απάντησή σας. Αν η γραμματική δεν είναι LL(1), κάνετε τους αναγκαίους μετασχηματισμούς και παρουσιάστε την ισοδύναμη LL(1) γραμματική.

Απάντηση

a)

Παραδείγματα Συμβολοσειρών:

- A⇒(B) ⇒ (C) ⇒(D) ⇒ **(x)** Εναλλακτικά **(w), (y), (z)**
- A⇒(B) ⇒(B, C) ⇒(C, C) ⇒ (D, D) ⇒ **(w, x)**
- A⇒(B) ⇒(B, C) ⇒(B, C, C) ⇒ (C, C, C) ⇒(D, D, D) ⇒ **(w, x, y)**
- A⇒(B) ⇒(C) ⇒(A) ⇒((B)) ⇒ (C) ⇒((D)) ⇒ **((z))**
- A⇒(B) ⇒(C) ⇒(A) ⇒((B)) ⇒ ((B, C)) ⇒((B, A)) ⇒ ((B, (B)) ⇒ ((C, (C, C)) ⇒((D, (D, D)) ⇒ **((x, y, z))**
- **((w,(w, w)), (y, y))**

Χαρακτηριστικά Συμβολοσειρών:

- ✓ Είναι **strings με ιεραρχικά τοποθετημένα ζευγάρια παρενθέσεων** (). Ουσιαστικά είναι ένα ζευγάρι παρενθέσεων που μπορεί να περιέχει άλλα ζευγάρια, τα οποία με τη σειρά τους μπορεί να περιέχουν άλλα κ.ο.κ.
- ✓ Το περιεχόμενο των ζευγαριών είναι άλλα ζευγάρια ή/και τα γράμματα w, x, y, z χωρισμένα με , (αν είναι περισσότερα από ένα γράμματα/ζευγάρια)
- ✓ Το ελάχιστο μήκος των συμβολοσειρών είναι 3

Συντακτική Ανάλυση Bottom-Up - Συντακτικοί Αναλυτές ολίσθησης-ελάττωσης (shift-reduce parsers).

Χρησιμοποιούν μια Στοίβα και δύο Πράξεις:

- ✓ **Ολίσθηση (shift):** Αφαιρεί ένα σύμβολο από την αρχή του string και το βάζει στην κορυφή της στοίβας
- ✓ **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.

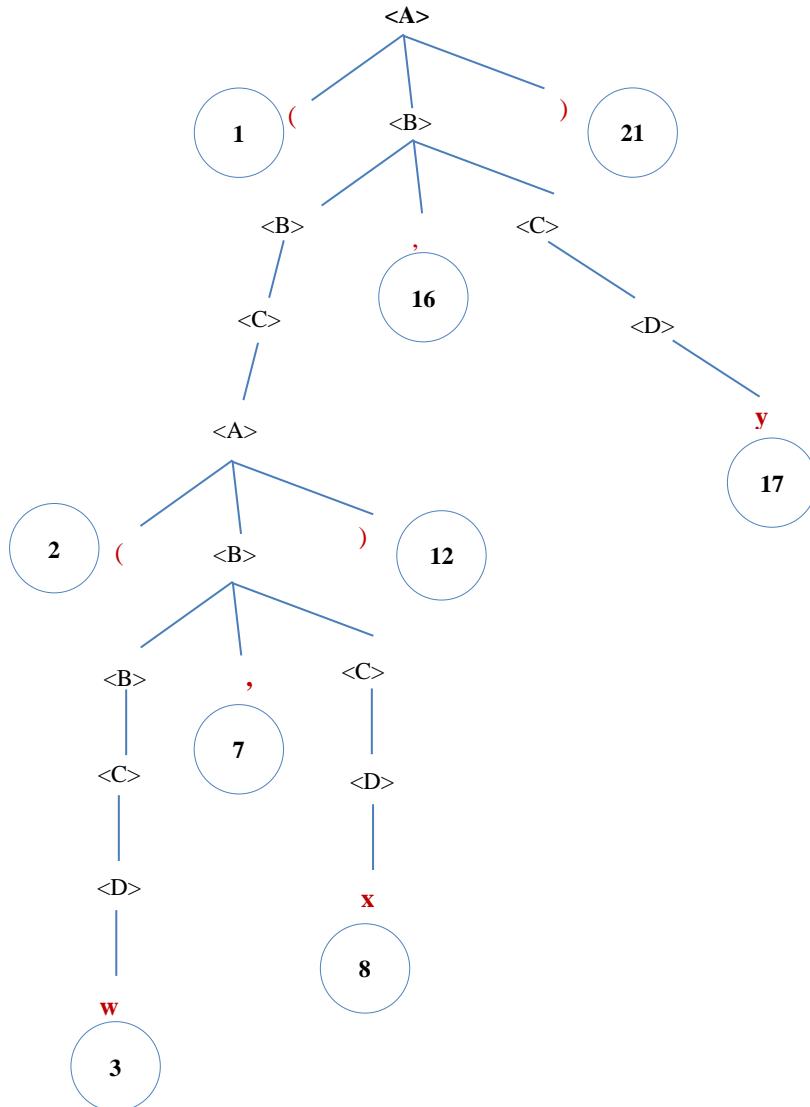
$<A> ::= ()$

$::= <C> \mid , <C>$

$<C> ::= <A> \mid <D>$

$<D> ::= w \mid x \mid y \mid z$

Το συντακτικό δέντρο (parse tree) για τη συμβολοσειρά $((w, x), y)$ είναι το ακόλουθο:



Παρατίθηση: Όταν κατασκευάζεται parse tree για μια συμβολοσειρά αυτό σημαίνει ότι η συμβολοσειρά αυτή είναι έγκυρη δηλ. ανήκει στη γλώσσα. Συνεπώς θα πρέπει η **Στοίβα Ολίσθησης – Ελάττωσης Bottom-Up συντακτικής ανάλυσης να την αναγνωρίσει.**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	((w, x), y) EOF	Ολίσθηση
1	((w, x), y) EOF	Ολίσθηση
2	((w, x), y) EOF	Ολίσθηση
3	((w	, x), y) EOF	Ελάττωση $\langle D \rangle ::= w$
4	((<D>	, x), y) EOF	Ελάττωση $\langle C \rangle ::= \langle D \rangle$
5	((<C>	, x), y) EOF	Ελάττωση $\langle B \rangle ::= \langle C \rangle$
6	((, x), y) EOF	Ολίσθηση
7	((,	x), y) EOF	Ολίσθηση
8	((, x), y) EOF	Ελάττωση $\langle D \rangle ::= x$
9	((, <D>), y) EOF	Ελάττωση $\langle C \rangle ::= \langle D \rangle$
10	((, <C>), y) EOF	Ελάττωση $\langle B \rangle ::= \langle B \rangle, \langle C \rangle$
11	((), y) EOF	Ολίσθηση
12	(()	, y) EOF	Ελάττωση $\langle A \rangle ::= (\langle B \rangle)$
13	(<A>	, y) EOF	Ελάττωση $\langle C \rangle ::= \langle A \rangle$
14	(<C>	, y) EOF	Ελάττωση $\langle B \rangle ::= \langle C \rangle$
15	(, y) EOF	Ολίσθηση
16	(,	y) EOF	Ολίσθηση
17	(, y) EOF	Ελάττωση $\langle D \rangle ::= y$
18	(, <D>) EOF	Ελάττωση $\langle C \rangle ::= \langle D \rangle$
19	(, <C>) EOF	Ελάττωση $\langle B \rangle ::= \langle B \rangle, \langle C \rangle$
20	() EOF	Ολίσθηση
21	()	EOF	Ελάττωση $\langle A \rangle ::= (\langle B \rangle)$
22	<A>	EOF	Αναγνώριση

- Στη στοίβα Bottom-UP ξεκινάμε βάζοντας στη στοίβα τον κενό χαρακτήρα ε και έχουμε μια συμβολοσειρά εισόδου που πρέπει να αναγνωρίσουμε
- Αν η συμβολοσειρά εισόδου είναι ΕΓΚΥΡΗ (δηλ. ανήκει στη γλώσσα) η στοίβα Bottom-UP καταλήγει έχοντας στη στοίβα MONO το αρχικό σύμβολο της γραμματικής (εδώ το <A>) και έχει διαβάσει ΟΛΗ τη συμβολοσειρά εισόδου (EOF) οπότε καταλήγει σε ΑΝΑΓΝΩΡΙΣΗ
- Αν η συμβολοσειρά εισόδου ΔΕΝ είναι ΕΓΚΥΡΗ (δηλ. ΔΕΝ ανήκει στη γλώσσα) η στοίβα Bottom-UP ΔΕΝ καταλήγει σε ΑΝΑΓΝΩΡΙΣΗ

c)

Η γραμματική δεν είναι LL(1) διότι υπάρχει άμεση αριστερή αναδρομή στο 2^o κανόνα: $\underline{} ::= <C> | \underline{}$, $<C>$

Θα εφαρμόσουμε Απαλοιφή Αριστερής Αναδρομής στον παραπάνω κανόνα:

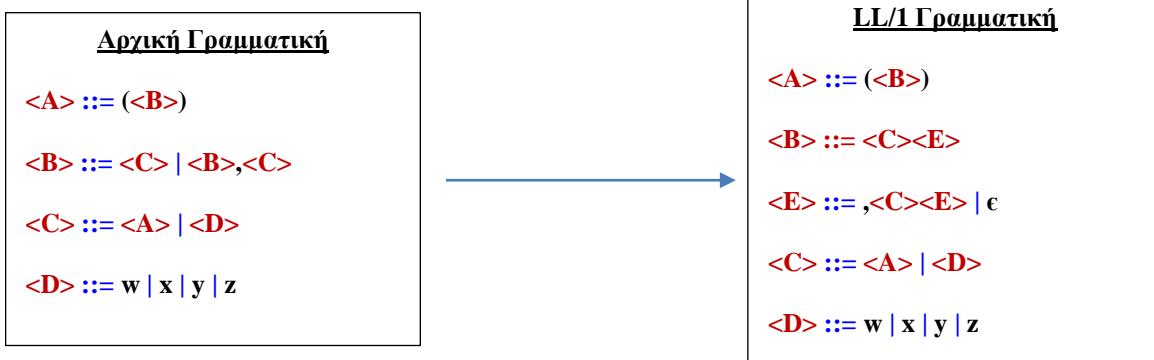
$$\boxed{A \rightarrow A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m \quad \leftrightarrow \quad A \rightarrow \beta_1 B | \dots | \beta_m B \\ B \rightarrow \alpha_1 B | \dots | \alpha_n B | \epsilon}$$

$$\underline{} ::= <C> | \underline{} , <C>$$

$$::= <C> <E>$$

$$<E> ::= , <C> <E> | \epsilon$$

Οπότε έχουμε πλέον την παρακάτω ισοδύναμη LL(1) γραμματική:



3.5.29 Θέμα 1 Ιούνιος 2023

Δίνεται η παρακάτω BNF γραμματική τμήμα της περιγραφής του συντακτικού μιας γλώσσας προγραμματισμού.

Τα τερματικά σύμβολα της γραμματικής είναι τα: **int**, **float**, **id**, **ένας χαρακτήρας**, **ένας χαρακτήρας κενού** και **ένα χαρακτήρας ;**

<S> ::= int <κενό> id <A> | float <κενό> id <A>

<A> ::= , <κενό> id <A> | ;

<κενό> ::= ένας χαρακτήρας κενού (μπορείτε να τον συμβολίζετε με _)

(a) Να περιγράψετε **τις κατασκευές που παράγονται από τη γραμματική**; Τι θα μπορούσε να περιγράψει στη γλώσσα C η παραπάνω γραμματική, ως τμήμα της συνολικής γραμματικής της γλώσσας; Περιγράψτε μερικά παραδείγματα νόμιμων εκφράσεων της γλώσσας

(b) Είναι LL(1) η γραμματική; Αιτιολογήστε την απάντηση σας

(c) Είναι το **int mi, pi**; μέλος της γλώσσας; Απαντήστε παρουσιάζοντας **τη Στοίβα-Ολίσθησης-Ellάttwosης Bottom-Up συντακτικής ανάλυσης** για το string

(d) Τι **τύπου είναι η γραμματική στην ιεραρχία Chomsky**; Αιτιολογήστε την απάντηση σας

(e) Μπορείτε να φτιάξετε μια **κανονική έκφραση** αντίστοιχη της γραμματικής και γιατί; Αν ναι παρουσιάστε τη

(f) Επιπλέον υπάρχει **ντετερμινιστικό πεπερασμένο αυτόματο αντίστοιχο της κανονικής έκφρασης**; Αν ναι παρουσιάστε το διάγραμμα καταστάσεων-μεταβάσεων του αυτόματου;

Απάντηση

(a)

Εγδεικτικές Παραγωγές

$S \Rightarrow \text{int} <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{int_id};$

$S \Rightarrow \text{float} <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{float_id};$

$S \Rightarrow \text{int} <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{int} <\text{κενό}> \text{id} , <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{int_id,_id};$

$S \Rightarrow \text{float} <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{float} <\text{κενό}> \text{id}, <\text{κενό}> \text{id} <\text{A}> \Rightarrow \text{float_id,_id};$

Χαρακτηριστικά Συμβολοσειρών

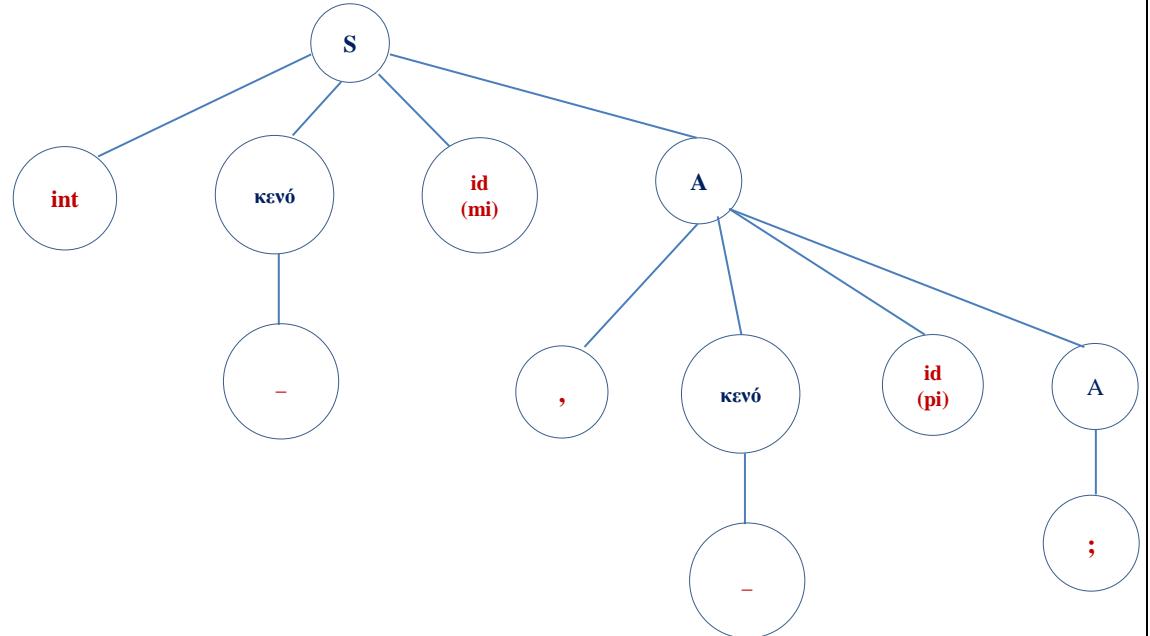
- Η γλώσσα περιλαμβάνει όλες τις συμβολοσειρές **που αποτελούν δήλωση μεταβλητών τύπου int ή float**
- Κάθε δήλωση περιλαμβάνει μια ή περισσότερες μεταβλητές (id's) **τύπου int ή float** που διαχωρίζονται με κόμμα και κενό και η δήλωση τελειώνει με το **χαρακτήρα ;**
- Η γραμματική αυτή θα μπορούσε να περιγράψει **το τμήμα δηλώσεων μεταβλητών στη γλώσσα C**
- Βέβαια η γραμματική αυτή περιλαμβάνει δήλωση μεταβλητών μόνο τύπου int ή float και όχι άλλων τύπων όπως στη C

(b)

Η γραμματική είναι LL/1 διότι:

- ΔΕΝ παρουσιάζει αριστερή αναδρομή
- ΔΕΝ έχει εναλλακτικούς κανόνες που αρχίζουν με το ίδιο τερματικό σύμβολο

(c) Παρακάτω δίνονται τα βήματα του parse tree για τη συμβολοσειρά **int m; p;**



Παρακάτω δίνονται τα βήματα της **Στοίβας-Ολίσθησης-Ελάττωσης Bottom-Up συντακτικής ανάλυσης** για την αναγνώριση της συμβολοσειράς **int <κενό> mi, <κενό> pi;**

Βήμα	Στοίβα	Είσοδος	Πράξη
0	ϵ	int _ mi, _ pi; EOF	Ολίσθηση
1	int	_ mi, _ pi; EOF	Ολίσθηση
2	int _	mi, _ pi; EOF	Ελάττωση $<\text{κενό}> ::= _$
3	int <κενό>	mi, _ pi; EOF	Ολίσθηση
4	int <κενό> mi	_ pi; EOF	Ολίσθηση
5	int <κενό> mi,	pi; EOF	Ολίσθηση
6	int <κενό> mi, _	pi; EOF	Ελάττωση $<\text{κενό}> ::= _$
7	int <κενό> mi, <κενό>	pi; EOF	Ολίσθηση
8	int <κενό> mi, <κενό> pi	; EOF	Ολίσθηση
9	int <κενό> mi, <κενό> pi;	EOF	Ελάττωση $<\text{A}> ::= ;$
10	int <κενό> mi , <κενό> pi <A>	EOF	Ελάττωση $<\text{A}> ::= , <\text{κενό}> \text{id} <\text{A}>$
11	int <κενό> <id><A>	EOF	Ελάττωση $<\text{S}> ::= \text{int} <\text{κενό}> \text{id} <\text{A}>$
12	<S>	EOF	Αναγνώριση

Παρατήρηση

Παρακάτω δίνονται τα βήματα της Στοίβας Ταιριάσματος – Πρόβλεψης Top-Down συντακτικής ανάλυσης για την αναγνώριση της συμβολοσειράς **int <κενό> mi, <κενό> pi;**

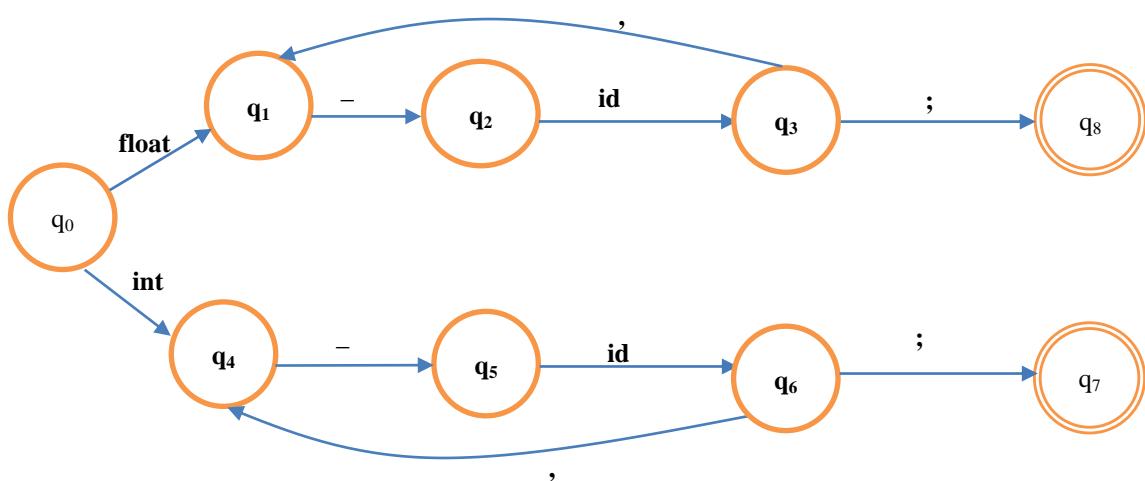
ήμα	Στοίβα	Είσοδος	Πράξη
0	<S>	int _ mi, _ pi; EOF	Πρόβλεψη <S> ::= int <κενό> id <A>
1	<A> id <κενό> int	int _ mi, _ pi; EOF	Ταίριασμα συμβόλου int
2	<A> id <κενό>	_ mi, _ pi; EOF	Πρόβλεψη <κενό> ::= _
3	<A> id _	_ mi, _ pi; EOF	Ταίριασμα συμβόλου _
4	<A> id (mi)	mi , _ pi; EOF	Ταίριασμα συμβόλου id
5	<A>	, _ pi; EOF	Πρόβλεψη <A> ::=, <κενό> id <A>
6	<A> id <κενό>,	, _ pi; EOF	Ταίριασμα συμβόλου ,
7	<A> id <κενό>	_ pi; EOF	Πρόβλεψη <κενό> ::= _
8	<A> id _	_ pi; EOF	Ταίριασμα συμβόλου _
9	<A> id (pi)	pi; EOF	Ταίριασμα συμβόλου pi
10	<A>	; EOF	Πρόβλεψη <A> ::= ;
11	;	; EOF	Ταίριασμα συμβόλου ;
12	ε	EOF	Αναγνώριση

(d) Η Γραμματική αυτή στην ιεραρχία Chomsky είναι κανονική διότι σε όλες τις παραγωγές, τα δεξιά μέλη αρχίζουν με τερματικό σύμβολο.

(e) Η κανονική έκφραση που περιγράφει την ίδια γλώσσα είναι η ακόλουθη:

$$(\text{int} \mid \text{float}) \text{ } \underline{\text{id}} \text{ } (\text{, } \underline{\text{id}})^*$$

(f) Το ντετερμινιστικό πεπερασμένο αυτόματο αντίστοιχο της κανονικής έκφρασης είναι:



4 ΘΕΜΑΤΑ ΘΕΩΡΙΑΣ ΚΑΙ ΚΩΔΙΚΑ

4.1 Θέμα 4 Σεπτέμβριος 2013

1. Τι ονομάζουμε μεταβλητές; Τι ιδιότητες έχουν; OXI

Απάντηση

Αφαιρετική αναπαράσταση διεύθυνσης μνήμης, ή συλλογής διευθύνσεων μνήμης του Η/Υ. Αποτελείται από 4 στοιχεία (Όνομα, Διεύθυνση, Τιμή και Ιδιότητες). Το Όνομα της είναι συνδυασμός από αλφαριθμητικούς χαρακτήρες. Οι ιδιότητες περιλαμβάνουν τον Τύπο Τιμών, Χρόνο Ζωής και Εμβέλεια.

2. Τι ονομάζουμε σύστημα τύπων; -OXI

Απάντηση

Η δυνατότητα ορισμού νέων Τύπων Δεδομένων και δήλωσης μεταβλητών, οι τιμές των οποίων περιορίζονται στα στοιχεία ενός ΤΔ, με πραγματοποίηση ελέγχου τύπου. Ένα Type System αποτελείται από:

1. Ένα μηχανισμό ορισμού ΤΔ και συσχέτισης τους με συγκεκριμένες κατασκευές γλώσσας, δηλαδή με αυτές που έχουν τιμή, όπως μεταβλητές, παράμετροι, εκφράσεις, ...
2. Ένα σύνολο κανόνων για Ισοδυναμία ΤΔ (Type Equivalence), Συμβατότητα ΤΔ (Type Compatibility) και Εξαγωγή (Type Inference)

4.2 Θέμα 1 Σεπτέμβριος 2012, 2015 και Σεπτέμβριος 2018

(a) Ποιο είναι το αποτέλεσμα του παρακάτω προγράμματος C; Εξηγείστε την απάντηση σας

*int *p;*

**p=15;*

(b) Δίνονται οι παρακάτω δηλώσεις μεταβλητών σε ένα πρόγραμμα της C:

short int s;

unsigned long int l;

char c;

float f;

double d;

Ποιες μετατροπές τύπων και με ποιο τρόπο θα γίνουν με την εκτέλεση κάθε μιας ξεχωριστά από τις παρακάτω εντολές; Μπορούν και ποιες προϋποθέσεις να υπάρξουν περιπτώσεις π.χ. απώλεια ακρίβειας σε κάποια/κάποιες περιπτώσεις;

(i) *s=l;*

ii) *l=s;*

(iii) *s=c;*

(iv) *f=l;*

(v) *d=f;*

(vi) *f=d;*

(c) Δίνεται η εντολή της C: *if A < B // C < D then*. Δείξτε τη σειρά εκτέλεσης υπολογισμών χρησιμοποιώντας παρενθέσεις. Κάντε το ίδιο για την αντίστοιχη εντολή στην Pascal: *if A < B or C < D then*. Για τις δύο παραπάνω περιπτώσεις υπάρχουν κάποιες προϋποθέσεις για τους τύπους των μεταβλητών A, B, C, D ώστε να μην επιστραφεί λάθος κατά τη μετάφραση;

Απάντηση

(a) Δίνονται οι εντολές:

*int *p;*

**p=15;*

Το αποτέλεσμα που προκύπτει είναι ένα **λογικό σφάλμα** διότι δηλώνουμε ένα δείκτη με όνομα p σε ακέραιο και μετά θέτουμε στο περιεχόμενο της μεταβλητής που δείχνει ο δείκτης p την τιμή 15. Επειδή όμως ο δείκτης p δεν δείγνει σε κάποια μεταβλητή δεν μπορεί να καταχωρίσει την τιμή αυτή. Ουσιαστικά θα έχουμε run-time error.

Το σωστό πρόγραμμα θα ήταν το εξής:

*int *p, x;*

p=&x;

**p=15; //x=15*

Τώρα ο δείκτης p δείχνει στη θέση μνήμης x, οπότε η ανάθεση *p=15 έχει νόημα αφού θέτει στο x την τιμή 15.

(b) Δίνονται οι δηλώσεις:

Θεωρία για μέγεθος κάθε μεταβλητής

char	1 byte	-128 to 127 or 0 to 255	
unsigned char	1 byte	0 to 255	
signed char	1 byte	-128 to 127	
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647	
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295	
short	2 bytes	-32,768 to 32,767	
unsigned short	2 bytes	0 to 65,535	
long	4 bytes	-2,147,483,648 to 2,147,483,647	
unsigned long	4 bytes	0 to 4,294,967,295	
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Σύμφωνα με την ακόλουθη θεωρία:

■ Μετατροπή ΤΔ

Διαδικασία στην οποία μια τιμή ενός ΤΔ, μετατρέπεται σε τιμή άλλου ΤΔ.

Κατηγορίες Μετατροπών:

- Διεύρυνση (widening). Π.χ. από **int** σε **float**
- Περιορισμός (narrowing). Π.χ. από **float** σε **int** (truncation)

Δύο Τρόποι:

- Implicit* (στη **C** όλα επιτρέπτα, στην **Pascal** μόνο διεύρυνση **INT** σε **REAL**)
- Explicit* (στην **Pascal** : **i:= trunc(r)**, **i:= round(r)**)

Οι μετατροπές τύπων που γίνονται είναι οι ακόλουθες:

(i) **s=l;** → Εδώ έχουμε **περιορισμό** (narrowing). Ο τύπος **long** της μεταβλητής **l** θα μετατραπεί σε **short**. Εδώ μπορεί να υπάρχει απώλεια ακρίβειας διότι ενδέχεται η τιμή της μεταβλητής **l** που είναι **long int** να μην μπορεί να αναπαρασταθεί ως **short int**.

Αν η τιμή της μεταβλητής **l** είναι στο εύρος από -32768 έως 32.767 τότε η τιμή της θα καταχωρηθεί κανονικά στην **s**, αλλιώς η τιμή αυτή δεν θα αναπαρίσταται σωστά (δηλαδή θα έχουμε απώλεια ακρίβειας)

(ii) **l=s;** → Εδώ έχουμε **διεύρυνση** (widening). Ο τύπος **short int** μετατρέπεται σε **long int**. Η τιμή της **s** θα καταχωρηθεί κανονικά στην **l**

(iii) **s=c;** → Έχουμε **διεύρυνση** (widening). Ο τύπος **char** δεσμεύει 1 byte και ο τύπος **short** 2 bytes . Στη μεταβλητή **s** καταχωρείται ο ASCII κωδικός του χαρακτήρα που βρίσκεται στη μεταβλητή **c**

(iv) **f=l;** → Δεν έχουμε ούτε **περιορισμό** ούτε **διεύρυνση** εφόσον η τιμή της μεταβλητής **l** χωράει στην τιμή της **f** με τα επιπλέον δεκαδικά ψηφία που θα προστεθούν διότι και ο τύπος **float** δεσμεύει 4 bytes και ο τύπος **unsigned long** δεσμεύει 4 bytes. Αν η τιμή της μεταβλητής **l** δεν χωρά στην τιμή της **f** με τα επιπλέον δεκαδικά ψηφία τότε θα έχουμε περιορισμό

(v) **d=f;** → Έχουμε **διεύρυνση** (widening). Ο τύπος **float** της μεταβλητής **f** δεσμεύει 4 bytes θα γενικευτεί σε τύπο **double** ο οποίος δεσμεύει 8 bytes με επιπλέον προσθήκη μηδενικών. Η τιμή της **f** καταχωρείται κανονικά στη **d**

(vi) **f=d;** → Έχουμε **περιορισμό (narrowing)**. Ο τύπος **double** της μεταβλητής **d** θα περιοριστεί σε τύπο **float** της μεταβλητής **f**, άρα εδώ μπορεί να υπάρχει απώλεια ακρίβειας διότι ενδέχεται η τιμή της **d** που είναι **double** να μην μπορεί να αναπαρασταθεί ως **float** αφού ο τύπος **float** δεσμεύει μόνο 4 bytes ενώ ο τύπος **double** δεσμεύει 8 bytes. Αν η τιμή της **d** είναι στο εύρος του **float** δηλ. από 1,2E-38 έως 3.4+38 θα καταχωρηθεί χωρίς καμία απώλεια αλλιώς θα υπάρχει απώλεια

(c)

C	Pascal
!	NOT
Postfix ++, --	* / div mod AND
Prefix ++, --	+ - OR
Unary -	= <> < <= > >=
* / %	
Binary + -	
< <= > >=	
== !=	
&&	

- Στη C οι συγκριτικοί τελεστές `<`, `<=`, `>`, `>=`, `!=` έχουν μεγαλύτερη προτεραιότητα από τους λογικούς τελεστές `&&` και `||` συνεπώς η προτεραιότητα των υπολογισμών είναι η εξής: ***if ((A<B) || (C<D))***
- Στην Pascal συμβαίνει το αντίθετο δηλ. οι λογικοί τελεστές NOT, AND, OR έχουν μεγαλύτερη προτεραιότητα από τους συγκριτικούς τελεστές `<=`, `<=`, `>` και συνεπώς η προτεραιότητα των υπολογισμών είναι η εξής: ***if ((A<(B OR C))<D) then***. Στην Pascal το αποτέλεσμα του λογικού τελεστή OR δηλ B OR C είναι TRUE ή FALSE, άρα θα πρέπει ο τύπος των μεταβλητών A, B, C και D να είναι BOOLEAN για να έχει νόημα η σύγκριση A< (B OR C) που θα είναι της μορφής TRUE<FALSE ή TRUE<TRUE ή FALSE<TRUE κ.λ.π. **Η προϋπόθεση λοιπόν για να μην επιστραφεί λάθος κατά τη μετάφραση είναι οι μεταβλητές A, B, C και D να είναι τύπου BOOLEAN**
- Στη C δεν υπάρχει κάποια προϋπόθεση διότι το αποτέλεσμα μιας σύγκρισης είναι 1 (TRUE) ή 0 (FALSE). Ακόμα και αν ο τύπος των A, B, C, D είναι π.χ. char συγκρίνονται οι ASCII κωδικοί των char που είναι ακέραιες τιμές

4.3 Θέμα 1 Ιούνιος 2012-OXI

- α) Αναφέρετε δύο γλώσσες προγραμματισμού με έμμεση δήλωση μεταβλητών όσον αφορά τον τύπο δεδομένων. Επίσης αναφέρετε χαρακτηριστικά παραδείγματα αυτών των γλωσσών
- β) Ποια η βασική διαφορά του τύπου δεδομένων δείκτη στην Pascal και στη C όσον αφορά τη μνήμη στην οποία δείχνουν οι μεταβλητές τύπου δείκτη. Ποια η λειτουργία των συναρτήσεων `free(x)` και `dispose(x)` στην C και Pascal αντίστοιχα

Απάντηση

- α) Ο τύπος δεδομένων των μεταβλητών καθορίζεται από τον περιεχόμενο που τους καταχωρούμε. Ως παράδειγμα τέτοιων γλωσσών μπορούμε να αναφέρουμε την PHP και τη Fortran

PHP	FORTRAN
<p>\$X=2; → ο τύπος της X γίνεται αυτόματα int βάσει του περιεχομένου που της καταχωρούμε.</p> <p>Δεν υπάρχει δήλωση τύπου δεδομένων στην PHP. Ο τύπος προσδιορίζεται αυτόματα από το περιεχόμενο που καταχωρούμε. Μετά αν υπάρχει η εντολή:</p> <p><code>\$X=5.5;</code> ο τύπος της γίνεται αυτόματα double βάσει του περιεχομένου που της καταχωρούμε.</p> <p>X="nikos"; ο τύπος της γίνεται αυτόματα String βάσει του περιεχομένου που της καταχωρούμε</p>	<p>Αν το όνομα της μεταβλητής αρχίζει από I, J, K, L, M, N η μεταβλητή θεωρείται αυτόματα τύπου INTEGER άλλις θεωρείται REAL</p> <p>Παραδείγματα I=5; και A=5.5;</p>

β)

Γενικά οι δείκτες δείχνουν σε αντικείμενα της heap memory. Στη C μπορεί να δείχνει ο δείκτης και σε αντικείμενα της stack μνήμης. Στην Pascal όχι.

Καταστροφή αντικειμένου από τη heap memory:

- C : free(x)
- C++ : delete x
- Pascal : dispose(x)

4.4 Θέμα 1 Ιούνιος 2013

(a) Περιγράψτε το είδος των δεδομένων που ορίζονται με τις παρακάτω δηλώσεις της γλώσσας C:

i. `float *A[3];`

ii. `float **B[4];`

(b) Ποια **πιθανή παράπλευρη συνέπεια (side effect)** έχει η εφαρμογή της ιδιότητας υπολογισμού περιορισμένης έκτασης (short circuit evaluation) **στον παρακάτω C κώδικα**; Είναι επιθυμητή;

`while (D!=0 && G/D>=2) ...`

(c) Στη **Fortran** με ποια σειρά θα γίνουν οι υπολογισμοί στην παρακάτω αριθμητική έκφραση; Παρουσιάστε την απάντηση σας με χρήση παρενθέσεων.

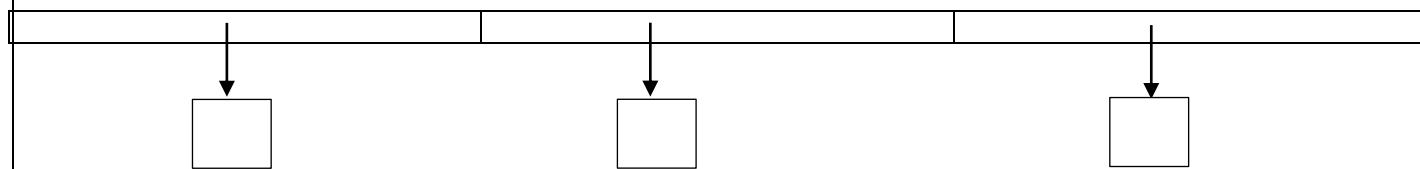
`A**B*C**D**E**F*G**H*I`

Απάντηση

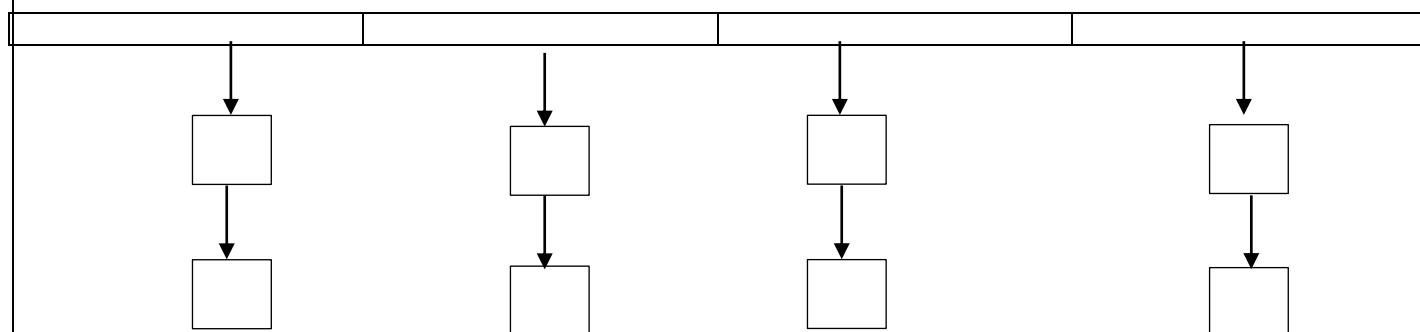
(a)

- i. Δηλώνεται ένας **μονοδιάστατος πίνακας με όνομα A που περιέχει 3 δείκτες** όπου ο καθένας δείχνει σε ένα πραγματικό

A



- ii. Δηλώνεται ένας μονοδιάστατος πίνακας με όνομα B που περιέχει 4 στοιχεία που το καθένα είναι **δείκτης σε δείκτη σε πραγματικό (float)**. Ουσιαστικά δηλώνεται ένας δισδιάστατος πίνακας πραγματικών



(b)

Θεωρία

• **Υπολογισμός Περιορισμένης Έκτασης**

(Short – Circuit Evaluation)

Υπολογισμός μιας έκφρασης, χωρίς να χρειάζεται να υπολογιστούν όλοι οι τελεστέοι.

A. Σε αριθμητικές εκφράσεις:

$(4 * A) * (B / 3 - 1)$ Τι επίπτωση υπάρχει εδώ;

Αν $A = 0$, δεν χρειάζεται να υπολογιστεί το $(B / 3 - 1)$.

– Δύσκολο να εντοπιστεί

– Χρησιμοποιείται σπάνια

B. Σε λογικές εκφράσεις:

(A >= 0) AND (B < 10)

Αν $A < 0$, τότε $(A >= 0) = \text{FALSE}$ και δεν χρειάζεται να υπολογιστεί το $(B < 10)$

– Πιθανές Παράπλευρες Συνέπειες (Side Effects):

(A > B) && (B++/2)

Το B θα αλλάζει τιμή (B++), μόνο όταν $A > B$

– **C, C++, Java** χρησιμοποιούν υπολογισμό περιορισμένης
έκτασης

Ο υπολογισμός G/D θα γίνει μόνο όταν η συνθήκη $D!=0$ είναι αληθής διότι εξετάζεται πρώτα η συνθήκη $D!=0$ και εφόσον είναι αληθής τότε εξετάζεται και η $2^{\text{η}}$ συνθήκη $G/D>=2$. **Η παράπλευρη συνέπεια (side effect) είναι ότι αν η συνθήκη $D!=0$ είναι ψευδής τότε δεν θα γίνει ποτέ η εκτέλεση της πράξης G/D ούτε της συνθήκης $G/D>=2$.** Βέβαια αυτό είναι επιθυμητό διότι αν το $D==0$ τότε δεν θα είχε νόημα ο υπολογισμός της πράξης G/D αφού ο παρονομαστής είναι μηδέν.

(c) Σύμφωνα με τον ακόλουθο πίνακα που δείχνει την προτεραιότητα των τελεστών της γλώσσας Fortran:

FORTRAN 77
**
* /
+ -
.EQ. .NE. .GT. .LT. .LE. .GE.
.NOT.
.AND.
.OR.

Όταν έχουμε τελεστές της ίδιας προτεραιότητας, η σειρά εκτέλεσης των πράξεων είναι από αριστερά προς τα δεξιά). Επίσης ο τελεστής δύναμης ** προηγείται του τελεστή *. Με βάση αυτά η παράσταση $A^{**}B*C^{**}D^{**}E^{**}F^{*}G^{**}H*I$ που δίνεται υπολογίζεται ως εξής:

$$(A^{**}B)^*((C^{**}D)^{*}E)^{*}F^{*}(G^{**}H)^{*}I$$

4.5 Θέμα 3 Ιούνιος 2014

Δίνεται η έκφραση $A \parallel B > C \&& D+E$ όπου $A=0$, $B=1$, $C=2$, $D=3$, $E=4$

Με ποια σειρά εκτελείται η εντολή αυτή; Δείξτε με παρενθέσεις. Ποια είναι η τιμή της έκφρασης;

Απάντηση

C
!
Postfix ++, --
Prefix ++, --
Unary -
* / %
Binary + -
< <= > >=
== !=
&&

Η έκφραση που δίνεται είναι σε γλώσσα C. Από τον ανωτέρω πίνακα προκύπτει ότι η προτεραιότητα είναι η εξής:

1. Πρώτα εκτελείται η πρόσθεση $D+E=3+4=7$ και μετά εκτελείται η συνθήκη $7!=0$ που δίνει τιμή true (1)
2. Μετά εκτελείται η συνθήκη με το συσχετιστικό τελεστή δηλ. $B>C$ δηλ. $B=1>C=2$ που δίνει τιμή false (0)
3. Μετά εκτελείται ο λογικός τελεστής $\&\&$ δηλ. **0 && 1** που δίνει τιμή false (0)
4. Μετά εκτελείται ο λογικός τελεστής \parallel δηλ. **0 || 0** που δίνει ως τελική τιμή της έκφρασης το 0 (false)

Η συνολική έκφραση με παρενθέσεις είναι:

$$(A \parallel ((B > C) \&& (D+E)))$$

Παρατήρηση: Όταν σε μια συνθήκη δεν υπάρχει συσχετιστικός τελεστής τότε η εξορισμού συνθήκη είναι $!=0$.

4.6 Θέμα 5 Ιούνιος 2014

Δίνονται οι εκφράσεις **char** A[]="LearningC", **char** *p=&A[5] και **char** *q=A; Τι θα επιστρέψει

- a) το *p
 - β) (*p+2)
 - γ) (*p)+2

Απάντηση

Το αλφαριθμητικό είναι το ακόλουθο:

0	1	2	3	4	5	6	7	8	9
L	e	a	r	n	i	n	g	C	\0

Στη γλώσσα C ισχύει ότι $A = \&A[0]$ δηλ. το όνομα ενός πίνακα ταυτίζεται με τη διεύθυνση του αριθμού των στοιχείων.

- α) Το *ρ επιστρέφει το περιεχόμενο της θέσης στην οποία δείχνει ο δείκτης ρ δηλ. το περιεχόμενο του στοιχείου A[5] δηλ. το χαρακτήρα i

β) Επειδή ο μοναδιαίος τελεστής * έχει μεγαλύτερη προτεραιότητα από τον τελεστή + παίρνουμε πρώτα το περιεχόμενο της θέσης στην οποία δείγνει ο δείκτης p δηλ. το περιεχόμενο του στοιχείου A[5] δηλ. το χαρακτήρα i και του προσθέτουμε το 2 και προκύπτει ο χαρακτήρας k. Ουσιαστικά στον ASCII κωδικό του χαρακτήρα i προστίθεται το 2. Όποιος και αν είναι αυτός ο ASCII κωδικός επειδή οι κώδικες ASCII είναι διαδοχικοί θα προκύψει ο χαρακτήρας k διότι $i+2=k$. Άρα $(*p+2)=k$

γ) Ότι ακριβώς και στο β) $(^*p)+2=k$

Παρατήρηση

Αν έδινε $*(p+2)$ τότε μεταφέρει πρώτα το δείκτη ρ δύο θέσεις δεξιά δηλ. στο χαρακτήρα g και επιστρέφει g άρα $*(p+2)=g$

4.7 Θέμα 4 Σεπτέμβριος 2014 και Θέμα 3 Ιούνιος 2017

(a) Δίνεται το παρακάτω τμήμα δηλώσεων ενός προγράμματος:

type A=array [1..10] of integer;

B=A;

C: A;

D: A;

E: B;

F: array [1..10] of integer;

Τι αντιπροσωπεύουν τα ονόματα A, B, C, D, E, F;

(b) Αν ισχύει **ισοδυναμία ονομάτων** (name equivalence) ποιες από τις οριζόμενες μεταβλητές θα αναγνωρίζονται ως ίδιου τύπου;

(c) Αν ισχύει **ισοδυναμία δομών** (structural equivalence) ποιες από τις οριζόμενες μεταβλητές θα αναγνωρίζονται ως ίδιου τύπου;

Απάντηση

- Με τη δήλωση type στην Pascal δηλώνουμε ένα νέο τύπο δεδομένων. Εδώ δηλώνουμε τον τύπο A ως πίνακα 10 ακεραίων
- Ομοίως και το όνομα B είναι τύπος δεδομένων που αντιπροσωπεύει ένα πίνακα 10 ακεραίων
- Το σύμβολο = στο type δηλώνει τύπο δεδομένων. Το σύμβολο: δηλώνει μεταβλητή
- Το C είναι μεταβλητή τύπου A
- Το D είναι μεταβλητή τύπου A
- Το E είναι μεταβλητή τύπου B.
- Το F είναι μεταβλητή τύπου πίνακας 10 ακεραίων

Προσοχή: Ότι έχει = είναι τύπος δεδομένων και ότι έχει : είναι μεταβλητή

Θεωρία για Ισοδυναμία Ονομάτων και Δομών

- Στην Ισοδυναμία Ονομάτων δύο Τύποι Δεδομένων είναι Ισοδύναμοι αν έχουν το ίδιο όνομα.
- Στην Ισοδυναμία Δομών δύο Τύποι Δεδομένων είναι Ισοδύναμοι αν έχουν το ίδια συστατικά και την ίδια δομή
- **Αν ισχύει η ισοδυναμία ονομάτων οι Τύποι Δεδομένων A, B και array [1..10] of integer ΔΕΝ είναι ισοδύναμοι αφού ΔΕΝ έχουν το ίδιο όνομα.** Άρα οι μεταβλητές C, D αναγνωρίζονται ως τύπου A, η μεταβλητή E αναγνωρίζεται τύπου B και η μεταβλητή F τύπου πίνακα 10 ακεραίων. Συνεπώς οι μεταβλητές C, D είναι διαφορετικού τύπου από τη B και την F
- **Αν ισχύει η ισοδυναμία δομών οι Τύποι Δεδομένων A, B και array [1..10] of integer είναι ισοδύναμοι γιατί έχουν τα ίδια συστατικά και την ίδια δομή (δηλ. ίδιο περιεχόμενο).** Άρα οι μεταβλητές C, D, E, F είναι όλες του ίδιου τύπου

```
type V1: array[1..10] of real;
  V2: array[1..10] of real;
var X, Z: V1;
    Y: V2;
procedure Sub(A: V1);
begin
  BEGIN
    X:= Y;
    Sub(Y);
  END.
```

Ερωτήματα:

- **X, Y, Z, A** έχουν ίδιο ΤΔ;
- Επιτρέπεται το **X:= Y**;
- Επιτρέπεται το **Sub(Y)**;

Ισοδυναμία ΤΔ (3)

Δύο λύσεις στο πρόβλημα:

■ **A. Ισοδυναμία Ονομάτων** (name equivalence)

Δύο ΤΔ είναι ισοδύναμοι μόνο αν έχουν το ίδιο όνομα. Δηλαδή στο παράδειγμα:

V1 ≠ V2 και **X:= Z** σωστό, **X:= Y, Sub(Y)** λάθος

Πιο δημοφιλής λύση: **Java, C++, Ada**

■ **B. Ισοδυναμία δομών** (structural equivalence)

Δύο ΤΔ είναι ισοδύναμοι αν ορίζουν μεταβλητές με ίδια συστατικά και δομή. Δηλαδή στο παράδειγμα:

V1 = V2 και **X:= Z, X:= Y, Sub(Y)** σωστά

C, Algol-68, FORTRAN, COBOL

Pascal : Συνδυασμός των 2 (Ισοδυναμία Δήλωσης)

4.8 Θέμα 5 Σεπτέμβριος 2014

Στις παρακάτω εντολές της Java και της Pascal οι μεταβλητές X, Y, Z, W είναι τύπου boolean με τρέχουσες τιμές **X=Z=true** και **Y=W=false**

Java: if X==Y && Z==W then X=Z else X=Y

Pascal: if X=Y and Z=W then X:=Z else X:=Y

Τι αποτέλεσμα θα έχουν οι δύο εντολές; Εξηγήστε την απάντηση σας

Απάντηση

Θέσαμε παρενθέσεις για να δείξουμε τη σειρά εκτέλεσης

Java: if ((X==Y) && (Z==W)) then

X=Z;

else

X=Y;

1. Στη Java πρώτα εξετάζονται **πρώτα οι συνθήκες στο if διότι ο τελεστής ισότητας ==** έχει μεγαλύτερη προτεραιότητα από το λογικό τελεστή **&&** (αυτό ισχύει και για τη C και για τη C++). Επειδή η 1^η συνθήκη είναι ψευδής άρα και η συνολική συνθήκη είναι ψευδής (δεν ελέγχεται η 2^η συνθήκη σύμφωνα με τον υπολογισμό περιορισμένης έκτασης)
2. Ο κώδικας πηγαίνει **στο else όπου καταχωρεί στη μεταβλητή X την τιμή της μεταβλητής Y που είναι false**

Pascal: if (X=(Y and Z)=W) then

X:=Z

else

X:=Y

Στην Pascal έχει μεγαλύτερη προτεραιότητα ο λογικός τελεστής AND σε σχέση με τον τελεστή ισότητα =, γιαντό οι έλεγχοι θα γίνουν ως εξής

1. if Y and Z //επειδή το Y είναι false άρα όλη η συνθήκη είναι false
2. if X=((Y and Z)=false) //μετά ελέγχουμε αν το περιεχόμενο της X είναι false. Η X έχει τιμή true άρα η συνθήκη είναι ψευδής (false)
3. if (X=((Y and Z)=false)=W) → μετά ελέγχουμε αν η συνολική συνθήκη που είναι false είναι ίση με την τιμή της W που είναι false. Η W έχει τιμή false άρα η συνθήκη false=false είναι αληθής (true)
4. Άρα η συνολική συνθήκη είναι αληθής και ο κώδικας πηγαίνει στην εντολή then X:=Z και στη μεταβλητή X καταχωρείται η τιμή true

Προτεραιότητα C Προτεραιότητα Pascal

C	Pascal
!	NOT
Postfix ++, --	* / div mod AND
Prefix ++, --	+ - OR
Unary -	= <> < <= > >=
* / %	
Binary + -	
< <= > >=	
== !=	
&&	

Προτεραιότητα Τελεστών Java

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= ^= = <<= >>= >>>=

4.9 Θέμα 4 Ιούνιος 2015

Δίνεται το παρακάτω πρόγραμμα C:

```
main
{
    char ch='w';
    char *chptr=&ch;
    int i=20;
    int *intptr=&i;
    float f=1.2;
    float *fptr=&f;;
    char *ptr="Hello!";
    printf("\n %c, %d, %d, %f, %c, %s \n", *chptr, intptr, *intptr, *fptr, *ptr, ptr);
}
```

- (a) Εξηγήστε όλους τους παραπάνω ορισμούς. Τι είδους μεταβλητές είναι οι οριζόμενες; Ποιες είναι οι αρχικές τιμές τους;
 (b) Τι θα τυπωθεί με την εντολή printf; Εξηγήστε τις απαντήσεις σας

Απάντηση

(a)

`char ch='w';` //δηλώνουμε τη μεταβλητή ch τόπου χαρακτήρα και την αρχικοποιούμε με το χαρακτήρα w

`char *chptr=&ch; //δηλώνουμε το δείκτη chptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής ch. Άρα ο δείκτης chptr θα "δείχνει" στη μεταβλητή ch`

`int i=20; //δηλώνουμε τη μεταβλητή i τύπου ακέραιου και την αρχικοποιούμε με την τιμή 20`

`int *intptr=&i; //δηλώνουμε το δείκτη intptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής i. Άρα ο δείκτης intptr θα "δείχνει" στη μεταβλητή i`

`float f=1.2; //δηλώνουμε τη μεταβλητή f τύπου float και την αρχικοποιούμε με την τιμή 1.2`

`float *fptr=&f; //δηλώνουμε το δείκτη fptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής f Άρα ο δείκτης fptr θα "δείχνει" στη μεταβλητή f`

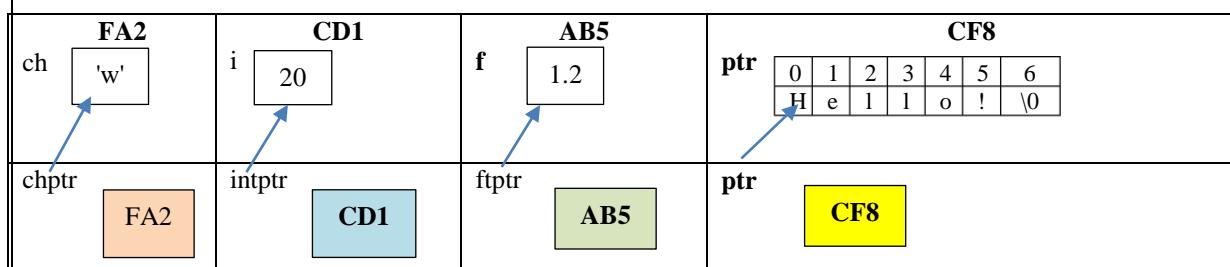
`char *ptr="Hello!"; //δηλώνουμε το δείκτη ptr και τον αρχικοποιούμε με τη λέξη Hello!. Άρα δημιουργείται ένας πίνακας χαρακτήρων με περιεχόμενο Hello! και όνομα ptr. Ο δείκτης ptr δείχνει αυτόματα στον 1o χαρακτήρα του αλφαριθμητικού δηλ. στο χαρακτήρα H`

(b) Τυπώνονται τα ακόλουθα αποτελέσματα: **w, 2686728, 20, 1.200000, H, Hello!**

Επεξήγηση Αποτελεσμάτων

w	Με το %c και το *chptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης chptr</u> δηλ. το <u>περιεχόμενο της μεταβλητής ch</u> που είναι ο χαρακτήρας w. Άρα τυπώνεται ο χαρακτήρας w
2686728	Με το %d και το intptr τυπώνεται ως ακέραιος το περιεχόμενο του δείκτη intptr που είναι η διεύθυνση της μεταβλητής i. Άρα τυπώνεται η διεύθυνση μεταβλητής i.
20	Με το %d και το *intptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης intptr</u> δηλ. το <u>περιεχόμενο της μεταβλητής i</u> που είναι η τιμή 20. Άρα τυπώνεται η τιμή 20
1.200000	Με το %f και το *fptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης fptr</u> δηλ. το <u>περιεχόμενο της μεταβλητής f</u> που είναι η τιμή 1.2. Άρα τυπώνεται η τιμή 1.2 (Πιο τυπικά τυπώνεται η τιμή της f ως 1.200000 δηλ. με ακρίβεια 6 δεκαδικών ψηφίων που είναι η προεπιλεγμένη ακρίβεια των μεταβλητών float στη C)
H	Με το %c και το *ptr <u>τυπώνεται το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης ptr</u> δηλ. τυπώνεται ο αρχικός μόνο χαρακτήρας του αλφαριθμητικού που είναι ο χαρακτήρας H. Άρα τυπώνεται ο χαρακτήρας H
Hello!	Με το %s και το ptr <u>τυπώνεται όλο το αλφαριθμητικό "Hello!"</u> Άρα τυπώνεται το Hello!

Παρατήρηση 1



Παρατήρηση 2

Όταν το * εμφανίζεται σε εντολή δήλωσης υποδηλώνει πάντα ένα δείκτη. Ο δείκτης είναι μια μεταβλητή που ως περιεχόμενο λαμβάνει μια διεύθυνση μνήμης. Π.χ. με τις δηλώσεις **char** *chptr, **float** *fptr και **int** *intptr δηλώνουμε δείκτες

Όταν το * εμφανίζεται σε άλλη εντολή εκτός από δήλωση σημαίνει το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης. Π.χ. στην εντολή printf το *chptr υποδηλώνει το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης chptr που είναι η μεταβλητή ch και έχει ως περιεχόμενο το χαρακτήρα w. Ομοίως στην εντολή printf το *intptr υποδηλώνει το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης intptr που είναι η μεταβλητή i και έχει ως περιεχόμενο τον ακέραιο 20 ή αριθμό 20. Ομοίως στην εντολή printf το *fptr υποδηλώνει το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης fptr που είναι η μεταβλητή f και έχει ως περιεχόμενο τον πραγματικό 1.2 ή αριθμό 1.2.

Παρατήρηση 3

Με την εντολή **char** *ptr="Hello!" ο δείκτης ptr δείχνει στο χαρακτήρα 'H'. Στην εντολή printf με το %c (character) και μεταβλητή εκτύπωσης το *ptr θα τυπωθεί ο χαρακτήρας 'H', ενώ στην ίδια εντολή με το %s (string) και μεταβλητή εκτύπωσης το ptr θα τυπωθεί όλο το αλφαριθμητικό Hello

4.10 Θέμα 3 Φεβρουάριος 2020

Δίνεται το παρακάτω πρόγραμμα C:

```
main
{
    char ch='f';
    char *chptr=&ch;
    int i=200;
    int *intptr=&i;
    float f=1.8;
    float *fptr=&f;
    char *ptr="Hello!";
    printf("\n %c, %d, %d, %f, %c, %s \n", *chptr, intptr, *intptr, *fptr, *ptr, ptr);
}
```

- (a) Εξηγήστε όλους τους παραπάνω ορισμούς. Τι είδους μεταβλητές είναι οι οριζόμενες; Ποιες είναι οι αρχικές τιμές τους;
- (b) Τι θα τυπωθεί με την εντολή printf; Εξηγήστε τις απαντήσεις σας

Απάντηση

(a)

char ch='f'; → δηλώνουμε τη μεταβλητή ch τύπου χαρακτήρα και την αρχικοποιούμε με το χαρακτήρα 'w'

char *chptr=&ch; → δηλώνουμε το δείκτη chptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής ch

int i=200; → δηλώνουμε τη μεταβλητή i τύπου int και την αρχικοποιούμε με την τιμή 200

int *intptr=&i; → δηλώνουμε το δείκτη intptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής i

float f=1.8; → δηλώνουμε τη μεταβλητή f τύπου float και την αρχικοποιούμε με την τιμή 1.2

float *fptr=&f; → δηλώνουμε το δείκτη fptr και τον αρχικοποιούμε με τη διεύθυνση της μεταβλητής f

char *ptr="Hello!"; → δηλώνουμε το δείκτη ptr και τον αρχικοποιούμε με τη φράση Hello!. Άρα δημιουργείται ένας πίνακας χαρακτήρων με περιεχόμενο Hello! και όνομα ptr

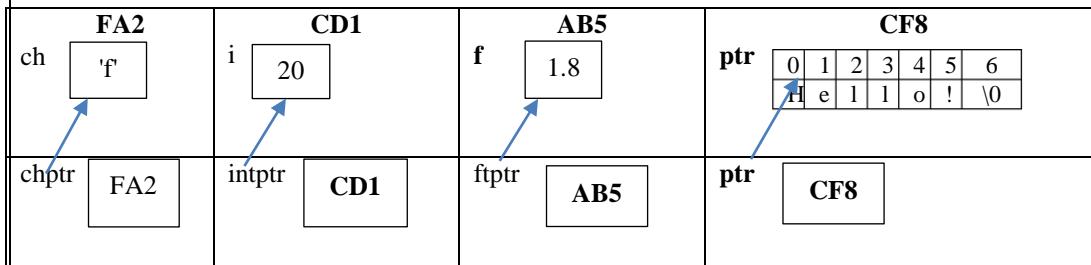
(b) Τυπώνονται τα ακόλουθα αποτελέσματα: **f, 2686728, 200, 1.800000, H, Hello!**

Επεξήγηση Αποτελεσμάτων

f	Με το %c και το *chptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης chptr</u> δηλ. το περιεχόμενο της μεταβλητής ch που είναι ο χαρακτήρας 'f'. Άρα τυπώνεται ο χαρακτήρας 'f'
2686728	Με το %d τυπώνεται ως ακέραιος το περιεχόμενο του δείκτη intptr δηλ. η διεύθυνση της μεταβλητής i
200	Με το %d και το *intptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης intptr</u> δηλ. το περιεχόμενο της μεταβλητής i που είναι η τιμή 200
1.800000	Με το %f και το *fptr τυπώνεται <u>το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης fptr</u> δηλ.. το περιεχόμενο της μεταβλητής f που είναι η τιμή 1.8 (Τυπώνεται ως 1.800000 δηλ. με ακρίβεια 6 δεκαδικών ψηφίων που είναι

	η προεπιλεγμένη ακρίβεια των μεταβλητών float στη C)
H	Με το %c και το *ptr <u>τυπώνεται</u> το περιεχόμενο της μεταβλητής στην οποία δείγνει ο δείκτης ptr δηλ. τυπώνεται ο αρχικός μόνο χαρακτήρας του αλφαριθμητικού που είναι ο χαρακτήρας H
Hello!	Με το %s τυπώνεται όλο το αλφαριθμητικό "Hello!"

Παρατίρηση



4.11 Θέμα 5 Ιούνιος 2015

Στο παρακάτω τμήμα προγράμματος της C, οι μεταβλητές a, b, c, d είναι τύπου integer με τρέχουσες τιμές a=1, b=2, c=3, d=4

```
if (d==0)
    a = d == c + 4*(b = d-1);
else
    a = d == b + 4*(c = d-1);
printf("\n %d, %d, %d, %d \n", a, b, c, d);
```

Τι θα τυπωθεί με την εντολή printf; Εξηγήστε τις απαντήσεις σας;

Απάντηση

Με την εντολή printf θα τυπωθούν οι τιμές: 0, 2, -1, 0

Επεξήγηση Αποτελεσμάτων

- Στην εντολή if (d==0) γίνονται 2 πράγματα: Με την εντολή d==0 στη μεταβλητή d καταχωρείται η τιμή 0. Μετά ελέγχεται η συνθήκη if ((d==0)!=0) διότι όταν στο if ΔΕΝ υπάρχει συνθήκη τότε η προεπιλεγμένη συνθήκη είναι το !=0. Η συνθήκη είναι ψευδής διότι το d έχει πάρει την τιμή 0 οπότε ΔΕΝ ισχύει 0!=0 και ο κώδικας συνεχίζει στο else
- Προσοχή** η εντολή if (d==0) δεν έχει σχέση με την εντολή if (d==0). Το σύμβολο == συμβολίζει καταχώριση ενώ το σύμβολο != συμβολίζει ισότητα
- Μετά στο c καταχωρείται η τιμή -1 διότι το d=0 οπότε c=0-1=-1. Προσοχή η μεταβλητή d έχει αλλάξει τιμή και έχει πάρει την τιμή 0
- Στην έκφραση 4*(c=d-1) πολλαπλασιάζεται το 4 με το αποτέλεσμα της συνθήκης (c=d-1)!=0. Επειδή η c=-1 η συνθήκη c=-1!=0 αληθεύει οπότε πολλαπλασιάζεται το 4 με το 1
- Το αποτέλεσμα προστίθεται στο b που έχει την τιμή 2 οπότε προκύπτει τιμή 2+4*1=6
- Μετά ελέγχεται η συνθήκη d==6 δηλ. 0==6 που είναι ψευδής και το αποτέλεσμα της συνθήκης είναι 0 (false)
- Οπότε στο a καταχωρείται η τιμή 0 που είναι το αποτέλεσμα της προηγούμενης συνθήκης
- Άρα στο printf τυπώνονται οι τιμές **a=0, b=2, c=-1, d=0**

Παρατήρηση

C
!
Postfix ++, --
Prefix ++, --
Unary -
* / %
Binary + -
< <= > >=
== !=
&&

4.12 Θέμα 5 Ιούνιος 2017

Δίνεται το παρακάτω τμήμα προγράμματος C:

```
int a=1; int b=2; int c=3; int d=4;  
if (d=5)  
    a = d == c + 4*(b = d-1);  
else  
    a = d == b + 4*(c = d-1);  
printf("\n %d, %d, %d, %d \n", a, b, c, d);
```

Τι θα τυπωθεί με την εντολή printf; Εξηγήστε τις απαντήσεις σας

Απάντηση

Στο printf τυπώνονται οι τιμές a=0, b=4, c=3 και d=5

Επεξήγηση Αποτελεσμάτων

- Εδώ με την εντολή d=5 στο d καταχωρείται η τιμή 5. Μετά ελέγχεται η συνθήκη if ((d=5)!=0) διότι όταν στο if ΔΕΝ υπάρχει συνθήκη τότε η προεπιλεγμένη συνθήκη είναι το !=0. Η συνθήκη είναι αληθής διότι το d έχει πάρει την τιμή 5 άρα ισχύει 5!=0 και ο κώδικας συνεχίζει στο then. Προσοχή η εντολή if (d=5) δεν έχει σχέση με την εντολή if (d==5). Το σύμβολο = συμβολίζει καταχώριση ενώ το σύμβολο == συμβολίζει ισότητα
- Στο b καταχωρείται η τιμή d-1 δηλ. η τιμή b=5-1=4 διότι η μεταβλητή d έχει αλλάξει τιμή και έχει πάρει την τιμή 5
- Στην έκφραση 4*(b =d-1) πολλαπλασιάζεται το 4 με το αποτέλεσμα της συνθήκης (b=d-1)!=0. Επειδή η b=4 η συνθήκη αληθεύει διότι b=4!=0
- Το αποτέλεσμα της συνθήκης είναι 1 (true)
- Πολλαπλασιάζεται το 4 με το 1 και το αποτέλεσμα προστίθεται στο c που είναι 3 και προκύπτει τιμή 7
- Μετά ελέγχεται η συνθήκη d==7 δηλ. av 5==7 που είναι ψευδής (0)
- Μετά γίνεται η καταχώριση στο a του αποτελέσματος της συνθήκης που είναι 0 άρα στο a καταχωρείται η τιμή 0

Στο printf τυπώνονται οι τιμές a=0, b=4, c=3 και d=5

4.13 Θέμα 2 Ατυπη Νοέμβριος 2011

Περιγράψτε το είδος δεδομένων που ορίζονται με τις παρακάτω δηλώσεις της γλώσσας C:

- (a) **float** *a[n];
- (b) **float** (*b)[n];
- (c) **double** (*c[n])();
- (d) **double** (*d()) [n];

Απάντηση

- (a) Δήλωση Πίνακα με όνομα a, η στοιχείων που το καθένα είναι δείκτης σε float, ουσιαστικά δηλώνουμε ένα πίνακα η δεικτών σε float
- (b) Δήλωση Δείκτη με όνομα b σε πίνακα που περιέχει η στοιχεία τύπου float
- (c) Δήλωση Πίνακα με όνομα c με η στοιχεία που είναι δείκτες σε συνάρτηση που δεν λαμβάνει ορίσματα και επιστρέφει αποτέλεσμα τύπου double
- (d) Δήλωση Συνάρτησης με όνομα d() που δεν λαμβάνει ορίσματα και επιστρέφει δείκτη σε πίνακα με η στοιχεία τύπου double

4.14 Θέμα 6 Ιούνιος 2017

Δίνεται το παρακάτω πρόγραμμα C++:

```
int main()
{
    int *x;
    int *y=new int [2];
    y[0]=10; y[1]=20;
    x=y;
    cout<<y[1]<<" "; cout<<x[0]<<" ";
    delete y;
    cout<<y[0]<<" "; cout<<x[1]<<" ";
}
```

x	DF1	DF2
y	10	20

(a) Εξηγείστε όλους τους ορισμούς, τις εντολές και τη διαχείριση μνήμης του προγράμματος

(β) Τι θα τυπωθεί με την εκτέλεση του προγράμματος; Εξηγείστε το αποτέλεσμα. Ποιο ιδιαίτερο φαινόμενο διαπιστώνετε;

Απάντηση

(a)

Επεξήγηση Εντολών

- Το x δηλώνεται ως δείκτης σε int με την εντολή **int *x;**
- Με την εντολή **new int [2];** δεσμεύονται δυναμικά μνήμη για ένα πίνακα 2 ακεραίων. Δυναμικά σημαίνει κατά τη διάρκεια εκτέλεσης. Η εντολή new αφού δεσμεύσει μνήμη για τον πίνακα, επιστρέφει αυτόματα τη διεύθυνση της 1^{ης} θέσης του πίνακα στο δείκτη y. Ο δείκτης y δείχνει συνεπώς στην 1^η θέση του πίνακα που δημιουργήθηκε δυναμικά

- Με τις εντολές `y[0]=10; y[1]=20;` καταχωρούμε στην 1^η θέση του πίνακα για την τιμή 10 και στη δεύτερη θέση του πίνακα για την τιμή 20. Άρα γεμίζουμε τον πίνακα για τιμές
- Με την εντολή `x=y;` βάζουμε το δείκτη `x` να δείχνει όπου και ο `y` δηλ. ο δείκτης `x` δείχνει στην 1^η θέση του πίνακα που δείχνει και ο δείκτης `y`. Γενικά όταν καταχωρούμε δείκτη σε δείκτη, βάζουμε το δείκτη αριστερά της καταχώρισης να δείχνει όπου και ο δείκτης δεξιά της καταχώρισης άρα γιαντό ο δείκτης `x` δείχνει όπου και ο δείκτης `y`
- Με τις εντολές `cout<<y[1]<<",";` `cout<<x[0]<<",";` τυπώνονται οι τιμές 20 και 10 αντίστοιχα. Επειδή οι δείκτες `x` και `y` δείχνουν στην 1^η θέση του πίνακα, τα στοιχεία `x[0]` και `y[0]` είναι το 10 και αντίστοιχα τα στοιχεία `x[1]` και `y[1]` είναι το 20
- Με την εντολή `delete` για την μνήμη που δείχνει ο δείκτης `y`. Επειδή τα στοιχεία του πίνακα καταλαμβάνουν διαδοχικές θέσεις μνήμης δεν μπορούμε πλέον να προσπελάσουμε τα επόμενα στοιχεία του πίνακα για άρα δεν μπορούμε να προσπελάσουμε το `y[1]`
- Η εντολή `cout<<y[0]<<","` τυπώνει απροσδιόριστο περιεχόμενο διότι έχουμε διαγράψει την 1^η θέση του πίνακα για
- Η εντολή `cout<<x[1]<<","` τυπώνει επίσης απροσδιόριστο περιεχόμενο διότι αφού έχουμε διαγράψει την 1^η θέση του πίνακα που δείχνει ο δείκτης `y` δεν μπορούμε να προσπελάσουμε τις επόμενες θέσεις του πίνακα αφού οι θέσεις ενός πίνακα είναι διαδοχικές

(β)

Αποτελέσματα Εκτέλεσης και εξήγηση Αποτελεσμάτων

Οι τιμές που τυπώνει το πρόγραμμα είναι οι εξής:

- Με την εντολή `cout<<y[1]` τυπώνεται η τιμή 20
- Με την εντολή `cout<<x[0]` τυπώνεται η τιμή 10 επειδή ο δείκτης `x` δείχνει στην 1^η θέση του πίνακα για
- Με την εντολή `cout<<y[0]` επειδή ο δείκτης `y` δεν δείχνει πουθενά τυπώνεται απροσδιόριστο περιεχόμενο π.χ. 13376016
- Με την εντολή `cout<<x[1]` ομοίως τυπώνεται απροσδιόριστο περιεχόμενο π.χ. 0

Ιδιαίτερο Φαινόμενο

Παρατηρούμε ότι έχουμε θέσει δύο δείκτες (τους `x` και `y`) να δείχνουν στην ίδια θέση μνήμης που είναι η αρχική θέση του πίνακα (**φαινόμενο ψευδωνυμίας**). Καταστρέφοντας τη θέση που δείχνει ο ένας δείκτης βάζουμε και τον άλλο δείκτη να δείχνει σε απροσδιόριστο περιεχόμενο

4.15 Θέμα 5 Σεπτέμβριος 2017 και Σεπτέμβριος 2019

a) Εξηγείστε κάθε γραμμή του κώδικα

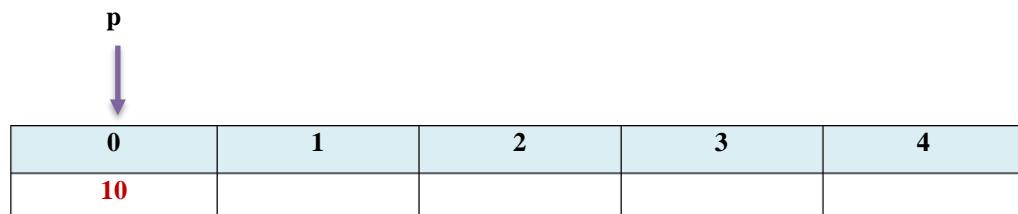
β) Τι εκτυπώνει ο κώδικας;

```
int main()
{
    int numbers[5];
    int *p;
    p=numbers; *p=10;
    p=numbers+3; *p=40;
    p=&numbers[2]; *p=30;
    p=&numbers; *(p+4)=50;
    p++; *p=20;
    for (int i=0; i<5; i++)
        cout<<numbers[i]<<",";
    return 0;
}
```

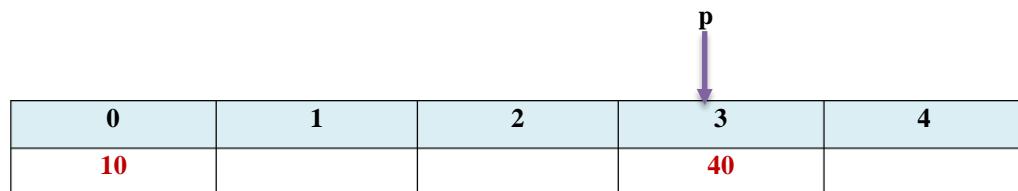
Απάντηση

a)

- `int numbers[5];` //δηλώνουμε πίνακα με όνομα *numbers* 5 ακέραιων
- `int *p;` //δηλώνουμε δείκτη με όνομα *p* σε ακέραιο
- `p=numbers; *p=10;` //με την εντολή *p=numbers* βάζουμε το δείκτη *p* να δείχνει στην 1^η θέση του πίνακα *numbers* διότι το όνομα ενός πίνακα στη C και C++ ταυτίζεται πάντα με τη διεύθυνση του αρχικού του στοιχείου δηλ. `numbers=&numbers[0]` Με την εντολή `*p=10` θέτουμε την τιμή 10 στην 1^η θέση του πίνακα *numbers*



- `p=numbers+3; *p=40;` //με την εντολή *p=numbers+3* μετακινούμε το δείκτη *p* 3 θέσεις δεξιά και δείχνει στο στοιχείο *numbers[3]* του πίνακα *numbers*. Με την εντολή `*p=40` θέτουμε την τιμή 40 στο στοιχείο *numbers[3]*



- `p=&numbers[2]; *p=30;` //με την εντολή *p=&numbers[2]* βάζουμε το δείκτη *p* να δείχνει στο στοιχείο *numbers[2]* του πίνακα *numbers*. Με την εντολή `*p=30` θέτουμε την τιμή 30 στο στοιχείο *numbers[2]*. Το & συμβολίζει διεύθυνση

p

0	1	2	3	4
10		30	40	

- `p=&numbers; *(p+4)=50;` //με την εντολή `p=&numbers[2]` βάζουμε το δείκτη `p` να δείχνει στο στοιχείο `numbers[0]` του πίνακα `numbers` δηλ. στην 1^η θέση του πίνακα `numbers`. Με την εντολή `*(p+4)=40` θέτουμε την τιμή 50 στο στοιχείο `numbers[4]` XΩΡΙΣ ΝΑ ΜΕΤΑΚΙΝΗΣΟΥΜΕ ΤΟ ΔΕΙΚΤΗ `p`. Το `*(p+4)` σημαίνει πήγαινε στο του στοιχείου `numbers[4]` και θέση 50

p

0	1	2	3	4
10		30	40	50

- `p++; *p=20;` //με την εντολή `p++` μετακινούμε το δείκτη `p` 1 θέση δεξιά και δείχνει στο `numbers[1]` του πίνακα `numbers`. Με την εντολή `*p=20` θέτουμε την τιμή 20 στο στοιχείο `numbers[1]`

p

0	1	2	3	4
10	20	30	40	50

- `for (int i=0; i<5; i++)` //με το for εκτυπώνουμε του πίνακα `numbers`
`cout<<numbers[i]<<",";`

b)

Οι τιμές που τυπώνει ο κώδικας είναι 10, 20, 30, 40, 50

0	1	2	3	4
10	20	30	40	50

4.16 Θέμα 3 Ιούνιος 2022 – Ερώτημα α

Δίνεται η παρακάτω εντολή της C στην οποία χρησιμοποιούνται ακέραιες μεταβλητές

`c= a> ++b && b++ && --a;`

Χρησιμοποιήστε παρενθέσεις για να δείξετε με ποια σειρά θα γίνουν οι υπολογισμοί στην παράσταση. Λαμβάνοντας υπόψη την ιδιότητα υπολογισμού περιορισμένης έκτασης (short circuit evaluation) ποιες τιμές έχουν οι μεταβλητές a, b, c μετά την εκτέλεση της παραπάνω εντολής όταν οι αρχικές τιμές των μεταβλητών είναι:

(i) a=2, b=5

(ii) a=5, b=2

Απάντηση

Προτεραιότητα Τελεστών

C
!
Postfix ++, --
Prefix ++, --
Unary -
* / %
Binary + -
< <= > >=
== !=
&&

`c=(((a> ++b)) && (b++)) && (--a);`

1. b++ διότι ο επιθεματικός τελεστής (postfix) έχει τη μεγαλύτερη προτεραιότητα

2. ++b διότι ο προθεματικός τελεστής (prefix) έχει την αμέσως επόμενη μεγαλύτερη προτεραιότητα

3. --a διότι ο προθεματικός τελεστής (prefix) έχει την αμέσως επόμενη μεγαλύτερη προτεραιότητα. Όταν έχουμε τελεστές της ίδιας προτεραιότητας η σειρά εκτέλεσης των πράξεων είναι από αριστερά προς τα δεξιά άρα πρώτα ++b και μετά --a

4. > διότι οι συσχετιστικοί τελεστές έχουν την αμέσως επόμενη μεγαλύτερη προτεραιότητα (η οποία μεγαλύτερη από τους λογικούς τελεστές && και ||)

5. ο αριστερός λογικός τελεστής && δηλ. (++b) && (b++)

6. μετά ο δεξιός λογικός τελεστής && διότι όταν έχουμε τελεστές της ίδιας προτεραιότητας η σειρά εκτέλεσης των πράξεων είναι από αριστερά προς τα δεξιά άρα το αποτέλεσμα της συνθήκης ((++b) && (b++)) με το && (--a)

(i)

Με υπολογισμό περιορισμένης έκτασης και αρχικές τιμές a=2 και b=5

- Πρώτα εκτελείται το b++ και το b θα γίνει 6
- Μετά εκτελείται το ++b και το b παίρνει την τιμή 7

- Μετά εκτελείται το --a και το a παίρνει την τιμή 1
- Μετά ελέγχεται η συνθήκη $a=1>b=7$ που είναι **false** και συνεπώς οι επόμενες συνθήκες δεν εξετάζονται σύμφωνα με το υπόλογισμό περιορισμένης έκτασης λόγω του τελεστή &&. Ότι και αν είναι οι άλλες 2 συνθήκες γνωρίζουμε ότι η συνολική συνθήκη είναι false
- **Στο c μπαίνει η τιμή 0 (false)**
- **Στο τέλος οι τιμές των μεταβλητών είναι a=1, b=7, c=0**

(ii)

Με υπόλογισμό περιορισμένης έκτασης με αργικές τιμές a=5 και b=2

- Πρώτα εκτελείται το b++ και το b θα γίνει 3
- Μετά εκτελείται το ++b και το b παίρνει την τιμή 4
- Μετά εκτελείται το --a και το a παίρνει την τιμή 4
- Μετά ελέγχεται η συνθήκη $a=4>b=4$ που είναι **false** συνεπώς οι επόμενες συνθήκες δεν εξετάζονται
- **Στο τέλος οι τιμές των μεταβλητών είναι a=4, b=4, c=0**

Παρατίρηση Προτεραιότητα Τελεστών στη C

C
!
Postfix ++, --
Prefix ++, --
Unary -
* / %
Binary + -
< <= > >=
== !=
&&

Υποθετική Ερώτηση για a=6, b=2

- Πρώτα εκτελείται το b++ και το b θα γίνει 3
- Μετά εκτελείται το ++b και το b παίρνει την τιμή 4
- Μετά εκτελείται το --a και το a παίρνει την τιμή 5
- Μετά ελέγχεται η συνθήκη $a=5>b=4$ που είναι **true** συνεπώς οι επόμενες συνθήκες εξετάζονται
- Μετά ελέγχεται η συνθήκη b++ είναι !=0 και είναι αληθής
- Μετά ελέγχεται η συνθήκη --a!=0 και είναι αληθής
- **Στο τέλος οι τιμές των μεταβλητών είναι a=5, b=4, c=1**

4.17 Θέμα 3 Ιούνιος 2022 – Ερώτημα β

1	<code>int main()</code>
2	{
3	<code>int data[6]={1, 2, 3, 4, 5, 6}</code>
4	<code>int *p, *q;</code>
5	<code>int a, b, c;</code>
6	<code>p=data;</code>
7	<code>q=p+2;</code>
8	<code>a=*p+*q;</code>
9	<code>b=q[2]+data[2];</code>
10	<code>*(q+1)=0;</code>
11	<code>++p; ++q;</code>
12	<code>c=*p-*q+ data[3];</code>
13	<code>cout<<a<<" " <<b<<" " <<c<<endl;</code>
14	<code>return 0;</code>
15	}

1. Εξηγήστε γραμμή-γραμμή τους ορισμούς και τη λειτουργία του παραπάνω προγράμματος

2. Τι θα τυπωθεί με την εκτέλεση του προγράμματος;

Απάντηση

	Κώδικας	Σχόλια Κώδικα
1	<code>int main()</code>	
2	{	
3	<code>int data[6]={1, 2, 3, 4, 5, 6}</code>	Δήλωση και αρχικοποίηση του πίνακα <code>data</code> με τις τιμές 1, 2, 3, 4, 5, 6 αντίστοιχα
4	<code>int *p, *q;</code>	Δήλωση δύο δεικτών <code>p</code> και <code>q</code> σε ακέραιους
5	<code>int a, b, c;</code>	Δήλωση 3 ακέραιων μεταβλητών <code>a</code> , <code>b</code> , <code>c</code>
6	<code>p=data;</code>	Βάζουμε το δείκτη <code>p</code> να δείχνει στην 1 ^η θέση του πίνακα <code>data</code> δηλ. να δείχνει στο στοιχείο <code>data[0]</code> διότι το όνομα ενός πίνακα ταυτίζεται πάντα με τη διεύθυνση του αρχικού του στοιχείου άρα <code>data=&data[0]</code>
7	<code>q=p+2;</code>	Βάζουμε το δείκτη <code>q</code> να δείχνει όπου και ο δείκτης <code>p</code> συν 2 θέσεις δεξιά δηλ. στην 3 ^η θέση του πίνακα <code>data</code> άρα δείχνει στο στοιχείο <code>data[2]</code> διότι ο <code>p</code> δείχνει στην αρχή του πίνακα
8	<code>a=*p+*q;</code>	Στο <code>a</code> καταχωρείται το άθροισμα του περιεχομένου του στοιχείου του πίνακα που δείχνει ο δείκτης <code>p</code> με το περιεχόμενο του στοιχείου που δείχνει ο δείκτης <code>q</code> άρα $a=1+3=4$ διότι <code>p=&data[0]</code> και <code>*p=1</code> ενώ ο <code>q=&data[2]</code> και <code>*q=3</code>
9	<code>b=q[2]+data[2];</code> <code>//b=*(q+2)+data[2];</code>	Το <code>q[2]</code> σημαίνει μεταφερόμαστε 2 θέσεις δεξιά από τη θέση που δείχνει ο δείκτης <code>q</code> και λαμβάνεται το περιεχόμενο αυτής της θέσης του πίνακα. Άρα λαμβάνεται το περιεχόμενο της θέσης <code>data[4]</code> που είναι η τιμή 5 και σε αντό αθροίζεται το στοιχείο της θέσης <code>data[2]</code> με τιμή 3 οπότε $5+3=8$. ΠΡΟΣΟΧΗ: Ο δείκτης <code>q</code> παραμένει στη θέση του διότι δεν κάνουμε καταχώριση στο δείκτη <code>q</code>
10	<code>*(q+1)=0;</code>	Στην επόμενη θέση του πίνακα από αυτή που δείχνει ο δείκτης <code>q</code> μπαίνει 0 άρα <code>data[3]=0</code> διότι ο δείκτης <code>q</code> δείχνει στο <code>data[2]</code> . ΠΡΟΣΟΧΗ: Ο δείκτης <code>q</code> παραμένει στη θέση του διότι δεν κάνουμε καταχώριση στο δείκτη <code>q</code>
11	<code>++p; ++q;</code>	Μεταφέρουμε τους δείκτες <code>p</code> και <code>p</code> μια θέση δεξιά άρα ο δείκτης <code>p</code> θα δείχνει στο

		στοιχείο $data[2]$ και ο δείκτης q θα δείχνει στο στοιχείο $data[4]$. Τα στοιχεία του πίνακα είναι σε διαδοχικές θέσεις μνήμης άρα οι δείκτες p και q με τη μοναδιαία αύξηση μεταφέρονται στα επόμενα στοιχεία του πίνακα από αυτά στα οποία έδειχναν
12	$c = *p - *q + data[3];$	Στο c καταχωρείται το αποτέλεσμα της παράστασης $*p$ (που είναι το περιεχόμενο της θέσης που δείχνει ο p δηλ. το περιεχόμενο της θέσης $data[1]=2$) μείον την τιμή $*q$ (που είναι το περιεχόμενο της θέσης που δείχνει ο q δηλ. το περιεχόμενο της θέσης $data[3]=0$) μείον το στοιχείο $data[3]$ που είναι 0 άρα στο c καταχωρείται η τιμή $c=2-0+0=2$
13	$cout << a << " " << b << " " << c << endl;$	Τυπώνονται οι τιμές των a , b , c που είναι $a=4$, $b=8$, $c=2$
14	$return 0;$	
15	}	

data

0	1	2	3	4	5
1	2	3	4	5	6



a 4

b 8

c 2

4.18 Θέμα 2 Σεπτέμβριος 2022

Που δεσμεύεται μνήμη και πόση μνήμη σε bytes γρηγοριούει κάθε μεταβλητή; Ποιος ο τρόπος πρόσβασης στη μνήμη του πίνακα κατά την αρχικοποίηση κάθε στοιχείου του;

#define N 1024

static double A[N][N];

```
void init()
{
    int i, j;
    for (j=0 ;j<N; ++j)
        for (i=0; i<N; ++i)
            A[i][j]=(i*j)/(i+j+1);
}
```

Απάντηση

- Ο πίνακας A δηλώνεται ως static μεταβλητή
- Μνήμη για την αποθήκευση του στατικού πίνακα A[N][N] δεσμεύεται στην καθολική (static μνήμη) και όχι στη stack
- Επειδή ο πίνακας A δηλώνεται ως static δεσμεύεται μνήμη γιαντόν κατά την εκκίνηση του προγράμματος και αποδεσμεύεται στο τέλος της εκτέλεσης του προγράμματος
- Κάθε μεταβλητή-στοιχείο του πίνακα επειδή είναι τύπου double **δεσμεύει 8 bytes μνήμης**
- Η συνολική μνήμη που δεσμεύεται για τον πίνακα A είναι: $(8 \times N \times N)$ bytes
- Ο τρόπος πρόσβασης στη μνήμη του πίνακα κατά την αρχικοποίηση κάθε στοιχείου του είναι κατά στήλες διότι στη συνάρτηση init οι εμφωλευμένες επαναλήψεις στους δείκτες i και j εκτελούνται πρώτα κατά στήλη οπότε ο πίνακας A γεμίζει κατά στήλη

Παρατηρήσεις

- Αν ο πίνακας δεν ήταν static μεταβλητή δηλ. αν είχαμε τη δήλωση double A[N][N] τότε θα δέσμευσε μνήμη στη stack
- Αν τα δύο for ήταν με αντίθετη σειρά δηλ. πρώτα το for με το i και μετά το for με το j τότε η αρχικοποίηση του πίνακα A θα ήταν κατά γραμμές

Παρατήρηση

- Αν δηλωθούν δείκτες αυτοί δείχνουν σε αντικείμενα της Heap Μνήμης
- Αν δηλωθούν απλές μεταβλητές αυτές δεσμεύονται μνήμη στη Stack
- Αν δηλωθούν static μεταβλητές αυτές δεσμεύονται μνήμη στη Global Memory

(b) Δίνονται οι εντολές:

```
int *p;  
*p=15;
```

Ποιο το αποτέλεσμα των εντολών αυτών;

Απάντηση

Το αποτέλεσμα που προκύπτει είναι λογικό σφάλμα διότι πρώτα ορίζουμε ένα δείκτη με όνομα p σε ακέραιο και μετά θέτουμε στο περιεχόμενο της μεταβλητής που δείχνει ο δείκτης p την τιμή 15. Επειδή όμως ο δείκτης p δεν δείγνει σε κάποια μεταβλητή δεν μπορεί να καταχωρίσει την τιμή αυτή κάπου. Ουσιαστικά θα έχουμε run-time error.

Το σωστό πρόγραμμα θα ήταν το εξής:

```
int *p, x;  
p=&x;  
*p=15; //x=15
```

Τώρα ο δείκτης p δείχνει στη θέση μνήμης x, οπότε η ανάθεση *p=15 έχει νόημα αφού θέτει στο x την τιμή 15.

4.19 Θέμα 2 Ιούνιος 2023

Δίνεται το παρακάτω τμήμα προγράμματος σε γλώσσα C:

1	int main()
2	{
3	int data[6]={ 10, 20, 30, 40, 50, 60};
4	int *p;
5	p=&data[2];
6	*(p-1)=(*(p-2))-;
7	*p+=1;
8	*++p=44;
9	*p+=33;
10	p++;
11	*p=(*p>50)? *p-1 : *p+1;
12	for (int i=0; i<6; i++) printf("%d\t", data[i]);
13	return 0;
14	}

1. Εξηγήστε γραμμή-γραμμή για τις γραμμές 3-11, τους ορισμούς και τη λειτουργία του παραπάνω προγράμματος
2. Τι θα τυπωθεί με την εντολή του προγράμματος δηλαδή ποιες θα είναι οι τελικές τιμές του πίνακα data

Απάντηση

- ✓ **Γραμμή 3: int data[6]={10, 20, 30, 40, 50, 60}** Με την εντολή αυτή δηλώνουμε και αρχικοποιούμε ένα πίνακα με όνομα data και μέγεθος 6 ακεραίων

0	1	2	3	4	5
10	20	30	40	50	60

- ✓ **Γραμμή 4: int *p** Με την εντολή αυτή δηλώνουμε το δείκτη p σε ακέραιο
✓ **Γραμμή 5: p=&data[2];** Με την εντολή καταχωρούμε στο δείκτη p τη διεύθυνση του στοιχείου data[2] άρα ο δείκτης p δείχνει στο στοιχείο του πίνακα data που βρίσκεται στη θέση 2 όπως δείχνει η επόμενη εικόνα:

0	1	2	3	4	5
10	20	30	40	50	60

p=&data[2]; 

Γραμμή 6: *(p-1)=(*(p-2))-;

- ✓ Με το p-2 βάζουμε το δείκτη p 2 θέσεις αριστερά από αυτή που βρίσκεται (χωρίς όμως να τον μετακινήσουμε). Άρα από τη θέση 2 που βρίσκεται ο δείκτης p τον βάζουμε να δείχνει στη θέση 0 του πίνακα

- ✓ Με το * παίρνουμε το περιεχόμενο αυτής της θέσης που είναι 10
- ✓ Με το p-1 βάζουμε το δείκτη p να δείχνει 1 θέση αριστερά από αυτή που είναι (χωρίς όμως να τον μετακινήσουμε) δηλ. από τη θέση 2 τον βάζουμε να δείχνει στη θέση 1 του πίνακα και με το *(p-1) πάμε στο περιεχόμενο της θέσης 1 και καταχωρούμε την τιμή 10 άρα **στη θέση 1 του πίνακα μπαίνει η τιμή 10**
- ✓ Με τον επιθεματικό τελεστή - αφαιρείται από το περιεχόμενο αυτής της θέσης η τιμή 1 οπότε **στη θέση 0 του πίνακα μπαίνει τελικά η τιμή 9**

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	30	40	50	60



* Γραμμή 7: $p+=1$;

- ✓ Ο δείκτης p δείχνει στη θέση 2 του πίνακα
- ✓ Η πράξη $p+=1$ γράφεται αναλυτικά ως: $*p = *p + 1$. Επειδή ο τελεστής * έχει μεγαλύτερη προτεραιότητα από τον τελεστή + πρώτα αυξάνεται το περιεχόμενο της θέσης που δείχνει ο p δηλ. το 30 κατά ένα και γίνεται 31 και μετά η νέα τιμή καταχωρείται στη θέση 2 του πίνακα

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	31	40	50	60



* Γραμμή 8: $*++p=44$;

- ✓ Πρώτα εκτελείται ο προθεματικός τελεστής αύξησης ++ γιατί έχει μεγαλύτερη προτεραιότητα από τον τελεστή * και βάζει το δείκτη p να δείχνει 1 θέση δεξιά ($p=p+1$) άρα ο δείκτης θα p δείχνει στο data[3]
- ✓ Μετά με το * στο περιεχόμενο αυτής της θέσης καταχωρείται η τιμή 44.

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	31	44	50	60



Γραμμή 9: $*p++=33$;

- ✓ Πρώτα στη θέση που δείχνει ο δείκτης p δηλ. στη θέση 3 του πίνακα τοποθετεί το 33 διότι ο τελεστής * έχει μεγαλύτερη προτεραιότητα από τον επιθεματικό τελεστή ++
- ✓ Μετά ο δείκτης p μεταφέρεται στην επόμενη θέση του πίνακα data

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	31	33	50	60

p



Γραμμή 10: p++;

- ✓ Με το p++ ο δείκτης p μεταφέρεται στην επόμενη θέση του πίνακα data.

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	31	33	50	60

p



Γραμμή 11: *p=(*p>50) ? *p-1 : *p+1;

- ✓ Με τον τριαδικό τελεστή *p=(*p>50)? *p-1 : *p+1; Εξετάζεται πρώτα αν το περιεχόμενο της θέσης που δείχνει ο δείκτης p είναι >50 και εφόσον αυτό ισχύει τότε εκτελείται η εντολή *p-1 ενώ αν δεν ισχύει η συνθήκη εκτελείται η εντολή: *p+1;
- ✓ Ο δείκτης p δείχνει στη θέση 5 του πίνακα data με περιεχόμενο 60. Η συνθήκη *p=60>50 είναι αληθής άρα εκτελείται η εντολή *p=*p-1. Πάμε στο περιεχόμενο της θέσης που δείχνει ο δείκτης p αφαιρούμε το 1 και το κάνουμε 59 και το καταχωρούμε πάλι στην ίδια θέση Συνεπώς *p=59 δηλ. η τιμή 59 μπαίνει στη θέση 5 του πίνακα

Οι τιμές του πίνακα μέχρι τώρα είναι:

0	1	2	3	4	5
9	10	31	33	50	59

p



Γραμμή 12: `for (int i=0; i<6; i++) printf("%d\\t", data[i]);`; Με αυτή την επανάληψη εκτυπώνουμε τον πίνακα

Οι τελικές τιμές του πίνακα είναι:

0	1	2	3	4	5
9	10	31	33	50	59

p



5 PYTHON

5.1 Μεταβλητές

Σε αντίθεση με άλλες γλώσσες προγραμματισμού, η **Python δεν δηλώνει μεταβλητές**. Μια μεταβλητή δημιουργείται τη στιγμή που εκχωρείται μια τιμή σε αυτή. Οι μεταβλητές δεν χρειάζεται να δηλώνονται σε κάποιο συγκεκριμένο τύπο και μπορούν να αλλάξουν τύπο μετά τη δημιουργία τους. Το όνομα μιας μεταβλητής μπορεί να περιλαμβάνει λατινικούς χαρακτήρες ή αριθμούς αλλά δεν μπορεί να αρχίζει με αριθμό.

Οι βασικοί τύποι δεδομένων στην Python είναι:

- <int> ακέραιος
- <float> πραγματικός
- <str> συμβολοσειρά
- <boolean> λογικός τύπος

5.2 Παράδειγμα Δήλωσης και Αρχικοποίησης Μεταβλητών

x=5 #η μεταβλητή x δηλώνεται αυτόματα τύπου int

print(x) #τυπώνεται το περιεχόμενο της x δηλ. το 5

print(type(x)) #τυπώνεται ο τύπος της x δηλ. το int

x=6.5 #η μεταβλητή x μετατρέπεται αυτόματα σε float

print(x) #τυπώνεται το περιεχόμενο της x δηλ. το 6.5

print(type(x)) #τυπώνεται ο τύπος της x δηλ. το float

x="John" #η μεταβλητή x δηλώνεται αυτόματα τύπου string

print(x) #τυπώνεται το περιεχόμενο της x δηλ. το John

print(type(x)) #τυπώνεται ο τύπος της x δηλ. το string

x='Mary' #η μεταβλητή z δηλώνεται αυτόματα τύπου string

print(x) #τυπώνεται το περιεχόμενο της x δηλ. το Mary

print(type(x)) #τυπώνεται ο τύπος της z δηλ. το string

x=True

print(t) #τυπώνεται το περιεχόμενο της t δηλ. το true

print(type(t)) #τυπώνεται ο τύπος της t δηλ. το bool

Αποτελέσματα Εκτέλεσης

5

<class 'int'>

6.5

<class 'float'>

John

<class 'str'>

```
Mary  
<class 'str'>  
True  
<class 'bool'>
```

5.3 Εντολές Εισόδου/Εξόδου

5.3.1 Απλή Εντολή Εξόδου print

Η εντολή `print()` εμφανίζει το περιεχόμενο μεταβλητής ή/ΚΑΙ μήνυμα και ενσωματώνει αυτόματη αλλαγή γραμμής. Για παράδειγμα:

- η εντολή `print(a)` #τυπώνει το περιεχόμενο της μεταβλητής `a`
- η εντολή `print('Καλημέρα')` #τυπώνει το μήνυμα `Καλημέρα`
- η εντολή `print('Αποτέλεσμα =', pi)` #τυπώνει το μήνυμα `Αποτέλεσμα=3.14` και τοποθετεί τον κέρσορα στην επόμενη γραμμή
- η εντολή `print('Αποτέλεσμα =', pi, end=' ')` #τυπώνει το μήνυμα `Αποτέλεσμα=3.14` και ο κέρσορας δεν αλλάζει γραμμή (καταργείται η αυτόματη αλλαγή γραμμής)

Παράδειγμα με δήλωση μεταβλητών και print

```
name="John"  
age=30  
job='Programmer'  
print("Όνομα:", name, " Ηλικία:", age, " Εργασία: ", job)
```

Αποτελέσματα Εκτύπωσης

Όνομα: John Ηλικία: 30 Εργασία: Programmer

5.3.2 Εντολή Εξόδου print με αποτελέσματα σε άγκιστρα

Η εντολή `print` τυπώνει μηνύματα και αποτελέσματα σε {} όταν αρχίζει με το `f`

```
name="John"  
age=30  
job='Programmer'  
print(f "Όνομα: {name} Ηλικία:{age} Εργασία: {job}") #Σε άγκιστρα μπαίνουν όλες τις μεταβλητές που τυπώνονται
```

Αποτελέσματα Εκτύπωσης

Όνομα: John Ηλικία:30 Εργασία: Programmer

5.3.3 Εντολή Εξόδου print με καθορισμό μορφοποίησης

Η εντολή `print` τυπώνει ακέραιες τιμές με το `%d`, πραγματικές τιμές με το `%f`, αλφαριθμητικά με το `%s`, κ.λ.π.

```
name="John"  
age=30  
job='Programmer'  
salary=950.75  
print("Όνομα %s, Ηλικία %d Μισθός %f Εργασία %s" %(name, age, salary, job))
```

Αποτελέσματα Εκτύπωσης

Όνομα John, Ηλικία 30 Μισθός 950.750000 Εργασία Programmer

5.3.4 Εντολή Εισόδου

Η εντολή `input()` εμφανίζει ένα μήνυμα μέσα σε παρένθεση και περιμένει να δώσουμε μια είσοδο. Η είσοδος μας καταχωρείται στη μεταβλητή που βρίσκεται αριστερά του τελεστή =. **Ότι εισάγεται με την εντολή `input()` θεωρείται εξορισμός ως `string`.** Για παράδειγμα:

- η εντολή `name=input('Δώσε το όνομα σου')` εμφανίζει το μήνυμα 'Δώσε το όνομα σου' και καταχωρεί στη μεταβλητή `name` το αλφαριθμητικό που εισάγουμε
- η εντολή `age=input('Δώσε την ηλικία σου')` εμφανίζει το μήνυμα 'Δώσε την ηλικία σου' και καταχωρεί στη μεταβλητή `age` την ηλικία ως αλφαριθμητικό π.χ. αν δώσουμε 30 στο `age` καταχωρείται η τιμή '30'
- η εντολή `salary=input('Δώσε το μισθό σου')` εμφανίζει το μήνυμα 'Δώσε το μισθό σου' και καταχωρεί στη μεταβλητή `salary` το μισθό ως αλφαριθμητικό π.χ. αν δώσουμε 950.75 στο `salary` καταχωρείται η τιμή '950.75'

5.3.4.1 Casting

Με το casting αλλάζουμε τον τύπο μιας μεταβλητής

Παράδειγμα με Casting

```
name=input('Δώσε το όνομα σου: ')
```

```
age=int(input('Δώσε την ηλικία σου: ')) #το δεδομένο που εισάγουμε από το πληκτρολόγιο μετατρέπεται αυτόματα από String σε int μέσω της συνάρτησης int
```

```
salary=float(input('Δώσε το μισθό σου: ')) το δεδομένο που εισάγουμε από το πληκτρολόγιο μετατρέπεται αυτόματα από String σε float μέσω της συνάρτησης float
```

```
print(f'Tο όνομα είναι {name}, η ηλικία είναι {age} και ο μισθός είναι {salary}')
```

```
print(f'Ο τύπος δεδομένων της μεταβλητής name είναι {type(name)})'
```

```
print(f'Ο τύπος δεδομένων της μεταβλητής age είναι {type(age)})'
```

```
print(f'Ο τύπος δεδομένων της μεταβλητής salary είναι {type(salary)})'
```

Αποτελέσματα Εκτέλεσης

```
Δώσε το όνομα σου: John
```

```
Δώσε την ηλικία σου: 40
```

```
Δώσε το μισθό σου: 950.75
```

```
Το όνομα είναι John, η ηλικία είναι 40 και ο μισθός είναι 950.75
```

```
Ο τύπος δεδομένων της μεταβλητής name είναι <class 'str'>
```

```
Ο τύπος δεδομένων της μεταβλητής age είναι <class 'int'>
```

```
Ο τύπος δεδομένων της μεταβλητής salary είναι <class 'float'>
```

5.4 Μερικές Διαφορές Python από C, C++, Java

- Στην Python δεν δηλώνουμε μεταβλητές
- Στην Python **δεν βάζουμε ερωτηματικό στο τέλος των εντολών**
- Στο **if, for, while, do** δεν βάζουμε **άγκιστρα**. Όσες εντολές ανήκουν στο σώμα του if, for, while, do μπαίνουν με εσοχές (tabs)
- Τα δεδομένα **που εισάγουμε με την input είναι όλα τύπου string**
- Στην Python τα **string τοποθετούνται και σε απλά και σε διπλά εισαγωγικά**
- Στην Python δεν υπάρχει τύπος χαρακτήρα, όλα είναι string

5.5 Αριθμητικοί Τελεστές στην Python

- + Πρόσθεση
- - Αφαίρεση
- * Πολλαπλασιασμός
- / Δεκαδική Διαίρεση π.χ. $5/3=1.66$
- // Ακέραια διαίρεση (πηλίκο) π.χ. $5//3=1$
- % Υπόλοιπο διαίρεσης π.χ. $5\%3=2$
- ** Έψωση σε δύναμη π.χ. $5**3=125$

Παράδειγμα με Αριθμητικές Πράξεις

a=`input("Δώσε 1η Τιμή: ")` #Η input είναι η συνάρτηση εισόδου και ότι δώσουμε αντιμετωπίζεται ως string

x=`int(a)` #Το αλφαριθμητικό που είναι στη μεταβλητή μετατρέπεται σε ακέραιο π.χ. '5' → 5

y=`int(input("Δώσε 2η Τιμή: "))` #μπορούμε σε μια εντολή να κάνουμε και εισαγωγή δεδομένων και μετατροπή

sum=x+y

dif=x-y

prod=x*y

rdiv=x/y #Τελεστής πραγματικού πηλίκου άρα στο rdiv μπαίνει το πραγματικό πηλίκο

idiv=x//y #Τελεστής ακέραιου πηλίκου άρα στο idiv μπαίνει το ακέραιο πηλίκο

rem=x%y #Τελεστής υπολοίπου

power=x**y #Τελεστής Δύναμης που υπολογίζει το x^y

`print("Άθροισμα =",sum)`

`print(f"Διαφορά= {dif}")`

`print("Τινόμενο =", prod)`

`print("Ακέραιο Πηλίκο=", idiv)`

`print("Πραγματικό Πηλίκο %.2f" %rdiv)` Με το %.2f στη συνάρτηση print η σύνταξη είναι "σχόλιο %.2f" %μεταβλητή

`print("Υπόλοιπο= %d" %rem)` #Με το %d στη συνάρτηση print η σύνταξη είναι "σχόλιο %d" %μεταβλητή

`print(f"Δύναμη {x}^{y} = {power}")` #Με το f στη συνάρτηση print ισχύει η μορφή: f'σχόλια {μεταβλητές}'

Δώσε 1η Τιμή: 5
Δώσε 2η Τιμή: 6
Αθροισμα = 11
Διαφορά= -1
Γινόμενο = 30
Ακέραιο Πηλίκο= 0
Πραγματικό Πηλίκο 0.83
Υπόλοιπο= 5
Δύναμη $5^6 = 15625$

5.5.1 Συντομογραφίες Αριθμητικών πράξεων

x += y # $x = x + y$

x -= y # $x = x - y$

x *= y # $x = x * y$

x /= y # $x = x / y$

x // y # $x = x // y$ Ακέραια διαιρεση του x με y

x%y # $x = x \% y$ Υπόλοιπο ακέραιας διαιρεσης του x με y

xy** # $x = x^{**}y$ Ύψωση του x στο y

Παρατήρηση

Τα διευκρινιστικά σχόλια στην Python (σχόλια προγραμματιστή) συμβολίζονται με το χαρακτήρα # στη αρχή της γραμμής.

5.5.2 Boolean (Λογικοί) Τελεστές

Οι βασικοί τελεστές για εκφράσεις Boolean είναι

- **not** (άρνηση)
- **and** (σύζευξη)
- **or** (διάζευξη)

Παραδείγματα με Boolean Τελεστές

<pre> >>> a = 5 >>> b = 4 >>> a > b True >>> c = 6 >>> a > b and a > c False >>> a > b or a > c True >>> a > b and not a > c True >>> a > b or not a > c True >>> not a > b or not a > c True >>> not (a > b) or a > c False >>> not a > b or a > c False </pre>	<pre> print(1>2 and 2>3) #False print(not(1>2 and 2>3)) #True print(1>2 or 2<3) #True print(1>2 and 2>3 or 1<4) #True </pre>
--	---

5.6 Τελεστές == id και is

Η Python έχει τους τελεστές σύγκρισης **==** και **is**. Ο τελεστής **==** συγκρίνει δύο μεταβλητές ελέγχοντας το περιεχόμενο τους, ενώ ο τελεστής **is** συγκρίνει δύο μεταβλητές και επιστρέφει TRUE αν οι δύο μεταβλητές αναφέρονται στο ίδιο αντικείμενο.

Στο επόμενο παράδειγμα παρουσιάζονται αυτοί οι τελεστές για 3 μεταβλητές με ακέραιες τιμές. **Όταν δύο μεταβλητές έχουν το ίδιο περιεχόμενο τότε έχουν και το ίδιο id δηλ. είναι οι ίδιες θέσεις μνήμης**

a	1	b	2	c
a=1		10771520		10771552
b=1				
c= 2				

print(a is b) #επιστρέφει **true** διότι τα ονόματα *a*, *b* αναφέρονται στην ίδια θέση μνήμης δηλ. στην ίδια μεταβλητή (ίδιο αντικείμενο)

print(a is c)#επιστρέφει **false** διότι τα ονόματα *a*, *c* ΔΕΝ αναφέρονται στην ίδια θέση μνήμης

print(id(a)) #τυπώνεται η διεύθυνση της μεταβλητής *a*

print(id(b)) #τυπώνεται η διεύθυνση της μεταβλητής *b*. Παρατηρούμε ότι οι *a* και *b* είναι οι ίδιες θέσεις μνήμης αφού έχουν ίδια διεύθυνση

print(id(c)) #τυπώνεται η διεύθυνση της μεταβλητής *c*

a=2 #αφού η *a* αποκτά το ίδιο περιεχόμενο με τη *c* γίνονται ίδιες θέσεις μνήμης

print(a is b) #επιστρέφει **false** διότι τα ονόματα *a*, *b* ΔΕΝ αναφέρονται στην ίδια θέση μνήμης αφού ΔΕΝ έχουν το ίδιο περιεχόμενο

print(a is c)#επιστρέφει **true** διότι τα ονόματα *a*, *c* αναφέρονται στην ίδια θέση μνήμης αφού έχουν το ίδιο περιεχόμενο

print(id(a)) #τυπώνεται η διεύθυνση της μεταβλητής *a*

print(id(b)) #τυπώνεται η διεύθυνση της μεταβλητής *b*

print(id(c)) #τυπώνεται η διεύθυνση της μεταβλητής *c*. Παρατηρούμε ότι οι *a* και *b* είναι οι ίδιες θέσεις μνήμης αφού έχουν ίδια διεύθυνση

print(a==c) #επιστρέφει **true** διότι τα ονόματα *a*, *c* έχουν το ίδιο περιεχόμενο

print(a==b) #επιστρέφει **false** διότι τα ονόματα *a*, *b* έχουν διαφορετικό περιεχόμενο

Παρατήρηση

Ο τελεστής **is** συγκρίνει **id** αντικειμένων (δηλ. διευθύνσεις μεταβλητών) ενώ ο τελεστής **==** συγκρίνει περιεχόμενα αντικειμένων

5.7 Συμβολοσειρές (Strings)

Μια συμβολοσειρά στην Python είναι μια ακολουθία χαρακτήρων που περικλείεται σε μονά ή διπλά εισαγωγικά. Το 'hello' είναι το ίδιο με το "hello". Τα αλφαριθμητικά μπορούν να εμφανιστούν στην οθόνη χρησιμοποιώντας την εντολή print. Για παράδειγμα print("hello") τυπώνεται στην οθόνη η λέξη hello. **H** Python δεν έχει τύπο δεδομένων char, ένας χαρακτήρας είναι μια συμβολοσειρά μήκους 1. Οι αγκύλες [] μπορούν να χρησιμοποιηθούν για την πρόσβαση στα στοιχεία μιας συμβολοσειράς.

Αποθήκευση Αλφαριθμητικού

a = "Hello,World!" #To a μετατρέπεται σε πίνακα χαρακτήρων (αλφαριθμητικό). Σε κάθε κελί του a μπαίνει ακριβώς ένας χαρακτήρας

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o	,	W	o	r	l	d	!

Με την επόμενη εντολή τυπώνεται ο χαρακτήρας στη θέση 1 του αλφαριθμητικού a (Ο 1ος χαρακτήρας βρίσκεται πάντα στη θέση 0): print(a[1]) #Τυπώνεται ο χαρακτήρας e

Αποκοπή τμήματος Αλφαριθμητικού

a = "abcdefghijkl"

0	1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h	i

Με τον τελεστή : προσδιορίζουμε ένα συγκεκριμένο τμήμα ενός αλφαριθμητικού. Ο γενικός συμβολισμός του τελεστή : είναι

αλφαριθμητικό[αρχή:τέλος]

και προσδιορίζονται όλοι οι χαρακτήρες του από την ΑΡΧΗ μέχρι και τον ΠΡΟΤΕΛΕΥΤΑΙΟ

Παραδείγματα Αποκοπής Τμήματος Αλφαριθμητικού (Slicing)

a[4:7] #τυπώνεται 'efg', δηλαδή τυπώνονται οι χαρακτήρες από τον 4^ο μέχρι τον 6^ο. Δεν συμπεριλαμβάνεται ο 7^{ος} κατά σειρά χαρακτήρας

0	1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h	i

a[:4] #τυπώνεται 'abcd', δηλαδή οι χαρακτήρες από τον 0^ο μέχρι τον 3^ο. Δεν συμπεριλαμβάνεται ο 4^{ος} κατά σειρά χαρακτήρας

0	1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h	i

a[4:] #τυπώνεται 'efghi' δηλαδή τυπώνονται οι χαρακτήρες από το χαρακτήρα στη θέση 4 μέχρι και τον τελευταίο. Συμπεριλαμβάνεται και ο τελευταίος χαρακτήρας αφού δεν υπάρχει προσδιορισμός μετά την :

0	1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h	i

word='Python'

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

- word[0] #ο χαρακτήρας του αλφαριθμητικού Python στη θέση 0 δηλ. το 'P'
- word[5] #ο χαρακτήρας του αλφαριθμητικού Python στη θέση 5 δηλ. το 'n'

Οι δείκτες μέσα στη αγκύλη του αλφαριθμητικού μπορεί να είναι και αρνητικοί αριθμοί. Σε αυτή την περίπτωση μετρούν από δεξιά προς αριστερά ξεκινώντας από το -1 όπου υποδηλώνει την τελευταία θέση του αλφαριθμητικού

- word[-1] #ο τελευταίος χαρακτήρας του αλφαριθμητικού Python δηλ. το 'n'
- word[-2] #ο προτελευταίος χαρακτήρας του αλφαριθμητικού Python δηλ. το 'o'
- word[-6] #ο αρχικός χαρακτήρας του αλφαριθμητικού Python δηλ. το 'P'

a = "abcdefghijkl"

```
print(a[-1]) #τυπώνεται ο τελευταίος χαρακτήρας δηλ. το i
print(a[-2]) #τυπώνεται ο προτελευταίος χαρακτήρας δηλ. το h
print(a[-3]) #τυπώνεται ο τρίτος χαρακτήρας από το τέλος δηλ. το g
print(a[-9]) #τυπώνεται ο αρχικός χαρακτήρας από το τέλος δηλ. το a
```

Υποσυμβολοσειρές-Τεμαγισμός

b="Hello, World!" → το b μετατρέπεται αντόματα σε πίνακα χαρακτήρων

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o	,	W	o	r	l	d	!

print(b[2:5]) → Τυπώνονται οι χαρακτήρες llo. Με τον επόμενο κώδικα λαμβάνονται οι χαρακτήρες από τη θέση 2 έως τη θέση 5 του αλφαριθμητικού b (η θέση 5 δεν συμπεριλαμβάνεται):

word='Python'

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

word[0:2] → οι χαρακτήρες από τη θέση 0 (συμπεριλαμβάνεται) μέχρι τη θέση 2 (εξαιρείται) δηλ. οι χαρακτήρες 'Py'

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

word[2:5] → οι χαρακτήρες από τη θέση 2 (συμπεριλαμβάνεται) μέχρι τη θέση 5 (εξαιρείται) δηλ. οι χαρακτήρες 'tho'

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

Συμπερίληψη όλου του αλφαριθμητικού

Η αρχή του αλφαριθμητικού **περιλαμβάνεται πάντα στον τελεστή** : ενώ το τέλος εξαιρείται. Αυτό εξασφαλίζει ότι **s[:i]+s[i:]** είναι ίσο με το συνολικό αλφαριθμητικό s διότι η έκφραση s[:i] περιλαμβάνει τους πρώτους i-1 χαρακτήρες ενώ η έκφραση s[i:] περιλαμβάνει τους χαρακτήρες από τον i-οστό μέχρι και τον τελευταίο

word='Python'

word[:2]+ word[2:] #τυπώνεται το αλφαριθμητικό 'Python'. O χαρακτήρας + συνενώνει το αριστερό με το δεξιό αλφαριθμητικό

word[4:]+ word[4:] #τυπώνεται το αλφαριθμητικό 'Python' O χαρακτήρας + συνενώνει το αριστερό με το δεξιό αλφαριθμητικό

word[-2:] # τυπώνονται οι 2 τελευταίοι χαρακτήρες από το τέλος (συμπεριλαμβάνεται και το τέλος) δηλ. το 'on'. Αυτό σημαίνει ότι αρχίζει 2 χαρακτήρες πριν το τέλος και πηγαίνει μέχρι και τον τελευταίο χαρακτήρα

'J' + word[1:] # τυπώνεται το αλφαριθμητικό 'Jython' (μετά το χαρακτήρα J συνενώνει το αλφαριθμητικό που είναι στη μεταβλητή word από το 2^o χαρακτήρα μέχρι και τον τελευταίο)

word[:2] + 'py' # τυπώνεται το αλφαριθμητικό 'Pypy'. To :2 λαμβάνει τους χαρακτήρες στις θέσεις 0 και 1 δηλ. τους Py και συνενώνει στο τέλος τους το αλφαριθμητικό 'py' και προκύπτει το 'Pypy'

word[:] # Αν λείπουν και οι δύο αριθμοί τυπώνεται όλο το αλφαριθμητικό 'Python'

Παρατήρηση

Ένα τμήμα δεικτών έχει χρήσιμες προεπιλογές. Είναι δυνατόν να παραλείπεται το πρώτο στοιχείο, όταν η τιμή του δείκτη ισούται με μηδέν ή να παραλείπεται το τέλος αυτού αφού το μέγεθος της συμβολοσειράς είναι σταθερό.

word[:2] → οι χαρακτήρες από τη θέση 0 (συμπεριλαμβάνεται) μέχρι τη θέση 2 (εξαιρείται) δηλ. το 'Py'

word[4:] → οι χαρακτήρες από τη θέση 4 (συμπεριλαμβάνεται) μέχρι το τέλος δηλ. το 'on'

Εάν χρειαζόμαστε μια διαφορετική συμβολοσειρά, θα πρέπει να δημιουργήσουμε μια νέα.

Παρατήρηση

Ο τελεστής + κάνει συνένωση αλφαριθμητικών

Παραδείγματα Προσπέλασης Λίστας με τον τελεστή :

a[0]

- 'London'

a[2]

- 1452

Τα μέλη μιας λίστας μπορεί να είναι διαφορετικού τύπου, η Python όμως τα αντιμετωπίζει το καθένα με τον τύπο του. Στο προηγούμενο παράδειγμα η λίστα έχει τα δύο πρώτα μέλη της (London, Rome) τύπου string και τα δύο επόμενα (1451, 9) τύπου int.
a[0]+a[1]

- 'LondonRome'

a[2]+a[3]

- 1461

Ο αριθμός που προσδιορίζει τη θέση ενός μέλους σε μια λίστα μπορεί να είναι αρνητικός, οπότε ξεκινάμε να υπολογίζουμε τη θέση του μετρώντας από το τέλος προς την αρχή. **Η θέση του τελευταίου στοιχείου της λίστας μπορεί να προσδιοριστεί και με το -1.** Συνεπώς στη λίστα a=['London', 'Rome', 1452, 9] ισχύει ότι:

a[3]

• 9

a[-1]

• 9

a[-3]

• 'Rome'

a[1:3]

• ['Rome', 1452]

Το a[1:3] σημαίνει ότι από τη λίστα a παίρνουμε το μέλος a[1] που είναι το 2^ο στοιχείο της και όλα τα επόμενα μέχρι το a[3] χωρίς να συμπεριλαμβάνεται το a[3] δηλ. παίρνουμε δύο μέλη.

Μπορούμε να έχουμε και αρνητικούς αριθμούς χωρίς πρόβλημα:

a[1:-1] → από το στοιχείο που είναι από τη θέση 1 μέχρι και το στοιχείο που είναι στην τελευταία θέση χωρίς να συμπεριλαμβάνεται το τελευταίο στοιχείο (δηλ. μέχρι και αντό που είναι στην προτελευταία θέση)

• ['Rome', 1452]

Αν λείπει ο πρώτος αριθμός σημαίνει ότι εννοούμε την αρχή της λίστας δηλ. τη θέση 0

a[:3] → από τη θέση 0 τα επόμενα 3 στοιχεία της λίστας a

• ['London', 'Rome', 1452]

Αν λείπει ο δεύτερος αριθμός θέλουμε να συμπεριληφθεί στο τμήμα της μέχρι και το τελευταίο στοιχείο της λίστας.

a[2:]

• [1452, 9]

a[2:4]

• [1452, 9]

Στο a[2:4], το 4 σημαίνει ότι το τελευταίο μέλος της λίστας θα είναι αυτό με τη θέση 3 γιατί στη λίστα μας δεν υπάρχει θέση 4

Αν λείπουν και οι δύο αριθμοί παίρνουμε όλα τα μέλη της λίστας στο τμήμα μας **εκτός του τελευταίου**

a[:]

• ['London', 'Rome', 1452]

squares=[1,2,4, 9, 16, 25]

squares

• [1, 2, 4, 9, 16, 25]

squares[0]

• 1

squares[-1]

- **25**

squares[-3:]

- **[9, 16, 25]**

5.7.1 Συναρτήσεις Διαχείρισης Συμβολοσειρών

Συνάρτηση strip

Με τη συνάρτηση strip() αφαιρούνται όλα τα κενά διαστήματα από την αρχή και το τέλος ενός αλφαριθμητικού
a = "Hello, World!"

```
print(a.strip()) #επιστρέφει "Hello, World!"
```

Συνάρτηση len

Με τη συνάρτηση len() υπολογίζεται το μήκος ενός αλφαριθμητικού
a = "Hello, World!"

```
print(len(a)) #τυπώνεται το μήκος της συμβολοσειράς που είναι 13 χαρακτήρες
```

Συνάρτηση lower

Η συνάρτηση lower() επιστρέφει το αλφαριθμητικό με πεζούς χαρακτήρες
a = "Hello, World!"

```
print(a.lower())#τυπώνεται το αλφαριθμητικό hello, world!
```

Συνάρτηση upper

Η συνάρτηση upper() επιστρέφει το αλφαριθμητικό με κεφαλαίους χαρακτήρες
a = "Hello, World!"

```
print(a.upper()) #τυπώνεται το αλφαριθμητικό HELLO, WORLD!
```

Συνάρτηση replace

Η συνάρτηση replace() αντικαθιστά όλες τις εμφανίσεις του 1^{ου} αλφαριθμητικού με το 2^ο
a = "Hello, World!"

```
print(a.replace("H", "J"))#τυπώνεται το αλφαριθμητικό Jello, World
```

a = "Hello, World!"

```
print(a.replace("l", "k"))#τυπώνεται το αλφαριθμητικό Hekko, Workd
```

Συνάρτηση split

Η συνάρτηση split() διασπά ένα αλφαριθμητικό σε υπο-αλφαριθμητικά αν βρει εμφανίσεις του διαχωριστή που ορίζουμε και επι-

στρέφει ένα πίνακα με τα επιμέρους υπο-αλφαριθμητικά

a = "Hello, World!"

```
print(a.split(","))#Το αλφαριθμητικό 'Hello, World' διασπάται σε δύο υπο-αλφαριθμητικά βάσει των χαρακτήρα , και συγκεκριμένα προκύπτουν τα υπο-αλφαριθμητικά 'Hello' και 'World' και τυπώνεται ένας πίνακας με τα δύο αλφαριθμητικά ['Hello' 'World!']
```

```
print(a.split("o"))# Το αλφαριθμητικό 'Hello, World' διασπάται σε διαφορετικά υπο-αλφαριθμητικά βάσει κάθε χαρακτήρα ο και γιαν- τό τυπώνεται ο πίνακας ['Hell', 'W', 'rld!']
```

Συνάρτηση count

Η συνάρτηση count υπολογίζει το συνολικό αριθμό εμφανίσεων ενός αλφαριθμητικού ή ενός χαρακτήρα σε ένα string

```
string1="Python Programming with Python Strings in Python"
```

print(f" Το συνολικό πλήθος όλων των εμφανίσεων της λέξης 'Python' στο string '{string1}' είναι {string1.count('Python')}") #To συνολικό πλήθος όλων των εμφανίσεων της λέξης 'Python' στο string 'Python Programming with Python Strings in Python' είναι 3

print(f" Πλήθος Εμφανίσεων Χαρακτήρα 'P' στο string {string1} = {string1.count('P')}") #Υπολογίζει το πλήθος όλων των εμφανίσεων του κεφαλαίου P στο string1. Πλήθος Εμφανίσεων Χαρακτήρα 'P' στο string Python Programming with Python Strings in Python = 4

Συνάρτηση find

```
string1="Python Programming with Python Strings in Python"
```

Η συνάρτηση find εντοπίζει μόνο τη θέση της 1^{ης} εμφάνισης ενός αλφαριθμητικού ή ενός χαρακτήρα σε ένα string

print(f "Η Θέση της 1ης εμφάνισης της λέξης 'Python' στο string {string1} με συνάρτηση find είναι η {string1.find('Python')}") #Η συνάρτηση find εντοπίζει τη θέση της 1ης μόνο εμφάνισης ενός αλφαριθμητικού σε ένα string H Θέση της 1^{ης} εμφάνισης της λέξης 'Python' στο string Python Programming with Python Strings in Python με συνάρτηση find είναι η 0. Αν η find δεν εντοπίσει το ζητούμενο στοιχείο επιστρέφει ως απάντηση το -1

Συνάρτηση index

```
string1="Python Programming with Python Strings in Python"
```

Η συνάρτηση index εντοπίζει μόνο τη θέση της 1^{ης} εμφάνισης ενός αλφαριθμητικού ή ενός χαρακτήρα σε ένα string

print(f "Η Θέση της 1ης εμφάνισης της λέξης 'Python' στο string {string1} με συνάρτηση index είναι η {string1.index('Python')}") #Η συνάρτηση index εντοπίζει τη θέση της 1^{ης} μόνο εμφάνισης ενός αλφαριθμητικού σε ένα string. H Θέση της 1^{ης} εμφάνισης της λέξης 'Python' στο string Python Programming with Python Strings in Python με συνάρτηση index είναι η 0. Αν η index δεν εντοπίσει το ζητούμενο στοιχείο δεν επιστρέφει καμία απάντηση

Σύγκριση Συμβολοσειρών με == και !=

Η λειτουργία των τελεστών == και != στις συμβολοσειρές είναι ακριβώς η ίδια όπως και στους αριθμούς (int και float). Ο τελεστής == επιστρέφει TRUE αν υπάρχει ακριβές ταίριασμα των συμβολοσειρών (δηλ. αν οι συμβολοσειρές είναι ίδιες) και FALSE σε διαφορετική περίπτωση. Ο τελεστής != έχει ακριβώς την αντίθετη λειτουργία από τον τελεστή ==. Αν οι συμβολοσειρές που συγκρίνονται είναι διαφορετικές μεταξύ τους επιστρέφει TRUE αλλιώς επιστρέφει FALSE

Παραδείγματα Σύγκρισης Συμβολοσειρών

```
a = "Test"
```

```
b="test"
```

```
d="TEST"
```

```
c=a == b
```

```
print(f{a} == {b} {c} ) #τυπώνει False
```

```
c=a!= b
```

```
print(f'{a} != {b} {c}') #τυπώνει True
```

c=a == d

```
print(f'{b} == {d} {c}') #τυπώνει False
```

c=a.lower() == b.lower()

```
print(f'{a.lower()} == {b.lower()} {c}') #τυπώνει True
```

c=a.upper() == b.upper()

```
print(f'{a.upper()} == {b.upper()} {c}') #τυπώνει True
```

Αποτελέσματα Εκτέλεσης

Test == test False

Test != test True

test == TEST False

test == test True

TEST == TEST True

5.7.2 Πράξεις Συμβολοσειρών

Οι συμβολοσειρές αποτελούν αμετάβλητες ακολουθίες χαρακτήρων δηλαδή δεν επιτρέπεται η αλλαγή τους. Επιτρέπεται όμως η πρόσθεση συμβολοσειρών όπως και ο πολλαπλασιασμός συμβολοσειράς με ακέραιο.

τελεστής	αποτέλεσμα
<seq> + <seq>	συνένωση
<seq> * <int>	επανάληψη
<seq>[]	δείκτης
len(<seq>)	μήκος ακολουθίας
<seq>[:]	τεμαχισμός
for <var> in <seq>:	επανάληψη
<expr> in <seq>	συμμετοχή (Boolean)

Παράδειγμα 1 – Συνένωση Συμβολοσειρών

```
>>> a = 'Καλή'  
>>> b = 'μέρα'  
>>> c = a + b  
>>> print(c)      → Τυπώνεται Καλημέρα
```

Παράδειγμα 2– Επανάληψη Συμβολοσειράς

Μπορούμε να πολλαπλασιάσουμε ένα string με αριθμό.

```
>>> a = 'Ηχώ'  
>>> b = a * 5  
>>> print(b)      #Τυπώνεται ΗχώΗχώΗχώΗχώΗχώ
```

Παράδειγμα 3– Τελεστής in και not in

str="My name is Python"

`print("is" in str)` #ελέγχει αν το αλφαριθμητικό is περιέχεται μέσα στο αλφαριθμητικό str και επειδή εδώ αντό συμβαίνει την πάνεται η τιμή True

`print("is" not in str)` #ελέγχει αν το αλφαριθμητικό is δεν περιέχεται μέσα στο αλφαριθμητικό str και επειδή εδώ αντό δεν συμβαίνει τυπώνεται η τιμή False

5.8 Δομές Εντολών στην Python

5.8.1 Δομή Επιλογής - Εντολή if

Παράδειγμα 1

```
x=int(input("Δώσε αριθμό"))
```

if x<0: #πρέπει σε κάθε τιμήμα του if (άρα και στο then) να υπάρχει ο τελεστής :

print('Αρνητικός Αριθμός') #κάθε εντολή μέσα στο σώμα του if (εδώ η print) πρέπει να μπει υποχρεωτικά σε εσοχή πατώντας tab αλλιώς είναι συντακτικό λάθος. Με το tab αντιλαμβάνεται το if ποιες εντολές είναι στο σώμα του

elif x==0: #πρέπει σε κάθε else τιμήμα του if να υπάρχει ο τελεστής : και η αντίστοιχη εσοχή

print('Μηδέν')#κάθε εντολή print πρέπει να μπει σε εσοχή πατώντας tab αλλιώς είναι συντακτικό λάθος

else:

print('Θετικός')#κάθε εντολή print πρέπει να μπει σε εσοχή πατώντας tab αλλιώς είναι συντακτικό λάθος

Παράδειγμα 2

```
a = 200
```

```
b = 33
```

```
if b>a:
```

print("b is greater than a")

```
elif a == b:
```

print("a and b are equal")

```
else:
```

print("a is greater than b")

Παράδειγμα 3 Εντολή if μιας γραμμής

```
if a > b: print("a is greater than b")
```

Παράδειγμα 4 Εντολή if-else μιας γραμμής

Σε αυτή τη μορφή της if πρώτα γράφεται πρώτα το αποτέλεσμα δηλ. η ενέργεια που θα εκτελείται και μετά η συνθήκη που ελέγχεται

```
print("A") if a>b else print("=") if a == b else print("B") #εννοείται ότι θα ισχύει if a<b
```

```
a=340
```

```
b=330
```

Ισχύει η συνθήκη a>b οπότε τυπώνεται A

```
a=330
```

```
b=330
```

Ισχύει η συνθήκη a==b οπότε τυπώνει =

```
a=330
```

```
b=340
```

δεν ισχύει καμία συνθήκη a>b οπότε τυπώνει B

Παράδειγμα 5 Εντολή if-else με λογικούς τελεστές

```
a = 200
```

```
b = 33
```

```
c = 500
```

```
if a > b or a > c:
```

```
    print("At least one of the conditions is True") #αν ισχύει είτε η μια είτε η άλλη συνθήκη
```

Παράδειγμα 6 Εντολή nested if

```
#Εισαγωγή Βαθμού και χαρακτηρισμός του ως εξής [0,5]->αποτυχία, [5,7]->μέτρια (7,10]->άριστα οπιδήποτε άλλο->Λάθος
```

```
x=float(input("Δώσε Βαθμό: "))
```

```
if x>=0 and x<5:
```

```
    print(f" Ο Βαθμός {x} είναι αποτυχία")
```

```
elif x>5 and x<=7:
```

```
    print(f" Ο Βαθμός {x} είναι μέτριος")
```

```
elif x>7 and x<=10:
```

```
    print(f" Ο Βαθμός {x} είναι άριστος")
```

```
elif x<0 or x>10:
```

```
    print(f" Ο Βαθμός {x} είναι εσφαλμένος")
```

5.8.2 Δομές Επανάληψης- Εντολή for

Παράδειγμα 1

Εισαγωγή 5 τιμών και χαρακτηρισμός τους ως άρτια ή περιττή. Η συνάρτηση range δεν λαμβάνει την τελική της τιμή

for i in range(1, 6): #i=1, i<6, i++. Η συνάρτηση range(a, b) αφορά ΠΑΝΤΑ το διάστημα [a, b]. Εδώ αφορά το διάστημα [1, 6]

```
x=int(input("Δώσε τιμή: "))
```

if x%2==0:

```
    print(f" Η τιμή {x} είναι άρτια")
```

else:

```
    print(f" Η τιμή {x} είναι περιττή")
```

Παράδειγμα 2

Εύρεση ΜΟ από n τυχαίους βαθμούς

```
sum=0 #αρχικοποίηση αθροιστή
```

```
n=int(input("Δώσε Πλήθος Βαθμών: "))
```

for i in range(1, n+1): #i=1, i<n+1, i++. Αυτό το for εκτελεί n επαναλήψεις. Η συνάρτηση range(a, b) αφορά ΠΑΝΤΑ το διάστημα [a, b]. Εδώ αφορά το διάστημα [1, n+1)

```
x=float(input("Δώσε Βαθμό: "))
```

sum+=x #Εναλλακτικά sum=sum+x. To sum είναι αθροιστής διότι αθροίζει τιμές

```
print(f" Ο Πραγματικός Μέσος Όρος είναι {sum/n}")
```

```
print(f" Ο Ακέραιος Μέσος Όρος είναι {sum//n}")
```

Παράδειγμα 3- Συνάρτηση range στο for

Η συνάρτηση range() δίνει εύρος τιμών στη μεταβλητή επανάληψης. Θέτει εξορισμό το 0 ως αρχική τιμή αλλά μπορούμε να προσδιορίσουμε και διαφορετική αρχική τιμή προσθέτοντας μια παράμετρο.

for x in range(2, 6): #Η συνάρτηση range(2, 6) περιλαμβάνει σε όλες τις τιμές μέχρι την προτελευταία δηλ. από 2 έως 5 (χωρίς το 6). Εδώ αφορά το διάστημα [2, 6). Σε κάθε επανάληψη στο x μπαίνει ακέραια τιμή από το διάστημα [2, 6).

```
print(x)
```

- 2
- 3
- 4
- 5

Η συνάρτηση range() αυξάνει **εξορισμού τηγ ακολουθία κατά 1**. Παρόλαυτα είναι δυνατό να προσδιορίσουμε διαφορετική τιμή αύξησης προσθέτοντας μια τρίτη παράμετρο π.χ. range(2, 30, 3). Το 3 είναι το βήμα αύξησης. Με τη συνάρτηση αυτή η ακολουθία αυξάνεται κατά 3 σε κάθε επανάληψη

```
for x in range(2, 30, 3): #στο x πάνε οι τιμές από 2 έως 30 (ΧΩΡΙΣ ΤΟ 30) με βήμα αύξησης το 3  
    print(x)
```

```
print(x) #εδώ τυπώνεται η τελευταία τιμή του x που είναι 29
```

- 2
- 5
- 8
- 11
- 14
- 17
- 20
- 23
- 26
- 29
- 29

Παράδειγμα 4- Εντολή else στο for

Η εντολή else σε μια for είναι ένα μπλοκ κώδικα που **εκτελείται όταν τερματιστεί το for**. Στο επόμενο παράδειγμα τυπώνονται όλοι οι αριθμοί από 0 μέχρι 5 και ένα μήνυμα όταν τελειώσει το for.

```
for x in range(6): #αφού στη συνάρτηση range δεν προσδιορίζεται αρχική τιμή, η εξορισμού αρχική τιμή στο x είναι το 0 μέχρι και το 5  
    print(x)  
else:  
    print("Finally finished!")
```

Τυπώνονται τα ακόλουθα αποτελέσματα:

- 0
- 1
- 2
- 3
- 4
- 5
- Finally finished!

Παράδειγμα 5- Διπλό for

#Εκτύπωση Προπαίδειας

for i in range(1, 11):

 for j in range(1, 11):

 print(f'{i}x{j}={i*j}')

print() #αλλαγή γραμμής μετά το τέλος κάθε εσωτερικού for

5.8.3 Δομές Επανάληψης- Εντολή while

Στην εντολή while εκτελείται επανάληψη όσο αληθεύει η συνθήκη του while και στο τέλος της συνθήκης μπαίνει :

Παράδειγμα 1

Εισαγωγή 5 τιμών και χαρακτηρισμός τους ως άρτια ή περιττή

i=1

while i<6:

 x=**int**(**input**("Δώσε τιμή: "))

if x%2==0:

print(f" Η τιμή {x} είναι άρτια")

else:

print(f" Η τιμή {x} είναι περιττή")

i+=1

Παράδειγμα 2

Εύρεση ΜΟ από n τυχαίους βαθμούς

sum=0 #αρχικοποίηση αθροιστή

n=**int**(**input**("Δώσε Πλήθος Βαθμών: "))

i=1

while i<=n: #while i<n+1

 x=**float**(**input**("Δώσε Βαθμό: "))

 sum+=x #sum=sum+x. Το sum είναι αθροιστής διότι αθροίζει τιμές

i+=1

print(f" Ο Μέσος Όρος είναι {sum/n}")

Παράδειγμα 3

Επανάληψη που εκτελείται άγνωστο αριθμό φορών μέχρι να δοθεί αρνητικός

while True:

 s=**int**(**input**("Δώσε τιμή: "))

if s<0:

break

if s>0:

print(s)

if s==0:

continue

print("End")

Παράδειγμα 4

Να τυπωθούν οι όροι της ακολουθίας Fibonacci. Η ακολουθία Fibonacci είναι μια ακολουθία όρων όπου ο καθένας προκύπτει ως άθροισμα των 2 προηγούμενων του όρων $f(n)=f(n-1)+f(n-2)$. Οι 2 αρχικοί όροι είναι το 0 και το 1. Οι επόμενοι όροι 1(1+0), 2(1+1) κ.λ.π.

$f(0)=0$, $f(1)=1$, $f(2)=1$, $f(3)=2$, $f(4)=3$

#Ακολουθία Fibonacci

a, b=0, 1 #στο a καταχωρείται το 0 και στο b το 1 ουσιαστικά σε ένα βήμα γίνεται μια διπλή αρχικοποίηση και οι τιμές καταχωρούνται κατ' αντιστοιχία

while $b < 10$: #στην επικεφαλίδα του while μπαίνει : και εσοχή για το σώμα του

print(b)

a, b= b, a+b #οι καταχωρίσεις κατά σειρά είναι πρώτα $a=b$ και μετά $b=b+a$

- 1
- 1
- 2
- 3
- 5
- 8

Επεξήγηση αποτελεσμάτων

	a	b	Εκτύπωση
Αρχικά	0	1	1
1 ^η επανάληψη	1	1	1
2 ^η επανάληψη	1	2	2
3 ^η επανάληψη	2	3	3 κ.λ.π.

5.9 Δομές Δεδομένων στην Python

5.9.1 Λίστες (Lists)

Μια λίστα είναι ένα σύνολο στοιχείων **ταξινομημένο και τροποποιήσιμο**. Οι λίστες συμβολίζονται με []. Η λίστα ομαδοποιεί δεδομένα και είναι ένας κατάλογος τιμών που διαχωρίζονται με κόμματα και βρίσκονται όλες μέσα σε []. **Δεν είναι υποχρεωτικό οι τιμές της λίστας να είναι όλες του ίδιου τύπου** π.χ. a=['London','Rome', 1452, 9]. Κάθε μέλος μιας λίστας έχει μια θέση μέσα σε αυτή. Η θέση του 1^{ου} μέλους είναι η 0. π.χ. Επίσης σε μια λίστα μπορεί και να υπάρχουν και ίδιες τιμές σε διαφορετικές θέσεις της.

Παράδειγμα 1-Δημιουργία Λίστας

Δημιουργία και εκτύπωση Λίστας

```
mylist = ["apple", "banana", "cherry"]
print(mylist) #τυπώνεται ['apple', 'banana', 'cherry']
```

Παράδειγμα 2-Εκτύπωση 2^{ου} στοιχείου Λίστας

```
mylist = ["apple", "banana", "cherry"]
print(mylist [1]) #τυπώνεται banana
```

Παράδειγμα 3-Τροποποίηση 2^{ου} στοιχείου Λίστας

```
mylist = ["apple", "banana", "cherry"]
mylist [1] = "blackcurrant" #τροποποιούμε το στοιχείο στη θέση 1 θέτοντας τον τιμή blackcurrant
print(mylist) #τυπώνεται ['apple', 'blackcurrant', 'cherry']
```

Παράδειγμα 4-Εκτύπωση Στοιχείων Λίστας

Τυπώνονται όλα τα στοιχεία της λίστας

```
mylist = ["apple", "banana", "cherry"]
for x in mylist:
    print(x) #τυπώνεται η λίστα
```

```
apple
banana
cherry
```

```
mylist = ["apple", "banana", "cherry"]
for x in mylist:
    print(x, end=' ') #τυπώνεται η λίστα με ένα κενό ανάμεσα στα στοιχεία της
apple banana cherry
```

Παράδειγμα 5-Εκτύπωση Μήκους Λίστας

Υπολογίζεται το πλήθος των στοιχείων της λίστας.

```
mylist = ["apple", "banana", "cherry"]
print(len(mylist)) # τυπώνεται η τιμή 3
```

Με δεδομένη τη λίστα ['a', 'b', 3, 'd', 1, 'a'] εκτελώντας τη συνάρτηση len() σε αυτή θα πάρουμε τα ακόλουθα:

```
len(a)
```

6

Η `len` είναι μια συνάρτηση που έχει εφαρμογή σε όλα τα αντικείμενα της Python
`k='asdfghjkl'`

`len(k)`

9

Παράδειγμα 6-Προσθήκη νέου στοιχείου στο τέλος της Λίστας

Συνάρτηση `append()`

Προστίθεται ένα νέο στοιχείο στο τέλος της Λίστας

`mylist = ["apple", "banana", "cherry"]`

`mylist.append("orange")`

`print(mylist) #τυπώνεται ['apple', 'banana', 'cherry', 'orange']`

Αν στη λίστα `a=['London','Rome', 1452, 9]` εκτελέσουμε την ακόλουθη εντολή:

`a.append(12)`

`['London','Rome', 1452, 9, 12]`

Συνάρτηση `extend()`

Η συνάρτηση αυτή προσθέτει στο τέλος της λίστας **περισσότερα μέλη στο τέλος της λίστας** (όσα μέλη υπάρχουν μέσα στην παρένθεση).

Αν στη λίστα `a=['London','Rome', 'Paris', 1452, 9, 12]` εκτελέσουμε την εντολή:

`a.extend(['Milano', 1812])`

`['London','Rome', 'Paris', 1452, 9, 12, 'Milano', 1812]`

Παράδειγμα 7-Προσθήκη νέου στοιχείου σε συγκεκριμένη θέση της Λίστας

Προστίθεται ένα νέο στοιχείο σε **συγκεκριμένη θέση της λίστας** και τα υπόλοιπα στοιχεία μετακινούνται αυτομάτως προς τα δεξιά.

`mylist = ["apple", "banana", "cherry"]`

`mylist.insert(1, "orange") #τυπώνεται ['apple', 'orange', 'banana', 'cherry']`

`print(mylist)`

Αν στη λίστα `a=['London','Rome', 1452, 9, 12]` εκτελέσουμε την εντολή:

`a.insert(2,'Paris')`

`['London','Rome', 'Paris', 1452, 9, 12]`

Παράδειγμα 8-Διαγραφή συγκεκριμένου στοιχείου Λίστας

Μέθοδος `remove()`

Η μέθοδος `remove()` διαγράφει συγκεκριμένο στοιχείο της λίστας όπως στο επόμενο παράδειγμα:

`mylist = ["apple", "banana", "cherry"]`

`mylist.remove("banana")`

`print(mylist) #τυπώνεται ['apple', 'cherry']`

Η συνάρτηση remove() αυτή αφαιρεί από μια λίστα την **πρώτη εμφάνιση** μιας συγκεκριμένης τιμής

```
a=['a', 1, 'b', 3, 'd', 1, 'a', 7]
```

```
a.remove(1)
```

```
a=['a', 'b', 3, 'd', 1, 'a', 7] #διαγράφει μόνο την 1η εμφάνιση της τιμής 1 από τη λίστα όχι όλες τις εμφανίσεις του 1
```

Εντολή del

Η μέθοδος del διαγράφει εναλλακτικά ένα συγκεκριμένο στοιχείο της λίστας όπως στο επόμενο παράδειγμα:

```
mylist = ["apple", "banana", "cherry"]
```

```
del mylist[0]
```

```
print(thislist) #τυπώνεται ['banana', 'cherry']
```

Παράδειγμα 9-Διαγραφή τελευταίου στοιχείου Λίστας

Μέθοδος pop()

Η μέθοδος pop() αφαιρεί το τελευταίο στοιχείο της λίστας και το επιστρέφει.

```
mylist = ["apple", "banana", "cherry"]
```

```
mylist.pop()
```

```
print(mylist)
```

Έχοντας ως δεδομένη τη λίστα a=['a', 'b', 3, 'd', 1, 'a', 7] και εκτελώντας τη συνάρτηση pop() σε αυτή θα έχουμε:

```
c=a.pop()
```

και αν τυπώσουμε τη λίστα θα πάρουμε το ακόλουθο περιεχόμενο:

```
['a', 'b', 3, 'd', 1, 'a']
```

Παράδειγμα 10-Διαγραφή όλης της Λίστας

Η μέθοδος clear() διαγράφει όλα τα στοιχεία της λίστας:

```
mylist = ["apple", "banana", "cherry"]
```

```
mylist.clear()
```

```
print(mylist) #τυπώνεται []
```

Παράδειγμα 11-Εντοπισμός στοιχείου Λίστας

Η συνάρτηση index() βρίσκει το πρώτο μέλος τη λίστας που είναι ίδιο με αυτό που έχουμε βάλει μέσα στην παρένθεση και επιστρέφει τη θέση του μέσα στη λίστα

```
['a', 'b', 3, 'd', 1, 'a']
```

```
a.index(3) #εντοπίζει την 1η θέση της λίστας που περιέχει το στοιχείο 3
```

```
2
```

```
a=['a','b',3,'d',3,'e',3]
```

```
print(a.index(3)) # εντοπίζει μόνο την 1η θέση της λίστας που περιέχει το στοιχείο 3
```

```
2
```

Παράδειγμα 12-Άθροισμα στοιχείων Λίστας

Μπορούμε να υπολογίσουμε το άθροισμα των μελών μιας λίστας (αν είναι αριθμοί) με τη συνάρτηση sum() όπως στο επόμενο παράδειγμα:

```
a=[1, 2, 3, 4, 5]
```

```
sum(a)
```

```
15
```

Παράδειγμα 13-Συνάρτηση range

Η συνάρτηση range() επιστρέφει μια λίστα αριθμών **μέχρι τον προηγούμενο αριθμό που γράφουμε στην παρένθεση** και η συνάρτηση list δημιουργεί μια λίστα

list(range(8)) #δημιουργία νέας λίστας με τις τιμές 0 μέχρι και 7 (χωρίς το 8)

[0, 1, 2, 3, 4, 5, 6, 7]

Ο αριθμός μέσα στην παρένθεση σηματοδοτεί την τελευταία τιμή η οποία δεν συμπεριλαμβάνεται στη λίστα

list(range(5, 11)) #δημιουργία νέας λίστας με τις τιμές 5 μέχρι και το 10 (χωρίς το 11)

[5, 6, 7, 8, 9, 10]

Αν δώσουμε 3 αριθμούς στην παρένθεση, ο 1^{ος} δηλώνει τον πρώτο αριθμό της λίστας, ο 2^{ος} τον τελικό αριθμό που δεν θα περιλαμβάνει η λίστα και ο 3^{ος} αριθμός το βήμα αύξησης των τιμών της λίστας

list(range(0, 30, 5))

[0, 5, 10, 15, 20, 25]

list(range(100, 0, -10))

[100, 90, 80, 70, 60, 50, 40, 30, 20, 10]

Όπως φαίνεται από το παραπάνω παράδειγμα η συνάρτηση range() μπορεί να δουλέψει και κατά φθίνουσα σειρά αν δώσουμε τον τρίτο αριθμό αρνητικό

Παράδειγμα 14-Τελεστής in

li=[1, 3, 5, 7, 9]

for i in li:#σε κάθε επανάληψη στη μεταβλητή i μπαίνει μια τιμή από τη λίστα li με τον τελεστή in. Επίσης είναι απαραίτητη η χρήση του τελεστή : στην επικεφαλίδα της for και η εσοχή για το σώμα εντολών της for

print(i)

Τυπώνονται τα ακόλουθα αποτελέσματα:

1
3
5
7
9

Παράδειγμα 15-Τελεστής in και index

mylist=[1, 3, 5, 7, 9]

for i in mylist:

print(i, mylist.index(i)) #Η συνάρτηση index επιστρέφει τη θέση του στοιχείου i στη λίστα

Τυπώνονται τα ακόλουθα αποτελέσματα:

1 0
3 1
5 2

7 3

9 4

Παράδειγμα 16-Τελεστής **in** και **len**

onoma=['John', 'Roger', 'Natalie', 'Tamara']

for a in onoma:

print(a, len(a)) #σε κάθε επανάληψη μπαίνει στο a μια τιμή (όνομα) της λίστας και τυπώνεται η τιμή αυτή (το όνομα αυτό) και το πλήθος χαρακτήρων του ονόματος αυτού

Τυπώνονται τα ακόλουθα αποτελέσματα:

John 4

Roger 5

Natalie 7

Tamara 6

Παράδειγμα 17-Τελεστής **in**

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x) #σε κάθε επανάληψη μπαίνει στο x μια τιμή (στοιχείο) της λίστας και τυπώνεται το στοιχείο αυτό

Τυπώνονται τα ακόλουθα αποτελέσματα:

apple

banana

cherry

Παράδειγμα 18-Τελεστής **in**

for x in "banana":

print(x) #σε κάθε επανάληψη μπαίνει στο x μια τιμή (ένας χαρακτήρας) του αλφαριθμητικού "banana" και τυπώνεται

Τυπώνονται τα ακόλουθα αποτελέσματα:

b

a

n

a

n

a

Παράδειγμα 19-Τελεστής in και break

fruits = ["apple", "banana", "cherry"]

for x in fruits:

 print(x)

 if x == "banana":

 break #με την εντολή αυτή η for τερματίζει στη 2^η επανάληψη της

Τυπώνονται τα ακόλουθα αποτελέσματα:

apple

banana

Παράδειγμα 20-Τελεστής in και continue

fruits = ["apple", "banana", "cherry"]

for x in fruits:

 if x == "banana":

 continue με την εντολή αυτή η for συνεχίζει στην επόμενη επανάληψη της αγνοώντας τις εντολές που βρίσκονται στο σώμα του for

 print(x)

Τυπώνονται τα ακόλουθα αποτελέσματα:

apple

cherry

Παράδειγμα 21-Τελεστής in με διπλό for

adj = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]

for x in adj: #για κάθε τιμή του x από τη λίστα adj εκτελείται πλήρως η εσωτερική επανάληψη του y (δηλ. το y λαμβάνει για κάθε x όλες τις τιμές από τη λίστα fruits

 for y in fruits:

 print(x, y)

Τυπώνονται τα ακόλουθα αποτελέσματα:

red apple

red banana

red cherry

big apple

big banana

big cherry

tasty apple

tasty banana

tasty cherry

Παράδειγμα 21-Εκτύπωση Λίστας ως δισδιάστατος πίνακας

pinax = [[1, 2, 3], [4, 5, 6]]

for row in pinax: #στο row μπαίνει στην 1^η επανάληψη το στοιχείο [1, 2, 3] και στη 2^η επανάληψη το στοιχείο [4, 5, 6]

 for item in row: #στο item μπαίνουν οι τιμές 1, 2, 3 όταν το row είναι το [1, 2, 3] και αντίστοιχα οι τιμές 4, 5, 6 όταν το row είναι το [4, 5, 6]

 print(item, end=" ")

 print()

Τυπώνονται τα ακόλουθα αποτελέσματα:

1 2 3

4 5 6

Παράδειγμα 22-Εισαγωγή Τιμών σε Λίστα Από πληκτρολόγιο

Εισαγωγή το πολύ 5 αριθμών από το χρήστη. Αν ο χρήστης δώσει x σταματά την εισαγωγή

count = 0

li = []

while count < 5:

 n = input('δώσε αριθμό:')

 if n == 'x':

 break

 if n.isdigit():

 count += 1

 li.append(int(n))

έξοδος από τον βρόχο

print('Η λίστα είναι:', li)

Παράδειγμα 23-Εκτύπωση Λίστας

mylist1 = ["Python", "Programming", 100] # Η λίστα είναι μια συλλογή στοιχείων ίδιου ή διαφορετικού τύπου

for x in mylist1: # εκτελείται μια επανάληψη όσα και τα στοιχεία της λίστας. Σε κάθε επανάληψη ενα στοιχείο της λίστας πηγαίνει στο x, δηλ. στην 1η επανάληψη μπαίνει η λέξη Python στο x, στη 2η επανάληψη μπαίνει η λέξη Programming στο και στη 3η μπαίνει η τιμή 100 στο x

print(x) # κάθε στοιχείο τυπώνεται σε ξεχωριστή γραμμή

for x in mylist1: #εκτελείται μια επανάληψη όσα και τα στοιχεία της λίστας. Σε κάθε επανάληψη ενα στοιχείο της λίστας πηγαίνει στο x

 print(x,end=' ') # κάθε στοιχείο τυπώνεται το ένα δίπλα στο άλλο. To end=' ' καταργεί την αλλαγή γραμμής. Τυπάνει το ενα στοιχείο δίπλα στο άλλο

mylist2 = [] # ορίζουμε μια κενή λίστα

for i in range(1, 6): # Εκτελείται επανάληψη για i από 1 μέχρι και 5

```
x = input("\n Δώσε στοιχείο λίστας: ") # εισάγουμε τιμές στη λίστα αυτή από το πληκτρολόγιο.
```

Παράδειγμα 24-Προσθήκη Τιμών σε Λίστα

```
mylist2.append(x) #προσθηκη κάθε στοιχείου στο τέλος της λίστας με την έτοιμη συνάρτηση append  
print("\n Τιμές 2ης Λίστας")  
for i in range(len(mylist2)):  
    print(mylist2[i], end=' ')
```

mylist2.insert(1, "Element") # Η έτοιμη συνάρτηση insert εισάγει στη 2η θέση της λίστας το αλφαριθμητικό Element. Οι λίστες αριθμούνται πάντα από τη θέση 0. Η διαφορά της insert από την append είναι ότι η insert εισάγει στοιχείο σε συγκεκριμένη θέση ενα η append στο τέλος της λίστας

```
print("\n Τιμές 2ης Λίστας μετά την προσθήκη του στοιχείου Element")  
for i in range(len(mylist2)): # Η έτοιμη συνάρτηση len υπολογίζει το πλήθος στοιχείων μιας λίστας  
    print(mylist2[i], end=' ')
```

```
mylist2[0] = 20 # Αλλάζουμε το πρώτο στοιχείο της λίστας σε 20  
mylist2.extend(["new", "language", 3]) # Με την extend επεκτείνουμε τη λίστα προσθέτοντας μια νέα λίστα στο τέλος της
```

```
mylist3 = mylist1 + mylist2 # Δημιουργούμε μια νέα λίστα όπου με τον τελεστή + συνενώνουμε δυο λίστες σε αυτή  
print("\n", mylist3)
```

Παράδειγμα 25-Διαγραφή Τιμών από Λίστα

```
mylist2.remove("Element") #Διαγράφεται η λέξη Element (μόνο στην 1η εμφάνιση της αν υπάρχει πολλές φορές)  
print("\n Τιμές 2ης Λίστας μετά τη διαγραφή του στοιχείου Element")  
for i in range(len(mylist2)): # Η έτοιμη συνάρτηση len υπολογίζει το πλήθος στοιχείων μιας λίστας  
    print(mylist2[i], end=' ')
```

```
mylist2.pop() # Διαγράφεται το τελευταίο στοιχείο της λίστας (όποιο και αν είναι)
```

```
print("\n Τιμές 2ης Λίστας μετά τη διαγραφή του τελευταίου στοιχείου Element")
```

```
for i in range(len(mylist2)):
```

```
    print(mylist2[i], end=' ')
```

```
del mylist3[len(mylist3) - 1] # Διαγράφουμε το τελευταίο στοιχείο της λίστας
```

```
print("\n Τιμές 2ης Λίστας μετά τη διαγραφή του τελευταίου στοιχείου Element")
```

```
for i in range(len(mylist2)):
```

```
    print(mylist3[i], end=' ')
```

Παράδειγμα 26-Ταξινόμηση Λίστα

mylist1.sort() # *Η συνάρτηση sort() ταξινομεί τα στοιχεία μια λίστας αλλά για να δουλέψει σωστά πρέπει να έχουμε στοιχεία με ίδιο τύπο στη λίστα π.χ. όλα string, όλα int*

```
print("\n Τιμές ταξινομημένης 1ης Λίστας")
```

```
for i in range(len(mylist1)):  
    print(mylist1[i], end=' ')
```

Παράδειγμα 27-Ταξινόμηση Λίστας

```
mylist3.reverse() # Η λίστα 3 αντιστρέφεται δηλ γράφονται τα στοιχεία της απο δεξιά προς τα αριστερά  
print("\nΤιμές Αντεστραμμένης 3ης Λίστας")  
for i in range(len(mylist3)):  
    print(mylist3[i], end=' ')
```

Παράδειγμα 28-Αντιγραφή Λίστας

```
mylist4 = mylist3.copy() # Η λίστα mylist3 αντιγράφεται στη λίστα 4  
print("\nΤιμές 4ης Λίστας")  
for i in range(len(mylist4)):  
    print(mylist4[i], end=' ')
```

Παράδειγμα 27-Max, Min και Άθροισμα Λίστας

```
print(f" Το μέγιστο στοιχείο της λίστα: {max(list)}") #Η συνάρτηση υπολογίζει το μέγιστο στοιχείο της λίστας  
print(f" Το ελάχιστο στοιχείο της λίστας: {min(list)}") #Η συνάρτηση υπολογίζει το ελάχιστο στοιχείο της λίστας  
print(f" Το άθροισμα της λίστας: {sum(list)}") #Η συνάρτηση υπολογίζει το άθροισμα της λίστας
```

5.9.2 Πλειάδες (Tuples)

Μια πλειάδα είναι ένα **ταξινομημένο σύνολο στοιχείων και ΜΗ τροποποιήσιμο**. Οι πλειάδες συμβολίζονται με () .

Παράδειγμα 1-Δημιουργία Πλειάδας

Δημιουργία και εκτύπωση Πλειάδας

```
mytuple = ("apple", "banana", "cherry")
```

```
print(mytuple) #τυπώνεται ('apple', 'banana', 'cherry')
```

Ενναλακτικά μια πλειάδα δημιουργείται καλώντας το δημιουργό tuple() όπως δείχνει το επόμενο παράδειγμα

```
mytuple =tuple(("apple", "banana", "cherry"))
```

```
print(mytuple) #τυπώνεται ('apple', 'banana', 'cherry')
```

Παράδειγμα 2- Δημιουργία Πλειάδας και Εκτύπωση 2^{ου} στοιχείου Πλειάδας

```
mytuple = ("apple", "banana", "cherry")
```

```
print(mytuple [1]) #τυπώνεται banana
```

Παράδειγμα 3-Εκτύπωση Στοιχείων Πλειάδας

Τυπώνονται όλα τα στοιχεία της πλειάδας (tuple)

```
mytuple =("apple", "banana", "cherry")
```

```
for x in mytuple: #σε κάθε επανάληψη ένα στοιχείο της πλειάδας μπαίνει στο x
```

```
print(x) #τυπώνεται η λίστα apple banana cherry
```

Παράδειγμα 4-Εκτύπωση Μήκους Πλειάδας

Τυπώνεται το πλήθος των στοιχείων της πλειάδας

```
mytuple = ("apple", "banana", "cherry")
```

```
print(len(mytuple)) #τυπώνεται η τιμή 3 που είναι το πλήθος στοιχείων της πλειάδας
```

ΠΡΟΣΟΧΗ: ΑΠΟ ΤΗ ΣΤΙΓΜΗ ΠΟΥ ΔΗΜΙΟΥΡΓΕΙΤΑΙ ΜΙΑ ΠΛΕΙΑΔΑ ΔΕΝ ΤΡΟΠΟΠΟΙΕΙΤΑΙ.

Παράδειγμα 5-Προσθήκη στοιχείου στην Πλειάδα

Ο ΑΚΟΛΟΥΘΟΣ ΚΩΔΙΚΑΣ ΕΙΝΑΙ ΛΑΘΟΣ ΓΙΑΤΙ ΤΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΠΛΕΙΑΔΑΣ ΔΕΝ ΤΡΟΠΟΠΟΙΟΥΝΤΑΙ.

```
mytuple = ("apple", "banana", "cherry")
```

```
mytuple [3]="orange" #Αυτό προκαλεί σφάλμα διότι η πλειάδα δεν είναι επεξεργάσιμη (τροποποιήσιμη)
```

```
print(mytuple)
```

Παράδειγμα 6-Διαγραφή Πλειάδας

Ο ΑΚΟΛΟΥΘΟΣ ΚΩΔΙΚΑΣ ΕΙΝΑΙ ΛΑΘΟΣ ΓΙΑΤΙ ΜΕΜΟΝΩΜΕΝΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΠΛΕΙΑΔΑΣ ΔΕΝ ΔΙΑΓΡΑΦΟΝΤΑΙ

```
thistuple = ("apple", "banana", "cherry") → Δημιουργία Πλειάδας
```

```
del thistuple[3] #Η εντολή αυτή προκαλεί λάθος διότι δεν μπορούμε να διαγράψουμε συγκεκριμένο στοιχείο της πλειάδας
```

ΜΠΟΡΟΥΜΕ ΟΜΩΣ ΝΑ ΔΙΑΓΡΑΨΟΥΜΕ ΟΛΗ ΤΗΝ ΠΛΕΙΑΔΑ

```
del thistuple #Η εντολή αυτή είναι σωστή και διαγράφει όλη την πλειάδα
```

```
print(thistuple) #Η εντολή αυτή προκαλεί λάθος διότι η πλειάδα δεν υπάρχει πλέον (the tuple no longer exists)
```

5.9.3 Σύνολα (sets)

Ένα σύνολο είναι μια συλλογή στοιχείων χωρίς ταξινόμηση και δεικτοδότηση. Τα σύνολα συμβολίζονται με {}. Σε ένα σύνολο μπορούμε να προσθέσουμε νέα στοιχεία, μπορούμε και να διαγράψουμε αλλά ΔΕΝ ΜΠΟΡΟΥΜΕ ΝΑ ΤΡΟΠΟΠΟΙΗΣΟΥΜΕ ΥΠΑΡΧΟΝΤΑ ΣΤΟΙΧΕΙΑ ΤΟΥ ΣΥΝΟΛΟΥ.

Παράδειγμα 1-Δημιουργία και Εκτύπωση Συνόλου

Δημιουργία και εκτύπωση Συνόλου. Επειδή τα στοιχεία ενός συνόλου δεν έχουν δεικτοδότηση δεν βρίσκονται σε συγκεκριμένες θέσεις και κάθε φορά που εκτυπώνουμε ένα σύνολο τα στοιχεία του εμφανίζονται σε διαφορετική σειρά

```
myset = {"apple", "banana", "cherry"}
```

```
print(myset)
```

{'cherry', 'apple', 'banana'}

και στην επόμενη εκτέλεση της print δίνει:

{'apple', 'banana', 'cherry'}

και στην επόμενη εκτέλεση της print δίνει:

{'banana', 'apple', 'cherry'}

Παράδειγμα 2- Δημιουργία και Εκτύπωση Συνόλου

Τυπώνονται όλα τα στοιχεία του συνόλου

```
myset = {"apple", "banana", "cherry"}
```

for x in myset:

```
    print(x) #τυπώνεται το σύνολο
```

apple

banana

cherry

Παράδειγμα 3-Εκτύπωση Πλήθους στοιχείων Συνόλου

Τυπώνεται το πλήθος των στοιχείων του συνόλου

```
myset = {"apple", "banana", "cherry"}
```

```
print(len(myset)) #τυπώνεται η τιμή 3
```

Παράδειγμα 4-Προσθήκη νέου στοιχείου και νέων στοιχείων στο σύνολο

Για να προσθέσουμε ένα στοιχείο στο σύνολο χρησιμοποιούμε τη μέθοδο add(). Για να προσθέσουμε πολλά στοιχεία στο σύνολο χρησιμοποιούμε τη μέθοδο update().

Προσθήκη ενός στοιχείου στο σύνολο

```
myset = {"apple", "banana", "cherry"}
```

```
myset.add("orange")
```

```
print(myset) #Τυπώνεται {'apple', 'banana', 'orange', 'cherry'}
```

Προσθήκη πολλών στοιχείων στο σύνολο

```
myset = {"apple", "banana", "cherry"}
```

```
myset.update(["orange", "mango", "grapes"])
```

```
print(myset) #Τυπώνεται {'orange', 'mango', 'cherry', 'grapes', 'banana', 'apple'}
```

Παράδειγμα 5-Διαγραφή συγκεκριμένου στοιχείου Συνόλου

Η μέθοδος remove() διαγράφει ένα συγκεκριμένο στοιχείο του συνόλου

```
myset = {"apple", "banana", "cherry"}  
myset.remove("banana")  
print(myset) #Τυπώνεται ['apple', 'cherry']
```

Παράδειγμα 6-Διαγραφή τελευταίου στοιχείου Συνόλου

Η μέθοδος pop() διαγράφει το **τελευταίο στοιχείο ενός συνόλου**.

```
myset = {"apple", "banana", "cherry"}  
x = myset.pop()  
print(x) #Τυπώνεται cherry  
print(myset) #Τυπώνεται {'banana', 'apple'}
```

Παράδειγμα 7-Διαγραφή όλου του Συνόλου

Η μέθοδος clear() διαγράφει όλο το σύνολο:

```
myset = {"apple", "banana", "cherry"}  
myset.clear()  
print(myset) #Τυπώνεται set()
```

5.9.4 Λεξικά (Dictionaries)

Ένα λεξικό (dictionary) είναι μια συλλογή στοιχείων μη ταξινομημένη, τροποποιήσιμη και δεικτοδοτημένη. Τα λεξικά στην Python συμβολίζονται με άγκιστρα και κάθε τιμή του λεξικού είναι ένα ζεύγος κλειδί: τιμή ("key":value)

Παράδειγμα 1-Δημιουργία Λεξικού

Δημιουργία και εκτύπωση Λεξικού

```
mydict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
print(mydict) #Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Παράδειγμα 2-Δημιουργία Λεξικού με κλήση του δημιουργού dict()

Η αντιγραφή λεξικού γίνεται εναλλακτικά με το δημιουργό dict() όπως φαίνεται στο επόμενο παράδειγμα:

mydict =**dict**(brand="Ford", model="Mustang", year=1964) #εδώ δημιουργούμε ένα νέο λεξικό με το δημιουργό dict() και το καταχωρούμε στη μεταβλητή thisdict

```
print(mydict) #Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Παράδειγμα 3-Εκτύπωση Τιμών Λεξικού

Μπορούμε να προσπελάσουμε τα στοιχεία ενός λεξικού γράφοντας όνομα λεξικού[κλειδί] όπως στο επόμενο παράδειγμα όπου θέλουμε να πάρουμε το κλειδί model:

```
x =mydict["model"]  
  
print(x) #Τυπώνεται Mustang
```

Παράδειγμα 4-Εκτύπωση Τιμών Λεξικού με τη μέθοδο get()

Εναλλακτικά μπορούμε να χρησιμοποιήσουμε τη μέθοδο get() για να πάρουμε το ίδιο αποτέλεσμα όπως στο επόμενο παράδειγμα:

```
x =mydict.get("model")  
  
print(x) #Τυπώνεται Mustang
```

Παράδειγμα 5-Αλλαγή Τιμής Λεξικού

Μπορούμε να αλλάξουμε την τιμή ενός στοιχείου μέσω της τιμής του κλειδιού του. Για παράδειγμα θέλουμε να αλλάξουμε το year σε 2018.

```
mydict={  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
mydict ["year"]=2018  
  
print(mydict) # Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Παράδειγμα 6-Εκτύπωση Κλειδιών και Τιμών Λεξικού

Όταν κάνουμε προσπέλαση στα στοιχεία ενός λεξικού οι επιστρεφόμενες τιμές είναι τα κλειδιά του λεξικού. Όμως υπάρχουν και μέθοδοι για να επιστρέψουν τις τιμές των κλειδιών.

Παράδειγμα 6.1 Εκτύπωση Κλειδιών Λεξικού

```
mydict ={  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
for x in mydict:  
    print(x) #τυπώνονται μόνο τα κλειδιά (keys)
```

brand
model
year

Παράδειγμα 6.2 Εκτύπωση Τιμών Λεξικού

```
mydict ={  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
for x in mydict: #σε κάθε επανάληψη καταχωρούνται στο x τα κλειδιά του λεξικού δηλ. κατά σειρά οι λέξεις brand, model, year  
    print(mydict[x]) #τυπώνονται τυπώνονται μόνο οι τιμές (values)
```

Ford
Mustang
1964

Μπορούμε εναλλακτικά να γρησμοποιήσουμε τη συνάρτηση values() για να πάρουμε όλες τις τιμές του λεξικού

```
for x in mydict.values(): #στο x καταχωρείται αυτόματα με τη συνάρτηση values() η τιμή κάθε κλειδιού  
    print(x) #τυπώνονται μόνο οι τιμές (values)
```

Ford
Mustang
1964

Παράδειγμα 6.3 Εκτύπωση Κλειδιών και Τιμών

Αν θέλουμε να εκτυπώσουμε και τα κλειδιά και τις τιμές τους χρησιμοποιούμε τη συνάρτηση `items()` όπως στο επόμενο παράδειγμα:

```
mydict = {
```

```
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

for x, y **in** mydict.items(): #στο x καταχωρείται σε κάθε επανάληψη ένα κλειδί (key) και στο y καταχωρείται η αντίστοιχη τιμή του (value)

```
    print(x, y) #τυπώνονται ταυτόχρονα και τα κλειδιά και οι τιμές
```

brand Ford

model Mustang

year 1964

Παράδειγμα 7-Έλεγχος Ύπαρξης Κλειδιού στο Λεξικό

Για να ελέγξουμε αν ένα κλειδί υπάρχει στο λεξικό χρησιμοποιούμε την εντολή `in` όπως στο επόμενο παράδειγμα:

```
mydict = {
```

```
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

if "model" **in** mydict: #έλεγχος αν το κλειδί model υπάρχει στο λεξικό mydict

```
    print("Yes, 'model' is one of the keys in the thisdict dictionary") #Τυπώνεται Yes, 'model' is one of the keys in the thisdict dictionary
```

Παράδειγμα 8-Εκτύπωση Πλήθους στοιχείων Λεξικού

Τυπώνεται το πλήθος των στοιχείων (ζευγών κλειδιών: τιμών) του Λεξικού.

```
print(len(mydict)) #Τυπώνεται 3
```

Παράδειγμα 9-Προσθήκη νέου στοιχείου στο Λεξικό

Η προσθήκη ενός νέου στοιχείου στο Λεξικό γίνεται χρησιμοποιώντας ένα νέο κλειδί και εκχωρώντας του τιμή όπως στο επόμενο παράδειγμα:

```
mydict = {
```

```
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
mydict["color"] = "red"
```

```
print(mydict) #Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Παράδειγμα 10-Αφαίρεση στοιχείου από το Λεξικό

Υπάρχουν πολλοί τρόποι για να αφαιρέσουμε στοιχεία από λεξικό.

Μέθοδος `pop()`

Η μέθοδος `pop()` διαγράφει το στοιχείο με το προσδιοριζόμενο όνομα κλειδιού και την τιμή του

```
mydict = {
```

```
    "brand": "Ford",
```

```
    "model": "Mustang",
```

```
    "year": 1964
```

```
}
```

```
mydict.pop("model")
```

```
print(thisdict) #Τυπώνεται {'brand': 'Ford', 'year': 1964}
```

Μέθοδος `popitem()`

Η μέθοδος `popitem()` διαγράφει το τελευταίο στοιχείο δηλ. το τελευταίο κλειδί και την τιμή του που έχουν εισαχθεί στο λεξικό.

```
mydict = {
```

```
    "brand": "Ford",
```

```
    "model": "Mustang",
```

```
    "year": 1964
```

```
}
```

```
mydict.popitem()
```

```
print(mydict) #Τυπώνεται {'brand': 'Ford', 'model': 'Mustang'}
```

Η μέθοδος `del()` διαγράφει το στοιχείο με το προσδιοριζόμενο κλειδί και την τιμή του όπως στο επόμενο παράδειγμα:

```
mydict = {
```

```
    "brand": "Ford",
```

```
    "model": "Mustang",
```

```
    "year": 1964
```

```
}
```

```
del mydict ["model"]
```

```
print(thisdict) #Τυπώνεται {'brand': 'Ford', 'year': 1964}
```

Παράδειγμα 11-Διαγραφή Λεξικού

Μέθοδος `del()`

Η μέθοδος `del()` διαγράφει ολόκληρο το λεξικό όπως φαίνεται στο επόμενο παράδειγμα:

```
mydict = {
```

```
    "brand": "Ford",
```

```
    "model": "Mustang",
```

```
    "year": 1964
```

```
}
```

```
del mydict
```

```
print(mydict) #Η εντολή αυτή θα προκαλέσει σφάλμα για το λεξικό με όνομα "thisdict" δεν υπάρχει πλέον.
```

Μέθοδος clear()

Η μέθοδος clear() διαγράφει όλο το περιεχόμενο του λεξικού και αυτό εξακολουθεί να υπάρχει ως κενό λεξικό όπως φαίνεται στο επόμενο παράδειγμα:

```
mydict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
mydict.clear()
```

```
print(thisdict) #Τυπώνεται {}. Αυτή είναι η διαφορά με την del του προηγούμενου παραδείγματος
```

Παράδειγμα 12-Αντιγραφή Λεξικού

Μέθοδος copy()

Η αντιγραφή λεξικού γίνεται με τη μέθοδο copy() όπως στο επόμενο παράδειγμα:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
Mydict=thisdict.copy() #δημιουργείται ένα νέο λεξικό με όνομα mydict και περιεχόμενο αντό του thisdict
```

```
print(mydict) # Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Μέθοδος dict()

Η αντιγραφή λεξικού γίνεται εναλλακτικά και με τη μέθοδο dict() όπως στο επόμενο παράδειγμα:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
mydict = dict(thisdict) #δημιουργείται ένα νέο λεξικό με όνομα mydict και περιεχόμενο αντό του thisdict
```

```
print(mydict) #Τυπώνεται {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

5.10 Συναρτήσεις (Functions)

Στην Python μια συνάρτηση δηλώνεται με την εντολή **def**: όνομα_συνάρτησης (**προαιρετικά ορίσματα**)

Στις παρενθέσεις βάζουμε προαιρετικά ορίσματα μέσω των οποίων η συνάρτηση λαμβάνει δεδομένα αλλά μπορεί και να παραλείπονται εφόσον η συνάρτηση δεν λαμβάνει δεδομένα. Οι εντολές που αποτελούν το σώμα της συνάρτησης μπαίνουν σε εσοχή (tab).

Δήλωση-Ορισμός Συνάρτησης

```
def my_function():
```

```
    print("Hello from a function")
```

Κλήση συνάρτησης

```
my_function() #έτσι γίνεται η κλήση της συνάρτησης με όνομα my_function και τυπώνεται το μήνυμα Hello from a function
```

Παράδειγμα 1: Υπολογισμός Μέσος Όρου τιμών με κλήση συνάρτησης

```
def mesos_oros(num1, num2, num3): #Λαμβάνει ως είσοδο 3 int ορίσματα και επιστρέφει float αποτέλεσμα  
    mo=(num1+num2+num3)/3  
    return mo
```

num1, num2, num3= **int(input("Δώσε 1η τιμή: ")), int(input("Δώσε 2η τιμή: ")), int(input("Δώσε 3η τιμή: "))**

ans=mesos_oros(num1, num2, num3) #μια συνάρτηση που επιστρέφει αποτέλεσμα καλείται είτε σε εντολή εκχώρησης είτε καλείται απευθείας μέσα στην εντολή print. Εδώ το αποτέλεσμα της συνάρτησης επιστρέφεται στη μεταβλητή ans

```
print(f" Ο ΜΟ των {num1}, {num2}, {num3} είναι {ans} ")
```

#Υπολογισμός ΜΟ από 3 τιμές μέσω συνάρτησης

```
def mesos_oros(num1, num2, num3): #Λαμβάνει ως είσοδο 3 int ορίσματα και επιστρέφει float αποτέλεσμα  
    return (num1+num2+num3)/3 #επιστρέφει αποτέλεσμα που το υπολογίζει κατευθείαν σε μια εντολή
```

num1, num2, num3= **int(input("Δώσε 1η τιμή: ")), int(input("Δώσε 2η τιμή: ")), int(input("Δώσε 3η τιμή: "))**

ans=mesos_oros(num1, num2, num3) #μια συνάρτηση που επιστρέφει αποτέλεσμα καλείται είτε σε εντολή εκχώρησης είτε καλείται απευθείας μέσα στην εντολή print. Εδώ το αποτέλεσμα της συνάρτησης επιστρέφεται στη μεταβλητή ans

```
print(f" Ο ΜΟ των {num1}, {num2}, {num3} είναι {ans} ")
```

Παράδειγμα 1 με έλεγχο πρότων αριθμών

Έλεγχος των αριθμών από 1..n για το ποιοι είναι πρώτοι και υπολογισμός πλήθους πρώτων

```
def is_prime(n): #λαμβάνει μια τιμή, ελέγχει αν είναι πρώτος ή όχι και επιστρέφει αντίστοιχα true ή false  
    d=0 #αρχικοποίηση μετρητή διαιρετών κάθε αριθμού
```

```
    for k in range (1, n+1): #με αυτό το for υπολογίζουμε το πλήθος διαιρετών του αριθμού n κάνοντας επανάληψη από το 1 μέχρι το
```

n

```
        if n%k==0: #Αν ισχύει η συνθήκη τότε το k είναι διαιρέτης του n και αφήνει υπόλοιπο 0  
            d+=1 #αυξάνουμε το μετρητή διαιρετών του αριθμού n
```

```
    if d==2 and n!=1: #αν ο αριθμός n έχει ακριβώς 2 διαιρέτες και ΔΕΝ είναι το 1 τότε είναι πρώτος
```

```
    return True
```

```
else:
```

```
    return False
```

```

#Main
n=int(input("Δώσε τελικό όρο: "))
p=0 #αρχικοποίηση μετρητή πρώτων αριθμών

for i in range (1, n+1): #for (i=1;i<n+1;i++)
    if (is_prime(i)==True): #για κάθε τιμή από 1 μέχρι n καλείται η συνάρτηση is_prime(i) και ελέγχει να το I είναι πρώτος
        print(f" Ο αριθμός {i} είναι πρώτος")
        p+=1 #αύξηση μετρητή πρώτων αριθμών κατά 1
    else:
        print(f" Ο αριθμός {i} ΔΕΝ είναι πρώτος")

print(f"\n Το Πλήθος πρώτων αριθμών από 1 έως {n} είναι {p}")

```

Παράδειγμα 2 με εναλλαγή τιμών

Πρόγραμμα που διαβάζει 2 τιμές, τις μεταβιβάζει σε συνάρτηση η οποία τις εναλλάσσει (αντιμεταθέτει) και οι τιμές τυπώνονται στο main εναλλαγμένες. **ΛΑΘΟΣ ΠΡΟΓΡΑΜΜΑ ΠΟΥ ΔΕΝ ΚΑΝΕΙ ΕΝΑΛΛΑΓΗ** διότι η εναλλαγή που γίνεται εντός της συνάρτησης δεν επιστρέφεται στο main

```

def swap(x, y):#Η εναλλαγή γίνεται σε 2 τοπικές μεταβλητές x, y της συνάρτησης swap και δεν επιστρέφουν στο main
    temp=x
    x=y
    y=temp

```

x, y=float(input("Δώσε 1η τιμή: ")),float(input("Δώσε 2η τιμή: ")) #Με αυτή την εντολή εισάγουμε 2 δεδομένα. Με το float μετατρέπουμε το δεδομένο από string σε float. Οι 2 τιμές που εισάγονται καταχωρούνται αντίστοιχα η 1η στο x και η 2η στο y

```

print(f" Πριν την εκτέλεση της συνάρτησης swap x={x} και y={y}")
swap(x, y)
print(f" Μετά την εκτέλεση της συνάρτησης swap x={x} και y={y}")

```

Πρόγραμμα που διαβάζει 2 τιμές, τις μεταβιβάζει σε συνάρτηση η οποία τις εναλλάσσει (αντιμεταθέτει) και οι τιμές τυπώνονται στο main εναλλαγμένες. **ΣΩΣΤΟ ΠΡΟΓΡΑΜΜΑ ΠΟΥ ΚΑΝΕΙ ΕΝΑΛΛΑΓΗ** διότι η εναλλαγή που γίνεται εντός της συνάρτησης επιστρέφεται στο main

```

def swap(a,b):
    temp=a
    a=b
    b=temp
    return a, b #Οι εναλλαγμένες τοπικές μεταβλητές επιστρέφουν πίσω στο main. H Python μπορεί σε ένα return να επιστρέψει πολλά αποτελέσματα

```

x, y=float(input("Δώσε 1η τιμή: ")),float(input("Δώσε 2η τιμή: ")) #Με αυτή την εντολή εισάγουμε 2 δεδομένα. Με το float μετατρέπουμε το δεδομένο από string σε float. #Οι 2 τιμές που εισάγονται καταχωρούνται αντίστοιχα η 1η στο x και η 2η στο y

```

print(f" Πριν την εκτέλεση της συνάρτησης swap x={x} και y={y}")

```

```
x, y=swap(x, y) #Καλούμε τη συνάρτηση swap(x, y) μέσω καταχώρισης των αποτελεσμάτων που επιστρέφει στα x και y αντίστοιχα
```

```
print(f" Μετά την εκτέλεση της συνάρτησης swap x={x} και y={y}")
```

Παράδειγμα Συνάρτησης με προεπιλεγμένες τιμές στις παραμέτρους της

#Συνάρτηση με προεπιλεγμένες τιμές (default τιμές) στα ορίσματα της

```
def whole_name(name="Panagiotis", surname="Papadopoulos", age=25):
```

return name + " " + surname + " " + str(age) #Η συνάρτηση str μετατρέπει το age σε string. Ο Τελεστής + κάνει συνένωση σε αλφαριθμητικά, άρα εδώ επιστρέφεται η μεταβλητή name που στο τέλος της συνενώνεται ένα κενό και μετά συνενώνεται η μεταβλητή age που μετατρέπεται σε string

```
print(whole_name("Maria", "Nikolaou", 20)) #καλείται με 3 ορίσματα οπότε ΔΕΝ μπαίνει καμία default τιμή
```

```
print(whole_name("Eleni", "Georgiou")) #καλείται με 2 ορίσματα οπότε μπαίνει 1 default τιμή. Συγκεκριμένα στο όρισμα age μπαίνει η τιμή 25
```

```
print(whole_name("Giorgos")) #καλείται με 1 όρισμα οπότε μπαίνουν 2 default τιμές. Συγκεκριμένα στο όρισμα surname μπαίνει η τιμή "Papadopoulos" και στο age μπαίνει η τιμή 25
```

```
print(whole_name()) #καλείται χωρίς ορίσματα οπότε μπαίνουν οι 3 default τιμές. Συγκεκριμένα στο όρισμα name μπαίνει η τιμή "Panagiotis", στο όρισμα surname μπαίνει η τιμή "Papadopoulos" και στο όρισμα age μπαίνει η τιμή 25
```

Παράδειγμα Παράμετροι Συνάρτησης-ΟΧΙ

```
def my_function(fname):
```

```
    print(fname + " Refsnes")
```

```
my_function("Emil") → καλείται η συνάρτηση my_function με όρισμα την τιμή Emil και τυπώνεται το μήνυμα Emil Refsnes  
my_function("Tobias") → καλείται η συνάρτηση my_function με όρισμα την τιμή Tobias και τυπώνεται το μήνυμα Tobias Refsnes  
my_function("Linus") → καλείται η συνάρτηση my_function με όρισμα την τιμή Linus και τυπώνεται το μήνυμα Linus Refsnes
```

Παράδειγμα με Εξορισμού τιμές στις Παράμετροι Συνάρτησης-ΟΧΙ

Αν καλέσουμε μια συνάρτηση χωρίς παραμέτρους τότε θα ισχύουν οι προεπιλεγμένες τιμές των παραμέτρων

```
def my_function(country = "Norway"):
```

```
    print("I am from " + country)
```

```
my_function("Sweden") # καλείται η συνάρτηση my_function με όρισμα Sweden και τυπώνεται το μήνυμα I am from Sweden  
my_function("India") # καλείται η συνάρτηση my_function με όρισμα India και τυπώνεται το μήνυμα I am from India  
my_function() #καλείται η συνάρτηση my_function χωρίς όρισμα και τυπώνεται το μήνυμα I am from Norway  
my_function("Brazil") #καλείται η συνάρτηση my_function με όρισμα Brazil και τυπώνεται το μήνυμα I am from Brazil
```

Παράδειγμα Μεταβίβαση Λίστας ως Παράμετρο Συνάρτησης-ΟΧΙ

```
def my_function(food):
```

```
    for x in food:
```

```
        print(x)
```

```
fruits = ["apple", "banana", "cherry"]
```

`my_function(fruits)` #καλείται η συνάρτηση `my_function` με όρισμα τη λίστα `fruits` και τυπώνεται η ακόλουθη λίστα:

```
apple  
banana  
cherry
```

Παράδειγμα Επιστροφή Τιμής από συνάρτηση-ΟΧΙ

```
def my_function(x):
```

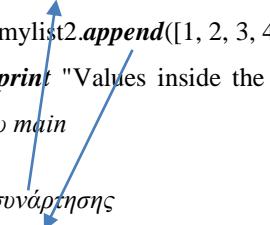
```
    return 5*x
```

`print(my_function(3))` #καλείται η συνάρτηση `my_function` με όρισμα την τιμή 3. Αυτή υπολογίζει και επιστρέφει το 15 που τυπώνεται
`print(my_function(5))` #καλείται η συνάρτηση `my_function` με όρισμα την τιμή 5. Αυτή υπολογίζει και επιστρέφει το 25 που τυπώνεται
`print(my_function(9))` #καλείται η συνάρτηση `my_function` με όρισμα την τιμή 9. Αυτή υπολογίζει και επιστρέφει το 45 που τυπώνεται

Παράδειγμα Ορίσματα Συνάρτησης με call by reference

Όλες οι παράμετροι (ορίσματα) στη Python περνούν με αναφορά. Αυτό σημαίνει ότι αν αλλάξουμε την παράμετρο που αναφέρεται σε μια συνάρτηση, η αλλαγή αυτή επηρεάζει και το κύριο πρόγραμμα. Για παράδειγμα:

```
# Ορισμός Function  
def change(mylist2):  
    mylist2.append([1, 2, 3, 4])  
    print "Values inside the function: ", mylist2 #επειδή η μεταβλητή mylist2 δεν επαναφέρεται εδώ επιδρά απευθείας στη mylist του main  
  
#Κλήση συνάρτησης  
mylist = [10,20,30]  
change(mylist)  
print("Values outside the function: ", mylist)
```



Αποτελέσματα Εκτέλεσης

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

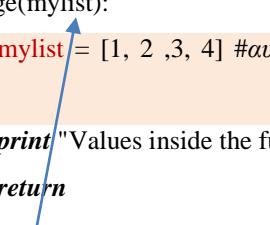
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

Παράδειγμα Ορίσματα Συνάρτησης με τοπική μεταβλητή

```
# Ορισμός Function
```

```
def change(mylist):
```

```
    mylist = [1, 2, 3, 4] #αυτή είναι μια νέα τοπική μεταβλητή με όνομα mylist μέσα στην συνάρτηση που παίρνει τις τιμές [1,2,3,4]  
    print "Values inside the function: ", mylist #τυπώνεται η τοπική μεταβλητή mylist με τις τιμές [1,2,3,4]  
    return
```



```
#Κλήση συνάρτησης
```

```
mylist = [10,20,30] #αυτή είναι μια τοπική μεταβλητή του main
```

```
change(mylist)
```

```
print "Values outside the function: ", mylist #τυπώνεται η τοπική μεταβλητή mylist του main με τις τιμές [10,20,30];
```

Η παράμετρος mylist είναι τοπική στη συνάρτηση changeme. Η αλλαγή της λίστας στη συνάρτηση δεν επηρεάζει τη λίστα του main. Η συνάρτηση δεν αλλάζει τίποτα στην καθολική μεταβλητή mylist και τελικά αυτό θα παράγει το ακόλουθο αποτέλεσμα -

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

Μεταβίβαση Λίστας σε Συνάρτηση (Call By reference)

#Συνάρτηση με μεταβίβαση Λίστας και επιστροφή αποτελέσματος

def changelist(listfun): #Στην παράμετρο listfun μεταβιβάζεται η διεύθυνση της λίστας listmain από το main. Άρα όλες οι αλλαγές που θα γίνουν στο listfun ενημερώνουν τη λίστα listmain του main και με τη χρήση του return. Και πάλι ΟΛΕΣ ΟΙ ΑΛΛΑΓΕΣ ΕΔΩ ΓΙΝΟΝΤΑΙ ΑΠΕΥΘΕΙΑΣ ΣΤΗ ΛΙΣΤΑ listmain

```
listfun.append(40)
listfun.append(50)
listfun.remove(10);
return listfun

listmain = [10, 20, 30]

print(f" Λίστα πριν την εκτέλεση της συνάρτησης changelist" {listmain} ")
listmain=changelist(listmain) #Καλούμε τη συνάρτηση changelist και μεταβιβάζουμε τη λίστα
print(f" Λίστα μετά την εκτέλεση της συνάρτησης changelist" {listmain} ")
```

Βασικό Συμπέρασμα

ΤΟ ΣΥΜΠΕΡΑΣΜΑ ΕΙΝΑΙ ΟΤΙ ΟΤΑΝ ΜΕΤΑΒΙΒΑΖΟΥΜΕ ΜΙΑ ΛΙΣΤΑ ΣΕ ΣΥΝΑΡΤΗΣΗ, ΤΟΤΕ ΟΠΟΙΑΔΗΠΟΤΕ ΑΛΛΑΓΗ ΚΑΝΟΥΜΕ ΣΤΗ ΛΙΣΤΑ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ ΑΠΕΙΚΟΝΙΖΕΤΑΙ ΑΠΕΥΘΕΙΑΣ ΣΤΗ ΛΙΣΤΑ ΤΟΥ MAIN ΕΙΤΕ ΚΑΝΟΥΜΕ RETURN ΕΙΤΕ OXI

Μεταβίβαση Λίστας σε Συνάρτηση (Κατάργηση του Call By reference)

#Συνάρτηση με μεταβίβαση Λίστας και δημιουργία νέας τοπικής λίστας και επιστροφής της

```
def squareoflist(listfun):
    templist=[] #Η λίστα templist είναι μια νέα τοπική μεταβλητή της συνάρτησης
    for i in range(len(listfun)): #Η έτοιμη συνάρτηση len υπολογίζει το πλήθος στοιχείων μιας λίστας
        templist.append(listfun[i]**2) #Τα τετράγωνα των στοιχείων της λίστας listfun που έχει λάβει τις τιμές της λίστας listmain από το main προστίθενται στη λίστα templist
    return templist #Η νέα λίστα επιστρέφεται από τη συνάρτηση διότι οι αλλαγές ΔΕΝ γίνονται στη λίστα του main αλλά σε μια τοπική λίστα οπότε πρέπει να επιστραφεί
```

listmain=[10,20,30]

reslist=squareoflist(listmain) #στη reslist επιστρέφεται η λίστα της συνάρτησης με τα τετράγωνα

print(f" Η αρχική λίστα είναι: {listmain}, Τα τετράγωνα της λίστας είναι: {reslist}")

Παράδειγμα Ορίσματα Μεταβλητού Μήκους (Variable-length arguments)-OXI

Μπορεί να χρειαστεί να επεξεργαστούμε μια συνάρτηση με περισσότερα ορίσματα από όσα ορίσαμε στη δήλωση της. Αυτά τα ορίσματα ονομάζονται ορίσματα μεταβλητού μήκους και δεν αναφέρονται στον ορισμό της συνάρτησης, σε αντίθεση με τα απαιτούμενα.

μενα και τα προεπιλεγμένα ορίσματα. Η σύνταξη για μια συνάρτηση με ορίσματα μεταβλητών χωρίς λέξεις-κλειδιά είναι η ακόλουθη:

```
def functionname([formal_args,] *var_args_tuple ):  
    "function_docstring"  
    function_suite  
    return [expression]
```

Ένας αστερίσκος (*) τοποθετείται πριν από το όνομα της μεταβλητής που διατηρεί τις τιμές όλων των ορισμάτων χωρίς λέξεις-κλειδιά. Αυτή η πλειάδα παραμένει κενή εάν δεν έχουν καθοριστεί πρόσθετα ορίσματα κατά τη διάρκεια της κλήσης συνάρτησης. Ακολουθεί ένα απλό παράδειγμα:

Ορισμός Function

```
def printinfo(arg1, *vartuple):#αντό δηλώνει ότι υπάρχει ένα υποχρεωτικό όρισμα arg1 και μια πλειάδα vartuple που μπορεί να παίρνει όλα τα υπόλοιπα ορίσματα που ο αριθμός τους δεν είναι προκαθορισμένος
```

```
    print "Output is: "  
    print arg1 #εδώ τυπώνεται το 10  
    for var in vartuple:  
        print var #εδώ δεν τυπώνεται τίποτα  
    return
```

#Κλήση συνάρτησης

```
printinfo(10) #το 10 μεταβιβάζεται στο arg1. Το vartuple είναι μια κενή πλειάδα ορισμάτων  
#τυπώνονται τα αποτελέσματα
```

Output is:10

Ορισμός Function

```
def printinfo(arg1, *vartuple):#αντό δηλώνει ότι υπάρχει ένα υποχρεωτικό όρισμα arg1 και μια πλειάδα vartuple που μπορεί να παίρνει όλα τα υπόλοιπα ορίσματα που ο αριθμός τους δεν είναι προκαθορισμένος
```

```
    print "Output is: "  
    print arg1 #εδώ τυπώνεται το 70  
    for var in vartuple:  
        print var #εδώ τυπώνονται οι τιμές 60, 50  
    return;
```

```
printinfo(70, 60, 50) #το 70 μεταβιβάζεται στο arg1 και τα υπόλοιπα ορίσματα 60,50 στην παράμετρο vartuple
```

Όταν εκτελεστεί το πρόγραμμα παράγονται τα ακόλουθα αποτελέσματα:

Output is:

70

60

50

5.10.1 Εντολή return

Η δήλωση return [έκφραση] βγαίνει από μια συνάρτηση, προαιρετικά επιστρέφοντας μια έκφραση στον καλούντα. Μια δήλωση επιστροφής χωρίς ορίσματα είναι ίδια με την εντολή return none. Όλα τα παραπάνω παραδείγματα δεν επιστρέφουν καμία τιμή. Μπορούμε να επιστρέψουμε μια τιμή από μια συνάρτηση ως εξής

Ορισμός Function

def sum(arg1, arg2):

total= arg1 + arg2 #δήλωση τοπικής μεταβλητής με όνομα total που αθροίζει τα ορίσματα arg1 και arg2

print "Inside the function : ", total

return total;

#Κλήση συνάρτησης

total = sum(10, 20); #η συνάρτηση καλείται με καταχώριση

print "Outside the function : ", total

Όταν εκτελεστεί το πρόγραμμα παράγονται τα ακόλουθα αποτελέσματα:

Αποτελέσματα Εκτέλεσης

Inside the function:30

Outside the function :30

5.11 Συναρτήσεις Λάμβδα (Lambda Functions)

Μια συνάρτηση Λάμβδα είναι μια μικρή ανώνυμη συνάρτηση που μπορεί να λάβει οποιοδήποτε αριθμό παραμέτρων αλλά να έχει μόνο μια έκφραση. Αυτές οι συναρτήσεις ονομάζονται ανώνυμες επειδή δεν δηλώνονται με τον τυπικό τρόπο χρησιμοποιώντας τη λέξη-κλειδί def. Μπορούμε να χρησιμοποιήσουμε τη λέξη-κλειδί **lambda** για να δημιουργήσουμε μικρές ανώνυμες συναρτήσεις. Οι συναρτήσεις λάμδα μπορούν να λάβουν οποιονδήποτε αριθμό ορισμάτων αλλά να επιστρέψουν μόνο μία τιμή με τη μορφή μιας έκφρασης.

Σύνταξη Η σύνταξη των συναρτήσεων λάμδα περιέχει μόνο μία δήλωση, η οποία έχει ως εξής -

lambda [arg1 [,arg2,.....argn]]:expression. Τα [] δηλώνουν προαιρετικά μέλη

Το επόμενο πρόγραμμα δείχνει τη λειτουργία των συναρτήσεων lambda:

Ορισμός lambda function

sum=lambda arg1, arg2: arg1+arg2 #δήλωση μιας ανώνυμης συνάρτησης με το keyword lambda που λαμβάνει 2 ορίσματα arg1, arg2. Η : δηλώνει το σώμα της συνάρτησης μέσα στο οποίο υπάρχει η πράξη arg1+arg2 και η lambda συνάρτηση έχει αυτόματο return

#Κλήση lambda συναρτήσεων

print "Value of total : ", sum(10, 20) #καλείται η lambda συνάρτηση μέσω της μεταβλητής sum στην οποία καταχωρείται το επιστρεφόμενο της αποτέλεσμα. Το 10 μεταβιβάζεται στο arg1, το 20 στο arg2, γίνεται η πράξη arg1+arg2 και η lambda συνάρτηση επιστρέφει το τελικό αποτέλεσμα 10+20=30 στη μεταβλητή sum και στο print τυπώνεται 30

print "Value of total : ", sum(20, 20) #καλείται η lambda συνάρτηση μέσω της μεταβλητής sum στην οποία καταχωρείται το επιστρεφόμενο της αποτέλεσμα. Το 20 μεταβιβάζεται στο arg1, το 20 στο arg2, γίνεται η πράξη arg1+arg2 και η lambda συνάρτηση επιστρέφει το τελικό αποτέλεσμα 20+20=40 στη μεταβλητή sum και στο print τυπώνεται 40

Αποτελέσματα Εκτέλεσης

Value of total : 30

Value of total : 40

Παράδειγμα με Επώνυμες και Ανώνυμες Συναρτήσεις (Lambda Functions)

def square(x):return x*x #Δήλωση **Επώνυμης Συνάρτησης με όνομα square** που λαμβάνει όρισμα x και επιστρέφει το τετράγωνο του

def cube(x):return x*x*x #Δήλωση **Επώνυμης Συνάρτησης** με όνομα cube που λαμβάνει ένα όρισμα x και επιστρέφει τον κύβο του

def power(x, y):return x**y #Δήλωση **Επώνυμης Συνάρτησης με όνομα power** που λαμβάνει ορίσματα x, y και επιστρέφει το x^y

squarel=lambda x :x*x # Δήλωση **Ανώνυμης (lambda)** συνάρτησης που λαμβάνει όρισμα x και επιστρέφει το τετράγωνο του στη μεταβλητή squarel

cubel=lambda x: x*x*x # Δήλωση **Ανώνυμης (lambda)** συνάρτησης που λαμβάνει όρισμα x και επιστρέφει τον κύβο του στη μεταβλητή cubel

powerl=lambda x, y: x**y # Δήλωση **Ανώνυμης (lambda)** συνάρτησης που λαμβάνει όρισμα x και όρισμα y και επιστρέφει τη δύναμη x^y στη μεταβλητή powerl

x=int(input("δώσε βάση: "))

```
y=int(input("δώσε εκθέτη: "))
```

```
print(f" Με κλήση Επώνυμων Συναρτήσεων τα αποτελέσματα είναι: Τετράγωνο του {x}={square(x)}, Κύβος του {x}={cube(x)}, Δύναμη του {x}^{y}={power(x,y)}")
```

```
print(f" Με κλήση Ανώνυμων Lambda Συναρτήσεων τα αποτελέσματα είναι: Τετράγωνο του {x}={square(x)}:, Κύβος του {x}={cubel(x)}:, Δύναμη του {x} ^ {y}={powerl(x,y)}")
```

Παρατήρηση:

Οι lambda συναρτήσεις καλούνται ως εξής: **όνομα_μεταβλητής που επιστρέφεται το αποτέλεσμα(όρισμα)**

Οι επώνυμες συναρτήσεις καλούνται ως εξής: **όνομα_συνάρτησης(όρισμα)**

Παράδειγμα 1

lambda function που προσθέτει 10 στο όρισμα της και επιστρέφει το αποτέλεσμα:

```
x=lambda a: a + 10
```

```
print(x(5)) #Τυπώνεται 15. Ουσιαστικά με το x(5) καλείται η lambda συνάρτηση, η οποία μετονομάζεται σε x. Το 5 μεταβιβάζεται στο a και στο a προστίθεται το 10. Υπολογίζεται η τιμή 5+10 και επιστρέφεται στο σημείο κλήσης της συνάρτησης lambda και τυπώνεται
```

Παράδειγμα 2

lambda function που πολλαπλασιάζει τα ορίσματα και επιστρέφει το αποτέλεσμα

```
x=lambda a, b: a*b
```

```
print(x(5, 6)) #Τυπώνεται 30. Ουσιαστικά με το x(5) καλείται η lambda συνάρτηση, η οποία μετονομάζεται σε x. Το 5 μεταβιβάζεται στο a, το 6 στο b και πολλαπλασιάζονται τα ορίσματα a, b επιστρέφεται το γινόμενο
```

Παράδειγμα 3

lambda function που αθροίζει τα ορίσματα της και επιστρέφει το αποτέλεσμα:

```
x=lambda a, b, c : a + b + c
```

```
print(x(5, 6, 2)) #Τυπώνεται 13
```

Παράδειγμα 4

Η δυναμική των lambda function είναι όταν χρησιμοποιούνται ως ανώνυμες συναρτήσεις εντός μιας άλλης συνάρτησης.

Για παράδειγμα έχουμε μια συνάρτηση που λαμβάνει όρισμα και το πολλαπλασιάζει με ένα άγνωστο αριθμό. Μπορούμε να χρησιμοποιήσουμε τον ίδιο ορισμό συνάρτησης για να κατασκευάσουμε μια συνάρτηση η οποία πάντα διπλασιάζει τον αριθμό που της στέλνουμε όπως στο επόμενο παράδειγμα:

```
def myfunc(n): #λαμβάνει όρισμα n
```

```
    return lambda a: a*n #στο return ορίζουμε μια lambda συνάρτηση που λαμβάνει όρισμα a και το πολλαπλασιάζει με το n
```

```
mydoubler =myfunc(2) #η myfunc(2) εδώ δεν καλείται
```

```
print(mydoubler(11)) # Με τη mydoubler(11) γίνονται δύο πράγματα ταυτόχρονα: Καλείται πρώτα η συνάρτηση myfunc με όρισμα 2 και μετά καλείται η lambda συνάρτηση με όρισμα 11 το οποίο μεταβιβάζεται στην παράμετρο a. Υπολογίζεται το αποτέλεσμα a*n=11*2=22 και επιστρέφεται και εκτυπώνεται
```

```
def myfunc(n):  
    return lambda a: a*n
```

mytripler = myfunc(3) η *myfunc(3)* εδώ δεν καλείται

print(mytripler(11)) #Με την *mytripler* (11) γίνονται δύο πράγματα ταυτόχρονα: Καλείται πρώτα η συνάρτηση *myfunc* με όρισμα 3 και μετά καλείται η *lambda* συνάρτηση με όρισμα 11 το οποίο μεταβιβάζεται στην παράμετρο *a*. Υπολογίζεται το αποτέλεσμα $a^*n=11*3=33$ και επιστρέφεται και εκτυπώνεται

Μπορούμε να χρησιμοποιήσουμε τον ίδιο ορισμό συνάρτησης για κάνουμε κα τα δύο όπως στο επόμενο παράδειγμα:

```
def myfunc(n):  
    return lambda a: a*n
```

mydoubler = myfunc(2) //εδώ δεν καλείται η *myfunc* με τιμή 2

mytripler = myfunc(3) // εδώ δεν καλείται η *myfunc* με τιμή 3

print(mydoubler(11)) # Με τη *mydoubler*(11) γίνονται δύο πράγματα ταυτόχρονα: Καλείται πρώτα η συνάρτηση *myfunc* με όρισμα 2 και μετά καλείται η *lambda* συνάρτηση με όρισμα 11 το οποίο μεταβιβάζεται στην παράμετρο *a*. Υπολογίζεται το αποτέλεσμα $a^*n=11*2=22$ και επιστρέφεται και εκτυπώνεται

print(mytripler(11)) #Με την *mytripler* (11) γίνονται δύο πράγματα ταυτόχρονα: Καλείται πρώτα η συνάρτηση *myfunc* με όρισμα 3 και μετά καλείται η *lambda* συνάρτηση με όρισμα 11 το οποίο μεταβιβάζεται στην παράμετρο *a*. Υπολογίζεται το αποτέλεσμα $a^*n=11*3=33$ και επιστρέφεται και εκτυπώνεται

5.12 Αναδρομή σε Python

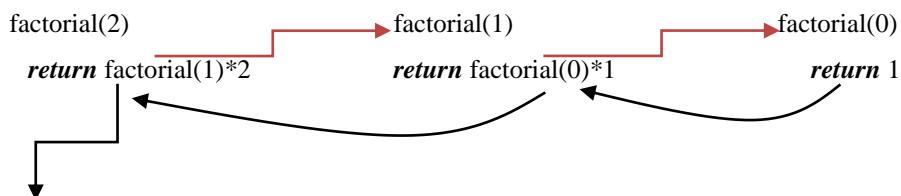
Παράδειγμα 1 – Υπολογισμός του $n!$ Με αναδρομική συνάρτηση

Το παράδειγμα αυτό παρουσιάζει μια αναδρομική συνάρτηση για τον υπολογισμό του $n!$. Ο αναδρομικός τύπος υπολογισμού του $n!$ είναι: $\text{factorial}(n) = \text{factorial}(n-1) * n$ με αρχική συνθήκη $0! = 1$.

Ο υπολογισμός του $n!$ με αναδρομική συνάρτηση γίνεται ως εξής:

```
def factorial(n):
    if n == 0: #συνθήκη τερματισμού αναδρομής
        return 1
    else:
        return factorial(n-1)*n #αναδρομική κλήση
```

Παράδειγμα Αναδρομικής Εκτέλεσης για $n=2$



Το τελικό αποτέλεσμα που είναι το $2 = 2! = 1 * 2$ επιστέφεται στο κυρίως πρόγραμμα

Ο υπολογισμός του $n!$ με επαναληπτική συνάρτηση γίνεται ως εξής:

```
def factorial(n):
    result = 1
    for i in range(2, n+1): #το i λαμβάνει τιμές από 2 μέχρι και το n
        result *= i #result=result*i

    return result
```

Παράδειγμα 2 – Υπολογισμός του n -οστού όρου της ακολουθίας Fibonacci

Το παράδειγμα αυτό παρουσιάζει μια αναδρομική συνάρτηση για τον υπολογισμό του n -οστού όρου της ακολουθίας Fibonacci.

Υπενθυμίζουμε ότι ο αναδρομικός τύπος υπολογισμού του n -οστού όρου της ακολουθίας Fibonacci είναι $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ με αρχικές συνθήκες $\text{fib}(0) = 0$ και $\text{fib}(1) = 1$

Ο αναδρομικός τύπος υπολογισμού του $n!$ είναι: $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ με αρχικές συνθήκες $\text{fib}(0) = 0$ και $\text{fib}(1) = 1$

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

Ο υπολογισμός του του n -οστού όρου της ακολουθίας Fibonacci με επαναληπτική συνάρτηση γίνεται ως εξής:

```
def fib(n):
    a, b = 0, 1 #στο a καταχωρείται το 0 και στο b καταχωρείται το 1
    for i in range(n+1): #το i παίρνει τιμές από 1 μέχρι n
        a, b = b, a + b #a=b και b=a+b
    return a
```

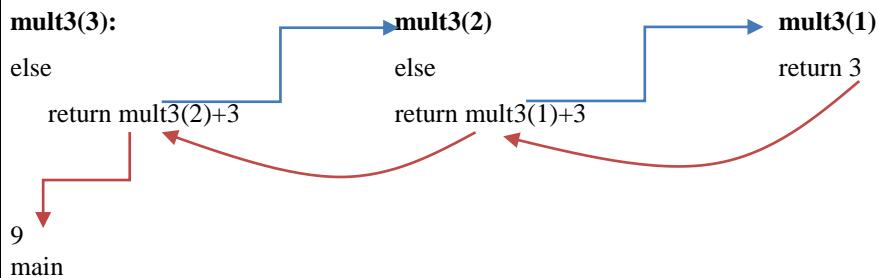
Παράδειγμα 3 – Υπολογισμός αναδρομικού τύπου

Υπολογισμός του αναδρομικού τύπου $f(n) = f(n-1) + 3$ με αρχική συνθήκη $f(1) = 3$,

```
def mult3(n):
    if n == 1:
        return 3
    else:
        return mult3(n-1) + 3

print(mult3(3))
```

Παράδειγμα Αναδρομικής Εκτέλεσης για $n=3$



Παράδειγμα 4 – Αναδρομική Συνάρτηση για τον υπολογισμό των αθροίσματος λίστας

```
import random
def sum_list(lis):
    if lis == []:#όταν η λίστα είναι κενή επιστρέφει ως άθροισμα το 0
        return 0
    else:
        return lis[0] + sum_list(lis[1:]) #στο 1ο στοιχείο της λίστας αθροίζονται όλα τα υπόλοιπα
```

```
specificlist=[1,2, 3]
```

```
print(f"Άθροισμα Λίστας {specificlist}:", sum_list(specificlist))
```

```
randlist=[]
```

```
n=int(input("Δώσε μέγεθος λίστας που θα γεμίσει με random τιμές από 1-10: "))
```

```
for i in range(n+1): #επανάληψη από 1 μέχρι n
    randlist.append(random.randrange(1, 10)) #Η λίστα randlist γεμίζει με τυχαίες τιμές από 1 .. 10 που παράγονται από τη συνάρτηση randrange στο εύρος 1..10
```

```
print(f" Η Λίστα με random τιμές από 1-10 είναι η {randlist} και έχει άθροισμα:", sum_list(randlist))
```

```

n = int(input("Δώσε μέγεθος λίστας που θα γεμίσει με random τιμές: "))
beg, end = int(input("Δώσε εύρος random τιμών (Από): ")), int(input("Δώσε εύρος random τιμών (Εως): "))
randlist2=[]
for i in range(n + 1):
    randlist2.append(random.randrange(beg, end)) # H λίστα randlist γεμίζει με τυχαίες τιμές από beg..end που παράγονται από
    τη συνάρτηση randrange στο εύρος 1..10

print(f" Η Λίστα με random τιμές από {beg} έως {end} είναι η {randlist2} και έχει άθροισμα:", sum_list(randlist2))
#print(random.randrange(20, 50, 3)) #To πρωτότυπο της είναι randrange(beg, end, step)

```

Παράδειγμα Αναδρομικής Εκτέλεσης για n=3

specificlist=[1, 2, 3]

sum_list(lis=[1, 2, 3]): else: <i>return lis[0] + sum_list(lis[1:])</i>	specificlist(lis=[2, 3]) else: <i>return lis[0] + sum_list(lis[1:])</i>	specificlist(lis=[3]) else: <i>return lis[0] + sum_list(lis[1:])</i>	specificlist(lis=[]) else: <i>return 0;</i>
--	--	---	--

6 ←

main

5.13 Κλάσεις/Αντικείμενα (Classes/Objects)

Παράδειγμα 1 με δήλωση κλάσης με δημιουργό και μεθόδους και δημιουργία στιγμιοτύπων

class Person:

#To self σε μια συνάρτηση δηλώνει ότι αποτελεί μέλος της κλάσης. Τα μέλη της κλάσης δεν δηλώνονται ξεχωριστά αλλά προσδιορίζονται με το self

def __init__(self, name, age): #εδώ ορίζουμε το δημιουργό της κλάσης με τη δήλωση __init__. Καλείται αυτόματα κάθε φορά που δηλώνουμε στιγμιότυπο τύπου Person και το αρχικοποιεί. Ο δημιουργός καλείται μόνο 1 φορά για το κάθε στιγμιότυπο. Εδώ λαμβάνει 2 ορίσματα

self.name=name #στο μέλος name που προσδιορίζεται με το self καταχωρείται το όρισμα name

self.age=age #στο μέλος age που προσδιορίζεται με το self καταχωρείται το όρισμα age

def print_person(self): #Δήλωση μεθόδου κλάσης με όνομα print_person. Κάθε μέθοδος κλάσης πρέπει κατ' ελάχιστο να περιέχει τη λέξη self. Η μέθοδος δεν λαμβάνει ορίσματα

print("Όνομα =", self.name, ", Ήλικια =", self.age)

def setname(self, nam): #Δήλωση μεθόδου κλάσης με όνομα setname. Η μέθοδος κλάσης setname έχει και το όρισμα self αλλά και το όρισμα nam

self.name=nam #Στο μέλος name που το προσδιορίζουμε με το self καταχωρείται η παράμετρος nam

def setage(self, age): #Δήλωση μεθόδου κλάσης με όνομα setage. Η μέθοδος κλάσης setage έχει και το όρισμα self αλλά και το όρισμα nam

self.age=age #Στο μέλος age που το προσδιορίζουμε με το self καταχωρείται η παράμετρος age

def getname(self): #Δήλωση μεθόδου κλάσης με όνομα getname. Κάθε μέθοδος κλάσης πρέπει κατ' ελάχιστο να περιέχει τη λέξη self

return self.name #επιστρέφεται το μέλος name που προσδιορίζεται με self

def getage(self): #Δήλωση μεθόδου κλάσης με όνομα getage. Κάθε μέθοδος κλάσης πρέπει κατ' ελάχιστο να περιέχει τη λέξη self

return self.age # επιστρέφεται το μέλος name που προσδιορίζεται με self

def all_details(self): #ήλωση μεθόδου κλάσης με όνομα all_details. Η μέθοδος αυτή επιστρέφει όλα τα μέλη συνενωμένα ως ένα αλφαριθμητικό

return f" Το Άτομο {self.name} έχει ηλικία {self.age}"

p1=Person("Nikos",36) #εδώ δηλώνουμε στιγμιότυπο της κλάσης Person και καλείται ο δημιουργός και το αρχικοποιεί

print("Όνομα =", p1.name, ", Ήλικια =", p1.age) #όνομα στιγμιοτύπου.όνομα μέλους

p1.print_person() #καλείται η μέθοδος print_person() για το στιγμιότυπο p1. Γράφουμε όνομα στιγμιοτύπου.όνομα μεθόδου

name=input("Δώσε Όνομα 2ου Person: ")

age=int(input("Δώσε Ήλικία 2ου Person: "))

#δήλωση 2ου στιγμιοτύπου με όνομα p2, κλήση δημιουργού για αρχικοποίηση του p2

```
p2=Person(name, age) #εδώ δηλώνουμε στιγμιότυπο της κλάσης Person και καλείται ο δημιουργός της
```

```
print(f"Όνομα = {p2.name} Ηλικία= {p2.age}")
```

```
p2.print_person() #καλείται η μέθοδος print_person() για το στιγμιότυπο p2
```

```
p3=Person(input("Δώσε Όνομα 3ου Person "),int(input("Δώσε Ηλικία 3ου Person "))) #εδώ εισάγουμε 2 στοιχεία με τις αντίστοιχες input και καλείται αυτόματα ο δημιουργός της κλάσης Person για αρχικοποίηση του p3
```

```
print("Όνομα =",p3.name, ", Ηλικία=",p3.age)
```

```
p3.print_person() #καλείται η μέθοδος print_person() για το στιγμιότυπο p3
```

```
p1.setname('Γιάννης') #Καλείται η μέθοδος setname για το στιγμιότυπο p1 και τροποποιεί το μέλος name
```

```
p1.setage(29) #Καλείται η μέθοδος setage για το στιγμιότυπο p1 και τροποποιεί το μέλος age
```

```
p1.print_person() #καλείται η μέθοδος print_person() για το στιγμιότυπο p1 για να τυπώνει τα νέα στοιχεία του
```

```
#με τον επόμενο κώδικα εντοπίζουμε το άτομο από τα p2 και p3 με τη μεγαλύτερη ηλικία
```

```
maxage=p1.getage()
```

```
maxname=p1.getname()
```

```
if (p2.getage())>maxage):
```

```
    maxage=p2.getage()
```

```
    maxname = p2.getname()
```

```
if (p3.getage())>maxage):
```

```
    maxage=p3.getage()
```

```
    maxname = p3.getname()
```

```
print(f"Το Άτομο με όνομα {maxname} έχει τη μεγαλύτερη ηλικία που είναι {maxage}")
```

```
print(p1.all_details())
```

```
print(p2.all_details())
```

```
print(p3.all_details())
```

```
persons=[] #Ορισμός Λίστας που αρχικά είναι κενή
```

```
#Με τις 3 επόμενες συναρτήσεις προσθέτουμε τα 3 στιγμιότυπα p1,p2,p3 στη λίστα persons
```

```
persons.append(p1)
```

```
persons.append(p2)
```

```
persons.append(p3)
```

```
print("\n Λίστα με Στιγμιότυπα")
```

```
for i in persons: #σε κάθε επανάληψη στο i μπαίνει ένα στιγμιότυπο από τη λίστα persons
```

```
    print(i.all_details())
```

```
print("\n Λίστα με Στιγμιότυπα (2)")
```

```

for i in range(len(persons)): # σε κάθε επανάληψη στο i μπαίνει ένας αριθμός από 0 έως len(persons))-1
    print(persons[i].all_details())

persons=persons[::-1] #αυτή η εντολή αντιστρέφει τη λίστα και την καταχωρεί πάλι στο persons
print("\n Αντίστροφη Λίστα με Στιγμιότυπα")
for i in range(len(persons)): # σε κάθε επανάληψη στο i μπαίνει ένας αριθμός από 0 έως len(persons))-1
    print(persons[i].all_details())

```

Παράδειγμα 2 με δήλωση κλάσης με δημιουργό και μεθόδους και δημιουργία στιγμιοτύπων

Για να δημιουργήσουμε μια νέα κλάση χρησιμοποιούμε την εντολή class και ορίζουμε κάποιες ιδιότητες σε αυτή την κλάση. Στη συνέχεια δηλώνουμε αντικείμενα στην κλάση αυτή. Στο επόμενο παράδειγμα ορίζουμε μια νέα κλάση με όνομα MyClass με την ιδιότητα (μέλος) x και μετά ορίζουμε το αντικείμενο p1 στην κλάση αυτή και τυπώνουμε την τιμή της ιδιότητας x.

`class MyClass:`

`x = 5 #το x είναι μέλος της κλάσης MyClass και αρχικοποιείται με την τιμή 5. Τα μέλη μιας κλάσης στην Python είναι εξορισμού public`

`p1 = MyClass() #δημιουργείται το αντικείμενο p1 καλώντας το δημιουργό της κλάσης MyClass`

`print(p1.x) #Τυπώνεται το μέλος x του αντικειμένου p1 που έχει τιμή 5`

Παράδειγμα 2-Δημιουργός Κλάσης

Όλες οι κλάσεις έχουν ένα δημιουργό. Ο δημιουργός κάθε κλάσης στην Python είναι η εξορισμού συνάρτηση `__init__()`. Ο δημιουργός καλείται αυτόματα κάθε φορά που δηλώνουμε ένα αντικείμενο στην κλάση και αρχικοποιεί τα μέλη του. Στο επόμενο παράδειγμα ορίζουμε μια κλάση με όνομα Person και καλούμε τη συνάρτηση `__init__()` (δηλ. το δημιουργό) για να δώσει αρχικές τιμές στις ιδιότητες-μέλη name και age

`class Person:`

`def __init__(self, name, age): #δήλωση δημιουργού. Είναι μια συνάρτηση που καλείται αυτόματα και δεσμεύει μνήμη για ένα στιγμιότυπο. Ο δημιουργός λαμβάνει 2 ορίσματα name και age`

`self.name = name #στο μέλος name που προσδιορίζεται από το self καταχωρείται το όρισμα name`

`self.age = age #στο μέλος age που προσδιορίζεται από το self καταχωρείται το όρισμα age`

`p1 = Person("John", 36) #ορίζουμε το αντικείμενο p1 και καλείται ο δημιουργός της κλάσης Person για να αρχικοποιήσει τα μέλη του p1. Επειδή γράψαμε δημιουργό στην κλάση όταν τον καλούμε πρέπει να τον δίνουμε 2 ορίσματα. Αν δεν γράψαμε δημιουργό τότε θα υπήρχε ένας προεπιλεγμένος κενός δημιουργός χωρίς ορίσματα και θα τον καλούσαμε γράφοντας Person()`

`print(p1.name) → τυπώνεται η τιμή John`

`print(p1.age) → τυπώνεται η τιμή 36`

Παρατήρηση

Η παράμετρος `self` είναι μια αναφορά στο τρέχον αντικείμενο της κλάσης και χρησιμοποιείται για την προσπέλαση μελών που ανήκουν στην κλάση.

Παράδειγμα 3-Μέθοδοι Κλάσης

Τα αντικείμενα μπορεί να περιέχουν μεθόδους. Οι μέθοδοι των αντικειμένων είναι συναρτήσεις που ανήκουν στα αντικείμενα. Στο επόμενο παράδειγμα ορίζουμε μια μέθοδο στην κλάση Person που τυπώνει ένα μήνυμα. Το αντικείμενο p1 που διαθέτει αυτή τη μέθοδο τυπώνει το μήνυμα.

class Person:

def __init__(self, name, age): → στην επικεφαλίδα του δημιουργού βάζουμε τη δεσμευμένη λέξη self για να αναφερθούμε στο τρέχον στιγμιότυπο

self.name = name

self.age = age

def myfunc(self): → Η myfunc είναι μέθοδος της κλάσης Person. Ο λόγος που βάζουμε όρισμα τη λέξη self (this) είναι για να προσπελάσουμε μέσα σε αυτή το μέλος name

print("Hello my name is " + self.name)

p1 = Person("John", 36) # ορίζουμε το αντικείμενο p1 και καλείται ο δημιουργός της κλάσης Person με 2 ορίσματα για να αρχικοποιήσει τα μέλη του

p1.myfunc() → τυπώνεται Hello my name is John

Παράδειγμα 4-Η παράμετρος self

Η παράμετρος self είναι μια αναφορά στο τρέχον αντικείμενο της κλάσης και χρησιμοποιείται για την προσπέλαση μελών που ανήκουν στην κλάση. Η παράμετρος self μπορεί να έχει οποιοδήποτε όνομα, αλλά πρέπει να είναι πάντα η 1^η παράμετρος οποιασδήποτε συνάρτησης στην κλάση.

Στο επόμενο παράδειγμα που είναι ίδιο με το προηγούμενο χρησιμοποιούμε τις λέξεις myobject και abc εναλλακτικά αντί για τη λέξη self για να δείξουμε στην πράξη ότι η παράμετρος self μπορεί να έχει οποιοδήποτε όνομα.

class Person:

def __init__(myobject, name, age):

myobject.name = name

myobject.age = age

def myfunc(abc):

print("Hello my name is " + abc.name)

y =Person("John", 36)

y.myfunc() → τυπώνεται Hello my name is John

Παράδειγμα 5-Τροποποίηση Ιδιοτήτων-Μελών

Μπορούμε να τροποποιήσουμε τις ιδιότητες των αντικειμένων μιας κλάσης. Στο επόμενο παράδειγμα τροποποιούμε την ηλικία του στιγμιοτύπου p1 σε 40.

class Person:

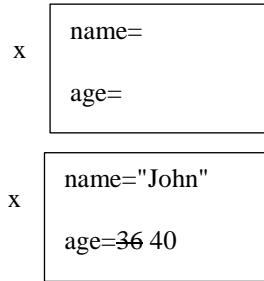
def __init__(self, name, age):

self.name = name

self.age = age

def myfunc(self):

print("Hello my name is " + self.name)



x = Person("John", 36) # εδώ καλείται αυτόματα ο δημιουργός της κλάσης δηλ. η μέθοδος __init__(self, name, age)

x.age = 40 #στο μέλος age του x καταχωρούμε το 40

print(x.age) → τυπώνεται 40

5.14 Κληρονομικότητα στην Python

Η Κληρονομικότητα μας επιτρέπει να ορίζουμε μια κλάση που κληρονομεί όλες τις μεθόδους και τις ιδιότητες της υπερκλάσης της.

- **Πατρική Κλάση (Parent class)** είναι η κλάση από την οποία γίνεται η κληρονομικότητα. Ονομάζεται και κλάση Βάσης
- **Κλάση Παιδί (Child class)** είναι η κλάση η οποία κληρονομείται από άλλη κλάση (από μια υπερκλάση). Λέγεται και Παραγόμενη κλάση.

Δημιουργία Αρχικής και Παραγόμενης Κλάσης

Ορίζουμε μια κλάση με όνομα Person με τις ιδιότητες (μέλη) firstname και lastname και τη μέθοδο printname και το δημιουργό class Person:

```
def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

def printname(self):
    print(self.firstname, self.lastname) #τυπώνονται τα μέλη firstname και lastname
```

x=Person("John", "Doe") #ορίζουμε το αντικείμενο x και καλείται ο δημιουργός της κλάσης Person. Το όρισμα John πηγαίνει στην παράμετρο fname και το όρισμα Doe πηγαίνει στην παράμετρο lname. Στο δημιουργό στο μέλος self.firstname καταχωρείται το όρισμα fname και στο μέλος self.lastname καταχωρείται το όρισμα lname

x.printname() #καλούμε τη μέθοδο printname() για το αντικείμενο x και τυπώνει John Doe

Η Κλάση Student είναι παραγόμενη της κλάσης Person και κληρονομεί τα μέλη και τις μεθόδους της κλάσης Person:

class Student(Person): #Στην παρένθεση γράφεται η υπερκλάση Person (πατρική κλάση) την οποία κληρονομεί η παραγόμενη κλάση Student (κλάση παιδί)

pass → Η λέξη pass δηλώνει ότι δεν θέλουμε να προσθέσουμε καμία άλλη ιδιότητα ή καμία άλλη μέθοδο στην κλάση Student. Αρα

η κλάση Student θα έχει τις ίδιες ιδιότητες(μέλη) και τις ίδιες μεθόδους (συναρτήσεις) με την κλάση Parent

x=Student("Mike", "Olsen") → Δηλώνουμε ένα αντικείμενο στην κλάση Student. Η κλάση Student δεν έχει δημιουργό. Επειδή η κλάση Student είναι παραγόμενη κατά τη δημιουργία ενός Student καλείται αυτόματα πρώτα ο δημιουργός της κλάσης Person. Αρχικοποιεί το μέλος firstname με την τιμή "Mike" και το μέλος lastname με την τιμή "Olsen" και αντά τα μέλη κληρονομούνται στη Student και δίνονται στο x. Η μέθοδος (συνάρτηση) printname κληρονομείται επίσης στην κλάση Student

x.printname() → Τυπώνεται το μήνυμα Mike Olsen

Η παραγόμενη κλάση Child κληρονομεί τις ιδιότητες και τις μεθόδους από την πατρική κλάση. Στο επόμενο παράδειγμα προσθέτουμε τη συνάρτηση `__init__()` στην κλάση Child αντί για το keyword pass και προσθέτουμε το μέλος `graduationyear`. Η συνάρτηση `__init__()` (δημιουργός) καλείται αυτόματα κάθε φορά που δημιουργούμε ένα αντικείμενο.

`class Person:`

```
def __init__(self, fname, lname): → δημιουργός της κλάσης Person
    self.firstname = fname
    self.lastname = lname
```

`def printname(self):` → Μέθοδος `printname` της κλάσης `Person`. Η μέθοδος αυτή κληρονομείται στην κλάση `Student`
`print(self.firstname, self.lastname)`

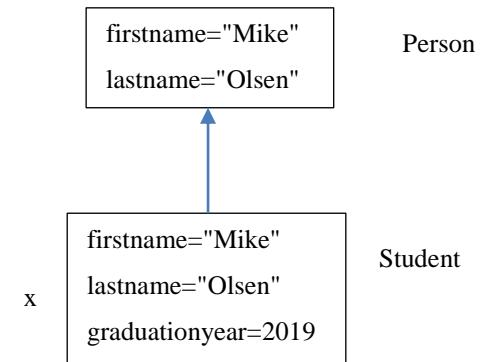
`class Student(Person):`

`def __init__(self, fname, lname):` → δημιουργός της κλάσης `Student` με

`Person.__init__(self, fname, lname)` → ο δημιουργός της κλάσης `Student` καλεί το δημιουργό της κλάσης `Person` με ορίσματα `fname, lname`

`self.graduationyear = 2019` → Το μέλος `graduationyear` της κλάσης `Student` αρχικοποιείται με την τιμή 2019. Άρα η κλάση `Student` έχει 3 μέλη: `firstname, lastname` και το `graduationyear`

`x = Student("Mike", "Olsen")` → δηλώνεται το αντικείμενο `x` τύπου `Student` και καλείται ο δημιουργός της κλάσης `Student`
`print(x.graduationyear)` → Τυπώνεται η τιμή 2019



Προσθήκη επιπλέον Μεθόδων στην Παραγόμενη Κλάση Student

`class Person:`

```
def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
```

`def printname(self):`
`print(self.firstname, self.lastname)`

`class Student(Person):`

`def __init__(self, fname, lname, year):`

`Person.__init__(self, fname, lname)` #καλείται πρώτα ο δημιουργός της υπερκλάσης για να δημιουργήσουμε πρώτα ένα `Person` από το οποίο θα κληρονομήσει το `Student`

`self.graduationyear = year`

`def welcome(self):` → Η μέθοδος `welcome` προστίθεται στην παραγόμενη κλάση `Student`
`print("Welcome", self.firstname, self.lastname, " to the class of ", self.graduationyear)`

```

x = Student("Mike", "Olsen", 2019)
x.welcome() → Τυπώνεται το μήνυμα Welcome Mike Olsen to the class of 2019
y = Student("John", "Doe", 2020)
y.welcome() → Τυπώνεται το μήνυμα Welcome John Doe to the class of 2020

```

Άλλο Παράδειγμα Κληρονομικότητας

Στο επόμενο παράδειγμα ορίζουμε την υπερκλάση Person. Διαθέτει ένα δημιουργό που αρχικοποιεί τα μέλη firstname και lastname. Επίσης διαθέτει και τη μέθοδο Name που επιστρέφει τα μέλη firstname και lastname. Στη συνέχεια ορίσουμε την παραγόμενη κλάση Employee που κληρονομεί την κλάση Person. Διαθέτει ένα δημιουργό που καλεί πρώτα το δημιουργό της υπερκλάσης Person και μετά αρχικοποιεί το μέλος staffnumber. Επίσης η κλάση Employee διαθέτει και τη μέθοδο GetEmployee που επιστρέφει το αποτέλεσμα της μεθόδου Name() και σε αυτό συνενώνει και το μέλος staffnumber.

class Person:

```

def __init__(self, first, last):
    self.firstname = first
    self.lastname = last

def Name(self):
    return self.firstname + " " + self.lastname

```

class Employee(Person):

```

def __init__(self, first, last, staffnum):
    Person.__init__(self, first, last)
    self.staffnumber = staffnum

def GetEmployee(self):
    return self.Name() + ", " + self.staffnumber

```

```

x = Person("Marge", "Simpson")
y = Employee("Homer", "Simpson", "1007")

```

```

print(x.Name()) → Τυπώνεται το μήνυμα Marge Simpson
print(y.GetEmployee()) → Τυπώνεται το μήνυμα Homer Simpson 1007

```

Παρατήρηση 1

Η μέθοδος `__init__` της (παραγόμενης) κλάσης Employee καλεί τη μέθοδο `__init__` της (υπερ) κλάσης Person. Θα μπορούσαμε εναλλακτικά να γράψουμε `super().__init__(first, last)` για να την καλέσουμε δηλαδή:

```

def __init__(self, first, last, staffnum):
    super().__init__(first, last) #καλείται πάλι ο δημιουργός της υπερκλάσης
    self.staffnumber = staffnum

```

Παρατήρηση 2

Αντί να χρησιμοποιήσουμε τις μεθόδους `Name`" and "GetEmployee" του προηγούμενου παραδείγματος θα ήταν προτιμότερο να συμπεριλάβουμε αυτή τη λειτουργικότητα στη μέθοδο `__str__` όπως δείχνουμε στη συνέχεια:

`class Person:`

`def __init__(self, first, last):`

`self.firstname = first`

`self.lastname = last`

`def __str__(self): #αντί η συνάρτηση με το εξορισμού όνομα __str__(self) επιστρέφει όλα τα μέλη συνενωμένα ως ένα string`

`return self.firstname + " " + self.lastname`

`def Name(self):`

`return self.firstname + " " + self.lastname`

`class Employee(Person):`

`def __init__(self, first, last, staffnum):`

`super().__init__(first, last)`

`self.staffnumber = staffnum`

`x = Person("Marge", "Simpson")`

`y = Employee("Homer", "Simpson", "1007")`

`print(x) → τυπώνεται το μήνυμα Marge Simpson`

`print(y) → τυπώνεται το μήνυμα Homer Simpson`

5.15 ΘΕΜΑΤΑ ΣΕ PYTHON

5.15.1 Θέμα 1^ο Ιούνιος 2021

Τί θα τυπώσει το παρακάτω πρόγραμμα Python;

L=[1, 2, [3, 4], [5, 6, 7], 8]

```
print(L[1], L[2], end=' ')
```

```
print(L[3][2])
```

Απάντηση

2 [3, 4] 7

Αιτιολόγηση

0	1	2	3	4
1	2	[3, 4]	[5, 6, 7]	8

- Το L[1] είναι το στοιχείο της λίστας L στη θέση 1 δηλ. το 2^ο στοιχείο της λίστας που είναι η τιμή 2
- Το L[2] είναι το στοιχείο της λίστας στη θέση 2 δηλ. το 3^ο στοιχείο της λίστας που είναι η υπολίστα [3, 4]
- Το L[3][2] είναι το στοιχείο της λίστας L στη θέση 3 δηλ. το 4^ο στοιχείο της λίστας που είναι η υπολίστα [5, 6, 7] από την οποία παίρνουμε το στοιχείο στη θέση 2 λαμβάνουμε το 7
- Η δήλωση `end=' '` καταργεί την αλλαγή γραμμής της εντολής `print` και τυπώνονται όλα τα αποτελέσματα στην ίδια γραμμή

5.15.2 Θέμα 8^ο Ιούνιος 2021

Τί θα τυπωθεί όταν εκτελεστούν με τη σειρά οι ακόλουθες εντολές Python;

L=[[15, [6, 76], [2, 7, 22]], 4]

```
print(L[2][2], L[1][0], L[3])
```

Απάντηση

22 6 4

Παρατήρηση

- Το L[2][2] είναι το στοιχείο της λίστας L στη θέση 2 δηλ. το 3^ο στοιχείο της λίστας που είναι η υπολίστα [2, 7, 22] από την οποία παίρνουμε το στοιχείο στη θέση 2 δηλ. το 22
- Το L[1][0] είναι το στοιχείο της λίστας L στη θέση 1 δηλ. το 2^ο στοιχείο της λίστας που είναι η υπολίστα [6, 76] από την οποία παίρνουμε το στοιχείο στη θέση 0 δηλ. το 6
- Το L[3] είναι το στοιχείο της λίστας L στη θέση 3 δηλ. το 4^ο στοιχείο της λίστας που είναι η τιμή 4

5.15.3 Θέμα 2ο Ιούνιος 2021

Τι εκτυπώνει ο επόμενος κώδικας:

```
def factorial(n):
    if (n<0 or n==0):
        return 'error'
    elif (n==1):
        return 1
    else:
        return (n*factorial(n-1))
```

```
num = -5;
print("number : ", num)
print("Factorial : ", factorial(num))
```

Απάντηση

number : -5

Factorial : error

5.15.4 Θέμα 3ο Ιούνιος 2021

Να ορίσετε μια αναδρομική συνάρτηση sum_n στην Python που θα δέχεται ως όρισμα ένα θετικό ακέραιο αριθμό N ($N>0$) και θα επιστρέψει το άθροισμα $1+2+3+\dots+(N-1)+N$. Η συνάρτηση να επιστρέψει μήνυμα σφάλματος αν ο ακέραιος είναι αρνητικός ή μηδέν. Η συνάρτηση θα διαβάζει τον ακέραιο από το όρισμα και θα τον επιστρέψει χωρίς να τον τυπώνει. Μην χρησιμοποιήστε input() και print()

Απάντηση

Ο αναδρομικός τύπος υπολογισμού του αθροίσματος $f(N)=1+2+3+\dots+(N-1)+N$ είναι $f(N)=f(N-1)+N$ με αρχική συνθήκη $f(1)=1$

```
def sum_n(n):
```

```
if (n<0 or n==0): #έλεγχος ορθών τιμών
    return 'error'
elif (n==1): #αρχική συνθήκη που τερματίζει την αναδρομή
    return 1
else:
    return n+sum_n(n-1) #αναδρομικός τύπος
```

Αν την καλέσουμε π.χ. ως sum_n(3) θα επιστρέψει την τιμή 6

Παρατήρηση

Αν ζητούσε επαναληπτική συνάρτηση για τον υπολογισμό του αθροίσματος $1+2+3+\dots+(N-1)+N$ θα γράφαμε τη εξής συνάρτηση:

```
def sum_n(n):
```

```
if (n<0 or n==0): #έλεγχος ορθών τιμών
    return 'error'
else:
    sum=0
    for i in range(1,n+1): #Κάνουμε επανάληψη από 1 μέχρι n. Η συνάρτηση range(1, n+1) είναι ισοδύναμη με την έκφραση for (i=1;i<n+1;i++)
        sum+=i #σε κάθε επανάληψη αθροίζουμε στο sum τις τιμές του i. Η εντολή είναι ισοδύναμη με το sum=sum+i
return sum
```

n=5

```
print('Αθροισμα =', sum_n(n))
```

5.15.5 Θέμα 4ο Ιούνιος 2021

Να βάλετε μια ακέραια τιμή στο s ώστε το παρακάτω πρόγραμμα να τυπώνει τη λέξη "success". Εξηγήστε σύντομα τη λειτουργία του κώδικα:

1. s=____ #βάλτε εδώ ένα ακέραιο

2. **while true:**

3. s=s-1

4. **if** s<0: **break**

5. **if** s>0:

 i. s=-s

 ii. **continue**

6. **print("success")** #ώστε να τυπώνεται αυτό

Απάντηση

- Οποιαδήποτε τιμή και να βάλουμε στο s η λέξη success θα τυπωθεί διότι π.χ.
- Αν s=0 κάνει break στην 1^η επανάληψη διότι το s θα γίνει -1 και θα τυπωθεί η λέξη success
- Αν s<0 κάνει break στην 1^η επανάληψη και θα τυπωθεί η λέξη success
- Αν s>0 στην 1^η επανάληψη γίνεται αρνητικός και στη 2^η επανάληψη με το break βγαίνει από το while και τυπώνει τη λέξη success

5.15.6 Θέμα 5ο Ιούνιος 2021-OXI

Έστω η κλάση Car η οποία διαθέτει τις μεθόδους:

__init__(self, kind, speed)

accelerating(**self**)

breaking(**self**)

Ένα παράδειγμα χρήσης της κλάσης αυτής ακολουθεί:

```
mycar = Car("Volvo",120)
```

```
print(mycar.speed) #να τυπώνεται 120
```

```
for i in range(5):
```

```
    mycar.accelerating()
```

```
print(mycar.speed) #να τυπώνεται 170
```

```
for i in range(6):
```

```
    mycar.breaking()
```

```
print(mycar.speed) #να τυπώνεται 110
```

Ζητείται με βάση τα παραπάνω να ορίσετε την κλάση Car και τις μεθόδους της

Απάντηση

class Car:

def __init__(self, kind, speed): #δήλωση δημιουργού. Είναι μια συνάρτηση που καλείται αυτόματα για ένα στιγμιότυπο και το αρχικοποιεί τη στιγμή που το δηλώνουμε

self.kind = kind #στο μέλος kind που προσδιορίζεται από το self καταχωρείται το όρισμα kind. Το όρισμα είναι η τιμή στην παρένθεση

self.speed = speed #στο μέλος speed που προσδιορίζεται από το self καταχωρείται το όρισμα speed. Το όρισμα είναι η τιμή στην παρένθεση

```
def accelerating(self):
```

```
    self.speed+=10
```

```
def breaking(self):
```

```
    self.speed-=10
```

5.15.7 Θέμα 6ο Ιούνιος 2021

Τι θα τυπωθεί όταν εκτελεστούν με τη σειρά οι ακόλουθες εντολές Python;
A=[1]

B=A

C=B

print(A, B, C)

B=[2]

print(A, B, C)

C[0]=3

print(A, B, C)

Απάντηση

[1] [1] [1]

[1] [2] [1]

[3] [2] [3]

Αιτιολόγηση

A=[1] #δημιουργείται μια λίστα με τιμή 1

B=A #Η μεταβλητή B δείχνει όπου και η A δηλ.. στην ίδια θέση μνήμης με περιεχόμενο [1] άρα οι A και B είναι η ίδια μεταβλητή (δηλ.. η ίδια λίστα)

C=B #Η μεταβλητή C δείχνει όπου και η B δηλ.. όπου και η A δηλ.. στην ίδια θέση μνήμης με περιεχόμενο [1] δηλ.. οι A, B και C είναι η ίδια μεταβλητή (δηλ.. η ίδια λίστα)

print(A, B, C) #τυπώνεται [1] [1] [1] δηλ.. η ίδια λίστα 3 φορές

B=[2] #Η μεταβλητή B γίνεται νέα λίστα με περιεχόμενο [2]

print(A, B, C) #τυπώνεται [1] [2] [1] δηλ.. 3 λίστες οι 2 με το ίδιο στοιχείο και η άλλη λίστα με την τιμή 2 διότι A και C παραμένουν οι ίδιες μεταβλητές

C[0]=3 #Στη θέση 0 της λίστας C καταχωρείται η τιμή 3. Οι μεταβλητές A και C είναι ίδιες ότι βάζουμε στη C μπαίνει και στην A

print(A, B, C) #τυπώνεται [3] [2] [3]

5.15.8 Θέμα 7ο Ιούνιος 2021

Να ορίσετε μια αναδρομική συνάρτηση στην Python που θα δέχεται ως όρισμα μία λίστα ακεραίων και θα επιστρέφει το άθροισμά τους. Η συνάρτηση να επιστρέφει 0 αν η λίστα είναι κενή.

Απάντηση

def sum_list(lis):

if lis == []:#όταν η λίστα είναι κενή επιστρέφει ως άθροισμα το 0

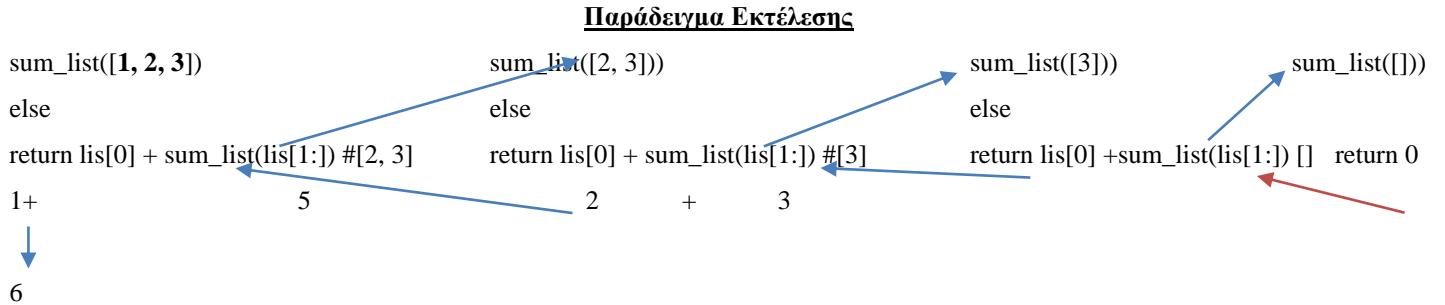
return 0

else:

return lis[0]+sum_list(lis[1:]) #στο 1^o στοιχείο της λίστας αθροίζονται όλα τα υπόλοιπα. Το lis[1:] είναι όλα τα στοιχεία της λίστας εκτός του αρχικού

```
lista=[1, 2, 3] #τα στοιχεία της λίστας  
print("Athroisma = ", sum_list(lista))
```

Ο κώδικας κατά την εκτέλεση του τυπώνει 6



5.15.9 Θέμα 9ο Ιούνιος 2021-OXI

Να περιγράψετε το αποτέλεσμα του παρακάτω κώδικα. Σε περίπτωση που δίνει σφάλμα, κάνετε τις απαραίτητες διορθώσεις `class Car():`

```
def __init__(self, number, color = "", make=""):  
    self.number = number  
    self.color = color  
    self.make = make
```

```
def __str__(self):  
    return "Αρ. κυκλοφορίας: {} – είδος {} χρώμα: {}".format(number, make, color)
```

```
c1 = Car('AXX1234', 'kókkivo', 'golf')
```

```
c2 = Car('AYY1223', 'μπλε', 'b')
```

```
for c in [c1, c2]: print(c)
```

Απάντηση

Ο διορθωμένος κώδικας είναι

```
class Car():
```

```
def __init__(self, number, color = "", make=""):
```

self.number = number
self.color = color
self.make = make

```
def __str__(self):
```

```
return "Αρ. κυκλοφορίας: {} - {} χρώμα: {}".format(self.number, self.make, self.color)
```

```
c1 = Car('AXX1234', 'κόκκινο', 'golf')
c2 = Car('AYY1223', 'μπλε', 'bmw')
```

```
for c in [c1, c2]: #σε κάθε επανάληψη μπαίνει πρώτα το c1 και μετά το c2 στο c και αυτό τυπώνεται  
    print(c)
```

5.15.10 Πρόγραμμα Python για την εκτύπωση περιττών αριθμών σε μια λίστα

Λαμβάνοντας μια λίστα αριθμών, γράψτε πρόγραμμα Python που αν εκτυπώνει όλους τους περιττούς αριθμούς στη λίστα.

Απαντήσεις

#1^{ος} Τρόπος. Εκτύπωση περιττών αριθμών σε μια λίστα με εντολή for και in

#Λίστα Αριθμών

```
list1=[10, 21, 4, 45, 66, 93]
```

for num in list1: #σε κάθε επανάληψη ένα στοιχείο της λίστας list1 καταχωρείται στο num με την εντολή in και η επανάληψη εκτελείται για το πλήθος στοιχείων της λίστας list1

if num%2==1: #αν οληθεύει η συνθήκη ο αριθμός είναι περιττός. Το % είναι το ακέραιο υπόλοιπο της διαιρεσης num/2

print(num, end = " ") #To end = " " καταργεί την αλλαγή γραμμής από την εντολή print και κάνει ένα κενό διάστημα ανάμεσα στις τιμές που τυπώνονται

Εξοδος Προγράμματος: [21 45 93]

#2^{ος} Τρόπος. Εκτύπωση περιττών αριθμών σε μια λίστα με εντολή for και range

#Λίστα Αριθμών

```
list1=[10, 21, 4, 45, 66, 93]
```

for i in range(len(list1)): #Η συνάρτηση len υπολογίζει το πλήθος τιμών της λίστας που είναι 6. Η συνάρτηση range(6) αφορά τις τιμές στο διάστημα [0, 6) και τις καταχωρεί στο i άρα στο i δίνονται οι τιμές 0, 1, 2, 3, 4 και 5

if list1[i]%2==1: #το στοιχείο list1[i] είναι περιττός

print(list1[i], end = " ") #To end = " " καταργεί την αλλαγή γραμμής από την print

Εξοδος Προγράμματος: [21 45 93]

#3^{ος} Τρόπος. Εκτύπωση περιττών αριθμών σε μια λίστα με εντολή while

#Λίστα Αριθμών

```
list1=[10, 21, 4, 45, 66, 93]
```

i=0 #μια λίστα αριθμείται από το μηδέν στην Python

while(i<len(list1)): #Η len υπολογίζει το πλήθος στοιχείων της λίστας list1 που είναι 6

if list1[i]%2== 1:

print(list1[i], end = " ")

i+=1#i=i+1. Στην εντολή while το i δεν αυξάνεται κατά 1 από μόνο του όπως στο for, το κάνουμε εμείς

Εξοδος Προγράμματος: [21 45 93]

#4^{ος} Τρόπος. Εκτύπωση περιττών αριθμών σε μια λίστα με List Comprehension:

```
list1= [10, 21, 4, 45, 66, 93]
```

only_odd=[**num for num in list1 if num%2==1**] #πρώτα γίνεται η εντολή for num in list1 όπου κάθε τιμή από τη list1 καταχωρείται στο num. Μετά ελέγχεται η συνθήκη if num%2==1. Όσες τιμές ικανοποιούν τη συνθήκη δηλ. είναι περιττές καταχωρούνται μέσω της μεταβλητής num σε μια νέα λίστα με όνομα only_odd δηλ. only_odd=[num]

print(only_odd)#εδώ εκτυπώνεται η λίστα με τους περιττούς

Εξοδος Προγράμματος: 21 45 93

#5^{ος} Τρόπος. Εκτύπωση περιττών αριθμών σε μια λίστα με lambda expressions:

```
list1=[10, 21, 4, 45, 66, 93, 11]
```

only_odd=list(filter(lambda x: (x%2==1), list1)) #Ορίζεται η συνάρτηση (lambda x: (x % 2==1)) που ελέγχει αν ένας αριθμός είναι περιττός. Η συνάρτηση filter επιλέγει τα στοιχεία της list1 που ικανοποιούν τη συνθήκη και η συνάρτηση list δημιουργεί μια νέα λίστα με τα αποτελέσματα της filter δηλ. με τους περιττούς αριθμούς και το όνομα της νέας λίστας είναι only_odd

```
print("Odd numbers in the list: ", only_odd) #εδώ εκτυπώνεται η λίστα με τους περιττούς
```

Έξοδος Προγράμματος: Odd numbers in the list: [21, 45, 93, 11]

5.15.11 Θέμα 4α Ομάδα Α Ιούνιος 2019

Εστω το παρακάτω πρόγραμμα Python που τυπώνει τους περιττούς αριθμούς από μια δοσμένη λίστα αριθμών χρησιμοποιώντας εκφράσεις από λογισμό λ:

```
list1 = [10, 21, 4, 45, 66, 93, 11]
odd_nos = list(filter(lambda x: (x % 2 != 0), list1))
print("Odd numbers in the list: ", odd_nos)
```

Ξαναγράψτε το πρόγραμμα με ένα δεύτερο τρόπο κάνοντας χρήση μόνο εντολών επανάληψης/ελέγχου, συμπληρώνοντας στη διακεκομένη γραμμή.

```
#Python program to print odd Numbers in a List
```

```
#list of numbers:
```

```
list1 = [10, 21, 4, 45, 66, 93, 11]
```

```
only_odd=[_____]
```

```
print(only_odd)
```

Απάντηση

```
only_odd = [num for num in list1 if num % 2 ==
1]
```

5.15.12 Πρόγραμμα Python για εναλλαγή του πρώτου και του τελευταίου στοιχείου σε μια λίστα

Δοθείσης μιας λίστας να γραφεί πρόγραμμα σε Python για εναλλαγή του πρώτου και του τελευταίου στοιχείου σε μια λίστα

Παράδειγμα

#1^{ος} Τρόπος. Υπολογίζουμε το μήκος της λίστας και εναλλάσσουμε το πρώτο με το τελευταίο στοιχείο της λίστας

`def swapList(newList): #με την εντολή def ορίζουμε συνάρτηση. To swapList είναι το όνομα της συνάρτησης και το newList είναι το όρισμα της συνάρτησης`

`temp = newList[0]`

`newList[0]=newList[len(newList)-1] #To len είναι έτοιμη συνάρτηση που υπολογίζει το πλήθος στοιχείων μας λίστας. To newList[0] είναι το αρχικό στοιχείο της λίστας και το newList[len(newList)-1] είναι το τελευταίο στοιχείο της λίστας στη θέση len(newList)-1`

`newList[len(newList)-1] = temp`

`return newList #επιστρέφει τη λίστα με την εναλλαγή`

#Κώδικας main

`newList=[12, 35, 9, 56, 24]`

`print(swapList(newList)) #Η συνάρτηση swapList καλείται σε εντολή print διότι επιστρέφει αποτέλεσμα. To αποτέλεσμα είναι η νέα λίστα με την εναλλαγή`

Έξοδος Προγράμματος: [24, 35, 9, 56, 12]

#2^{ος} Τρόπος. Εναλλάσσουμε το πρώτο με το τελευταίο στοιχείο της λίστας

`def swapList(newList):`

`temp=newList[0]`

`newList[0]=newList[-1] #to newList[-1] αναφέρεται στο τελευταίο στοιχείο της λίστας`

`newList[-1] = temp`

`return newList #επιστρέφει τη λίστα με την εναλλαγή`

#Κώδικας main

`newList = [12, 35, 9, 56, 24]`

`print(swapList(newList))`

Έξοδος Προγράμματος: [24, 35, 9, 56, 12]

#3^{ος} Τρόπος. Εναλλάσσουμε το πρώτο με το τελευταίο στοιχείο της λίστας

Το τελευταίο στοιχείο της λίστας αναφέρεται ως `list[-1]`. Γιαυτό μπορούμε εναλλακτικά να εναλλάξουμε τα στοιχεία `list[0]` και `list[-1]`

`def swapList(newList):`

`newList[0], newList[-1]=newList[-1], newList[0]`

`return newList`

#Κώδικας main

`newList = [12, 35, 9, 56, 24]`

`print(swapList(newList))`

Έξοδος Προγράμματος: [24, 35, 9, 56, 12]

5.15.13 Πρόγραμμα Python που ελέγχει την ύπαρξη στοιχείου σε μια λίστα

#1^{ος} Τρόπος.

Κώδικας Python για τον έλεγχο ύπαρξης στοιχείου με εντολές for και in

Αρχικοποίηση Λίστας

```
test_list = [1, 6, 3, 5, 3, 4]
```

#Έλεγχος αν το 4 υπάρχει στη λίστα με επανάληψη

```
for i in test_list:
```

```
    if(i == 4):
```

```
        print("Element Exists")
```

#2^{ος} Τρόπος.

Έλεγχος αν το 4 υπάρχει στη λίστα με εντολή in

if (4 in test_list): //ελέγχεται αυτόματα όλη η λίστα χωρίς εντολή επανάληψης. Συγκεκριμένα συγκρίνει κάθε στοιχείο με το 4 και ελέγχει αν είναι ίσο με το 4

```
print("Element Exists")
```

5.15.14 Πρόγραμμα Python για Αντιστροφή Λίστας

Η Python προσφέρει διάφορους τρόπους για την αντιστροφή μιας λίστας

#1^{ος} Τρόπος. Με χρήση της έτοιμης συνάρτησης reversed()

#Αντιστροφή λίστας με χρήση της reversed()

```
def Reverse_list(lst):
```

return [ele for ele in reversed(lst)] #το for εφαρμόζεται σε κάθε στοιχείο της λίστας. Το κάθε στοιχείο της λίστας τοποθετείται στη μεταβλητή ele. Μετα δημιουργείται μια νέα λίστα με τα στοιχεία ele. Η συνάρτηση reversed το τοποθετεί το κάθε ele στο τέλος της λίστας ώστε να αντιστραφεί η λίστα

Κώδικας στο main

```
lst = [10, 11, 12, 13, 14, 15]
```

```
print(Reverse_list(lst))
```

Εξοδος Προγράμματος: [15, 14, 13, 12, 11, 10]

#2^{ος} Τρόπος. Με χρήση της εξορισμού συνάρτησης reverse()

Αντιστροφή λίστας με τη συνάρτηση reverse()

```
def Reverse_list (lst):#το lst είναι μια τοπική μεταβλητή. Λαμβάνει το listI από το main
```

lst.reverse() #Χρησιμοποιώντας τη μέθοδο reverse() μπορούμε να αντιστρέψουμε τα περιεχόμενα της λίστας.

return lst #η αντεστραμμένη λίστα επιστρέφει πίσω

```
list1 = [10, 11, 12, 13, 14, 15]
```

```
print(Reverse_list (lst))
```

Εξοδος Προγράμματος: [15, 14, 13, 12, 11, 10]

#3^{ος} Τρόπος. Με χρήση του τελεστή ::-1

```
lst = [10, 11, 12, 13, 14, 15]
```

```
lst=lst[::-1] #παίρνει όλη την λίστα από το τελενταίο στοιχείο μέχρι το πρώτο και τα καταχωρεί πάλι στη λίστα lst
```

```
print(lst)
```

Εξοδος Προγράμματος: [15, 14, 13, 12, 11, 10]

5.15.15 Πρόγραμμα Python για μέτρηση Άρτιων και Περιττών σε μια Λίστα

Δοθείστης μιας λίστας αριθμών να γραφεί πρόγραμμα σε Python που μετρά πλήθος Άρτιων και Περιττών στοιχείων σε μια λίστα.

Input: list1 = [2, 7, 5, 64, 14]

Output: Even = 3, odd = 2

#1^{ος} Τρόπος. Υπολογισμός πλήθους Άρτιων και Περιττών με εντολή for

#Λίστα Αριθμών

```
list1 = [10, 21, 4, 45, 66, 93, 1]
even_count, odd_count = 0, 0 #ταυτόχρονη αρχικοποίηση μετρητών
for num in list1: #σε κάθε επανάληψη μπαίνει μια τιμή από το list1 στο num
    if num%2 == 0:
        even_count +=1 #αύξηση μετρητή άρτιων
    else:
        odd_count +=1 αύξηση μετρητή περιττών

print("Even numbers in the list: ", even_count)
print("Odd numbers in the list: ", odd_count)
```

Έξοδος Προγράμματος

Even numbers in the list: 3

Odd numbers in the list: 4

#2^{ος} Τρόπος. Υπολογισμός πλήθους Άρτιων και Περιττών με εντολή while

#Λίστα Αριθμών

```
list1 = [10, 21, 4, 45, 66, 93, 11]
even_count, odd_count = 0, 0
i=0
while(i<len(list1)):
    if list1[i] % 2 == 0:
        even_count += 1
    else:
        odd_count +=1

    i+=1 #αύξηση μετρητή επαναλήψεων. Αυτό γίνεται αυτόματα στο for
```

print("Even numbers in the list: ", even_count)

print("Odd numbers in the list: ", odd_count)

Έξοδος Προγράμματος

Even numbers in the list: 3

Odd numbers in the list: 4

#3^{ος} Τρόπος. Υπολογισμός πλήθους Αρτιων και Περιττών με Εκφράσεις Lambda

#Λίστα Αριθμών

```
list1 = [10, 21, 4, 45, 66, 93, 11]
```

odd_count = len(list(filter(lambda x:(x%2==1), list1))) #Το 1^ο βήμα είναι ο ορισμός της έκφρασης lambda που θέτει τη συνθήκη επιλογής περιττών αριθμών. Το 2^ο βήμα είναι η εφαρμογή της συνάρτησης filter που εφαρμόζει την έκφραση lambda στα στοιχεία της list1 και επιλέγει αυτά που την ικανοποιούν. Το 3^ο βήμα είναι η εκτέλεση της συνάρτησης len που δημιουργεί μια λίστα με τα αποτελέσματα της filter. Το 4^ο βήμα είναι η εκτέλεση της συνάρτησης len που υπολογίζει το πλήθος στοιχείων της προκύπτουσας λίστας περιττών

even_count = len(list(filter(lambda x: (x%2 ==0), list1))) #δημιουργείται μια λίστα με τους άρτιους και με τη συνάρτηση len υπολογίζεται το πλήθος της

```
print("Even numbers in the list: ", even_count)
```

```
print("Odd numbers in the list: ", odd_count)
```

Έξοδος Προγράμματος

Even numbers in the list: 3

Odd numbers in the list: 4

#4^{ος} Τρόπος. Υπολογισμός πλήθους Αρτιων και Περιττών με List Comprehension

#Λίστα αριθμών

```
list1 = [10, 21, 4, 45, 66, 93, 11]
```

only_odd = [num for num in list1 if num% 2==1] # Το 1^ο βήμα είναι η εκτέλεση της εντολής for που σαρώνει τα στοιχεία της list1. Το 2^ο βήμα είναι η συνθήκη if num% 2==1 που επιλέγει τα περιττά στοιχεία της list1. Το 3^ο βήμα είναι η καταχώριση κάθε περιττού στοιχείου στη μεταβλητή num και το 4^ο βήμα είναι η δημιουργία μιας νέας λίστας που υποδηλώνεται με τις αγκύλες στην οποία τοποθετούνται τα περιττά στοιχεία

odd_count = len(only_odd) //με τη συνάρτηση len υπολογίζεται το πλήθος των περιττών της λίστας only_odd και αποθηκεύεται στη μεταβλητή odd_count

```
print("Even numbers in the list: ", len(list1)-odd_count)
```

```
print("Odd numbers in the list: ", odd_count)
```

Έξοδος Προγράμματος

Even numbers in the list: 3

Odd numbers in the list: 4

5.15.16 Πρόγραμμα Python για εκτύπωση θετικών αριθμών και μηδενικών σε μια Λίστα

Input: list1 = [12, -7, 5, 64, -14]

Output: 12, 5, 64

#1^{ος} Τρόπος. Εκτύπωση θετικών αριθμών λίστας με εντολή for

```
#Λίστα Αριθμών  
list1 = [11, -21, 0, 45, 66, -93]  
for num in list1:  
    if num>=0:  
        print(num, end = " ")
```

Έξοδος Προγράμματος [11 0 45 66](#)

#2^{ος} Τρόπος. Εκτύπωση θετικών αριθμών λίστας με εντολή while

```
#Λίστα Αριθμών  
list1= [-10, 21, -4, -45, -66, 93]  
num=0  
while(num<len(list1)):  
    if list1[num]>= 0:  
        print(list1[num], end = " ")  
    num += 1 #στην εντολή while πρέπει εμείς να αυξάνουμε το μετρητή επαναλήψεων
```

Έξοδος Προγράμματος: [21 93](#)

#3^{ος} Τρόπος. Εκτύπωση θετικών αριθμών λίστας με List Comprehension

```
list1 = [-10, -21, -4, 45, -66, 93]  
pos_nos = [num for num in list1 if num >= 0]  
print("Positive numbers in the list: ", pos_nos)
```

Έξοδος Προγράμματος

Positive numbers in the list: 45 93

#4^{ος} Τρόπος. Εκτύπωση θετικών αριθμών λίστας με Lambda Εκφράσεις

```
#Λίστα Αριθμών  
list1 = [-10, 21, 4, -45, -66, 93, -11]  
  
pos_nos = list(filter(lambda x: (x >= 0), list1))  
print("Positive numbers in the list: ", pos_nos)
```

Έξοδος Προγράμματος Positive numbers in the list: [21, 4, 93](#)

5.15.17 Πρόγραμμα Python που μετρά το πλήθος θετικών και αρνητικών σε μια Λίστα

Παράδειγμα

Input: list1 = [2, -7, 5, -64, -14]

Output: pos = 2, neg = 3

#1^{ος} Τρόπος. Εκτύπωση πλήθους θετικών και αρνητικών στοιχείων λίστας με εντολή for

#Λίστα αριθμών

list1 = [10, -21, 4, -45, 66, -93, 1]

pos_count, neg_count = 0, 0

for num **in** list1:

if num >= 0:

 pos_count += 1

else:

 neg_count += 1

print("Positive numbers in the list: ", pos_count)

print("Negative numbers in the list: ", neg_count)

Έξοδος Προγράμματος

Positive numbers in the list: 4

Negative numbers in the list: 3

#2^{ος} Τρόπος. Εκτύπωση πλήθους θετικών και αρνητικών στοιχείων λίστας με εντολή while

Λίστα αριθμών

list1 = [-10, -21, -4, -45, -66, 93, 11]

pos_count, neg_count = 0, 0

num = 0

while(num < **len**(list1)):

if list1[num] >= 0:

 pos_count += 1

else:

 neg_count += 1

 num += 1

print("Positive numbers in the list: ", pos_count)

print("Negative numbers in the list: ", neg_count)

Έξοδος Προγράμματος

Positive numbers in the list: 2

Negative numbers in the list: 5

#3^{ος} Τρόπος. Εκτύπωση πλήθους θετικών και αρνητικών στοιχείων λίστας με εκφράσεις Lambda

Λίστα Αριθμών

list1 = [10, -21, -4, 45, 66, 93, -11]

neg_count = **len(list(filter(lambda x: (x < 0), list1)))**

pos_count = **len(list(filter(lambda x: (x >= 0), list1)))**

print("Positive numbers in the list: ", pos_count)

print("Negative numbers in the list: ", neg_count)

Έξοδος Προγράμματος

Positive numbers in the list: 4

Negative numbers in the list: 3

#4^{ος} Τρόπος. Εκτύπωση πλήθους θετικών και αρνητικών στοιχείων λίστας με List Comprehension

#Λίστα Αριθμών

list1 = [-10, -21, -4, -45, -66, -93, 11]

only_pos = [num **for** num **in** list1 **if** num >= 1]

pos_count = **len(only_pos)**

print("Positive numbers in the list: ", pos_count)

print("Negative numbers in the list: ", len(list1) - pos_count)

Έξοδος Προγράμματος

Positive numbers in the list: 1

Negative numbers in the list: 6

5.15.18 Πρόγραμμα Python για τη διαγραφή της N-οστής εμφάνισης μιας δοθείσας λέξης σε μια λίστα - ΟΧΙ

Παραδείγματα:

Input: list=["geeks", "for", "geeks"]

word = geeks, N = 2

Output: list =["geeks", "for"]

Input: list =["can", "you", "can", "a", "can" "?"]

word = can, N = 1

Output: list = ["you", "can", "a", "can" "?"]

#1^{ος} Τρόπος. Λαμβάνοντας μια άλλη λίστα

#Ορίζουμε μια νέα λίστα π.χ. με όνομα newList. Εκτελούμε επανάληψη στα στοιχεία της λίστας και ελέγχουμε αν η λέξη που θα διαγραφεί ταιριάζει με το στοιχείο και τον αριθμό εμφάνισης, διαφορετικά συνενώνουμε (append) το στοιχείο στη newList.

#Συνάρτηση διαγραφής i-οστής λέξης

def RemoveIthWord(lst, word, N):

```
newList = []
count = 0
#επανάληψη στα στοιχεία της λίστας
for i in lst:
    if(i == word):
        count = count + 1 #υπολογίζουμε το πλήθος εμφανίσεων της λέξης
    if(count != N): #αν η εμφάνιση της λέξης δεν είναι η N-οστή
        newList.append(i) #προσθέτουμε τη λέξη αυτή με τη συνάρτηση append σε μια νέα λίστα με όνομα newList
    else: # if(i != word):
        newList.append(i) #αν η λέξη δεν είναι η ζητούμενη την προσθέτουμε με τη συνάρτηση append σε μια νέα λίστα
lst = newList
if count == 0:
    print("Item not found")
else:
    print("Updated list is: ", lst)
return newList

list = ["geeks", "for", "geeks"]
word = "geeks"
N = 2
RemoveIthWord(list, word, N)
```

Έξοδος Προγράμματος

Updated list is: ['geeks', 'for']

#2^{ος} Τρόπος. Αφαίρεση στοιχείου απευθείας από τη λίστα

Αντί να δημιουργήσουμε μια νέα λίστα, διαγράφουμε το στοιχείο αντιστοίχισης από την ίδια τη λίστα. Κάνουμε επανάληψη στα στοιχεία της λίστας και ελέγχουμε αν η λέξη που πρέπει να αφαιρεθεί ταιριάζει με το στοιχείο και τον αριθμό εμφάνισης, Εάν ναι διαγράφουμε αυτό το στοιχείο και επιστρέφουμε αληθές. Αν επιστρέψουμε το True, εκτυπώνουμε τη λίστα διαφορετικά, εκτυπώνουμε το μήνυμα "Item not Found".

Function για διαγραφή της j-οστής εμφάνισης

```
def RemoveNthWord(list, word, j): #το j υποδηλώνει την εμφάνιση της λέξης που θα διαγράψουμε
    count = 0
    for i in range(0, len(list)): #ισοδύναμα for i in range(len(list)):
        if (list[i] == word):
            count = count + 1
        if(count == j):
            del(list[i]) #η έτοιμη συνάρτηση del διαγράφει το στοιχείο από τη θέση i
            return True #επιστρέφεται η λέξη True ως ένδειξη ότι η λέξη διαγράφηκε

    return False #στο τέλος του for εκτελείται η εντολή return False ως ένδειξη ότι η λέξη δεν διαγράφηκε
```

list = ['geeks', 'for', 'geeks']

word = 'geeks'

j= 2

result = RemoveNthWord (list, word, j)

if (result ==True):

print("Updated list is: ", list)

else:

print("Item not Updated")

Updated list is: ['geeks', 'for']

5.15.19 Πρόγραμμα Python για τη διαγραφή πολλαπλών στοιχείων από μια λίστα -ΟΧΙ

Δοθείσης μιας λίστας αριθμών, να γραφεί πρόγραμμα Python για τη διαγραφή πολλαπλών στοιχείων από μια λίστα βασιζόμενοι σε μια συνθήκη

#1^{ος} Τρόπος: Θέλουμε να διαγράψουμε κάθε άρτιο στοιχείο μιας λίστας

#Λίστα Αριθμών

```
list1 = [11, 5, 17, 18, 23, 50]
```

#εκτέλεση επανάληψης στα στοιχεία της λίστας και πρόσθεση τους στη μεταβλητή total

```
for ele in list1:
```

```
    if ele % 2 == 0:
```

```
        list1.remove(ele)
```

#εκτύπωση τροποποιημένης λίστας

```
print("New list after removing all even numbers: ", list1)
```

Έξοδος Προγράμματος

New list after removing all even numbers: [11, 5, 17, 23]

#2^{ος} Τρόπος: Χρησιμοποίηση list comprehension

Διαγραφή όλων των άρτιων περιλαμβάνοντας τα στοιχεία που δεν είναι άρτια (δηλ. διατηρώντας τα περιττά στοιχεία).

δημιουργία λίστας

```
list1 = [11, 5, 17, 18, 23, 50]
```

#δημιουργία νέας λίστας που διαγράφει όλα τα άρτια στοιχεία

```
list1 = [elem for elem in list1 if elem % 2 == 1]
```

```
print(list1)
```

Έξοδος Προγράμματος

11 5 17 23

#3^{ος} Τρόπος: Διαγραφή συνεχόμενων στοιχείων χρησιμοποιώντας list slicing. Ο επόμενος κώδικας Python διαγράφει τα στοιχεία με δείκτες από 1 μέχρι 4

δημιουργία λίστας

```
list1 = [11, 5, 17, 18, 23, 50]
```

#διαγραφή στοιχείων από θέση 1 μέχρι 4 π.χ. τα στοιχεία 5, 17, 18, 23 θα διαγραφούν

```
del list1[1:5]
```

```
print(list1)
```

Έξοδος Προγράμματος

11 50

#4ος Τρόπος. Χρησιμοποιώντας list comprehension

Ας υποθέσουμε ότι τα στοιχεία που πρέπει να διαγραφούν είναι γνωστά αντί των δεικτών αυτών των στοιχείων. Σε αυτή την περίπτωση, μπορούμε να εξαλείψουμε άμεσα αυτά τα στοιχεία χωρίς να μας ενδιαφέρουν οι δείκτες

δημιουργία λίστας

```
list1 = [11, 5, 17, 18, 23, 50]
```

#στοιχεία προς διαγραφή

```
unwanted_num = {11, 5}
```

```
list1 = [ele for ele in list1 if ele not in unwanted_num]
```

#εκτύπωση τροποποιημένης λίστας

```
print("New list after removing unwanted numbers: ", list1)
```

Έξοδος Προγράμματος

New list after removing unwanted numbers: [17, 18, 23, 50]

5.15.20 Θέμα 4β Ομάδα Α Ιούνιος 2019

Έστω το παρακάτω πρόγραμμα Python που έχει ως σκοπό να διαγράψει από δοσμένη λίστα όλους τους άρτιους αριθμούς. Συμπληρώστε στο παρακάτω πρόγραμμα την κενή γραμμή _____ :

#Python program to remove multiple elements from a list

```
list1 = [11, 5, 17, 18, 23, 50]
```

#iterate each element in list

#and add them in variable total

```
for ele in list1:
```

if ele % 2 == 0:

#printing modified list

```
print("New list after removing all even numbers: ", list1)
```

Απάντηση

#creating a list

```
list1 = [11, 5, 17, 18, 23, 50]
```

#iterate each element in list

#and add them in variable total

```
for ele in list1:
```

if ele % 2 == 0:

list1.remove(ele)

#printing modified list

```
print("New list after removing all even numbers: ", list1)
```

Εξοδος Προγράμματος

New list after removing all even numbers: [11, 5, 17, 23]

5.15.21 Πρόγραμμα Python για έλεγχο αν ένα string είναι παλίνδρομο ή όχι

Δοθέντος ενός string, να γραφεί πρόγραμμα Python που να ελέγχει αν είναι παλίνδρομο ή όχι. Ένα string ονομάζεται παλίνδρομο αν το αντεστραμμένο string είναι ίδιο με το αρχικό string.

#1^{ος} Τρόπος: Βρίσκουμε το αντεστραμμένο string και εξετάζουμε αν το αντεστραμμένο και το κανονικό string είναι ίδια

```
def reverse_string(s): #συνάρτηση που αντιστρέφει ένα string
```

```
    return s[::-1] #επιστρέφει την αντεστραμμένη λίστα
```

```
def isPalindrome(s):
```

```
    rev=reverse_string (s) #Κλήση της συνάρτησης reverse
```

```
    if (s == rev): # ελέγχουμε αν τα δύο string είναι ίδια ή όχι δηλαδή συγκρίνουμε το αλφαριθμητικό με το αντεστραμμένο του
```

```
        return True
```

```
    return False
```

```
# main
```

```
s = "Anna"
```

```
ans = isPalindrome(s)
```

```
if ans == 1:
```

```
    print("Yes")
```

```
else:
```

```
    print("No")
```

Έξοδος Προγράμματος

Yes

#2ος Τρόπος: Με Σύγκριση των αντίστοιχων χαρακτήρων

Εκτελούμε το βρόχο από την αρχή μέχρι το μήκος/2 και ελέγχουμε τον πρώτο χαρακτήρα με το τελευταίο χαρακτήρα της συμβολοσειράς, το δεύτερο στο δεύτερο τελευταίο και ούτω καθεξής Εάν ο χαρακτήρας δεν ταιριάζει, η συμβολοσειρά δεν θα είναι παλίνδρομη. Παρακάτω είναι η εφαρμογή αυτής της προσέγγισης:

```
def isPalindrome(str):
```

```
    #εκτέλεση επανάληψης από το 0 to len/2
```

```
    for i in range(0, len(str)/2):
```

```
        if str[i]!= str[len(str)-1-i]:
```

```
            return False
```

```
    return True #επιστρέφεται στο τέλος του for
```

```
#main function
```

```
s = "Anna"
```

```
ans = isPalindrome(s)
```

```
if ans == 1:
```

```
    print("Yes")
```

```
else:
```

```
    print("No")
```

Έξοδος Προγράμματος

Yes

5.15.22 Πρόγραμμα Python εφαρμογής της Δυαδικής Αναζήτησης (Binary Search) -OXI

#1^{ος} Τρόπος. Με Αναδρομική Συνάρτηση

#Επιστρέφει τη θέση του x στο arr αν το x υπάρχει στο arr αλλιώς επιστρέφει -1

```
def binarySearch (arr, l, r, x):
```

```
    if r >= l:
```

```
        mid = l + (r - l)/2
```

```
        if arr[mid] == x:
```

```
            return mid
```

```
        elif arr[mid] > x:
```

```
            return binarySearch(arr, l, mid-1, x)
```

```
        else:
```

```
            return binarySearch(arr, mid+1, r, x)
```

```
    else:
```

```
        return -1
```

#Αρχικός Πίνακας

```
arr = [ 2, 3, 4, 10, 40 ]
```

```
x = 10
```

#Κλήση Συνάρτησης

```
result = binarySearch(arr, 0, len(arr)-1, x)
```

```
if result != -1:
```

```
    print "Element is present at index %d" % result
```

```
else:
```

```
    print "Element is not present in array"
```

Έξοδος Προγράμματος

Element is present at index 3

#2^{ος} Τρόπος. Με Επαναληπτική Συνάρτηση

```
def binarySearch(arr, l, r, x):
```

```
    while l <= r:
```

```
        mid = l + (r - l)/2;
```

```
        if arr[mid] == x:
```

```
            return mid
```

```
        elif arr[mid] < x:
```

```
            l = mid + 1
```

```
        else:
```

```
            r = mid - 1
```

```
    return -1
```

Λίστα αριθμών

```
arr = [ 2, 3, 4, 10, 40 ]
```

```
x = 10
```

```
# Κλήση Συνάρτησης
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Έξοδος Προγράμματος

Element is present at index 3

5.15.23 Πρόγραμμα Python για τη μέτρηση εμφανίσεων τιμών σε μια λίστα χρησιμοποιώντας λεξικά

Διθείσης μιας μη ταξινομημένης λίστας στοιχείων (που μπορεί να είναι ακέραιοι ή όχι) βρείτε τη συγχρόνη εμφάνισης κάθε ξεχωριστού στοιχείου στη λίστα χρησιμοποιώντας ένα λεξικό (dictionary)

Είσοδος: [1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]

Έξοδος:

1 : 5
2 : 4
3 : 3
4 : 3
5 : 2

#1ος Τρόπος. Με Συνάρτηση

```
def CountFrequency(my_list):  
    freq = {} #Δημιουργία κενού dictionary  
    for item in my_list: #σε κάθε επανάληψη ένα στοιχείο της λίστας τοποθετείται στο item  
        if (item in freq): #ελέγχεται αν η τιμή είναι στο λεξικό δηλ. ελέγχεται αν η τιμή έχει ξαναδοθεί  
            freq[item] += 1 #πάμε στο λεξικό freq στη θέση item και αυξάνουμε τον αντίστοιχο μετρητή. Π.χ. αν η τιμή είναι το 1  
            και έχει ξαναδοθεί (δηλ. την έχουμε ξαναβάλει στο λεξικό freq) τότε κάνουμε το εξής: freq[1] += 1  
        else:  
            freq[item] = 1  
  
    for key, value in freq.items(): #αντό το for είναι μέσα στη συνάρτηση και τυπώνει τα αποτελέσματα  
        print("%d : %d" %(key, value))  
  
if __name__ == "__main__": #αυτή η συνθήκη υποδηλώνει ότι είμαστε στο "main" δηλ. εκτός συνάρτησης  
    my_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]  
    CountFrequency(my_list)
```

Έξοδος Προγράμματος

1 : 5
2 : 4
3 : 3
4 : 3
5 : 2

#2ος Τρόπος. Με Dictionary

```
def CountFrequency(my_list):  
    freq = {} #Δημιουργία κενού dictionary  
    for items in my_list:  
        freq[items] = my_list.count(items) #με τη συνάρτηση count υπολογίζεται το πλήθος εμφανίσεων του items και τοποθετείται  
        στο λεξικό στη θέση items μπαίνει η τιμή ως το κλειδί και στο freq[items] ο μετρητής εμφανίσεων ως τιμή  
  
    for key, value in freq.items():  
        print("% d : % d"%(key, value))
```

```
if __name__ == "__main__":
    my_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]
    CountFrequency(my_list)
```

Έξοδος Προγράμματος

1 : 5
2 : 4
3 : 3
4 : 3
5 : 2

5.15.24 Θέμα 4 Ομάδα Β Ιούνιος 2019

Έστω το παρακάτω πρόγραμμα Python που μετράει τη συχνότητα εμφάνισης ενός στοιχείου λίστας (list element) κάνοντας χρήση δομής dictionary. Συμπληρώστε, στο παρακάτω πρόγραμμα, τις κενές γραμμές _____:

```
def CountFrequency(my_list):
```

```
    freq = {} #Creating an empty dictionary
```

```
    for item ____ my_list:
```

```
        if (item ____ freq):
```

```
            freq[item] += 1
```

```
        else:
```

```
            freq[item] = 1
```

```
    for ___, value in ____:
```

```
        print ("% d : % d"%(key, ____))
```

```
#Driver function
```

```
if __name__ == "__main__":
```

```
    my_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]
```

```
    CountFrequency(my_list)
```

Απάντηση

```
def CountFrequency(my_list):
```

```
    freq = {} #Creating an empty dictionary
```

```
    for item in my_list:
```

```
        if (item in freq):
```

freq[item] += 1 #Μέσα στο dictionary με όνομα freq καταχωρώ για κάθε στοιχείο της λίστας ένα μετρητή εμφανίσεων του. Για παρόδειγμα αν στη λίστα συναντήσω την τιμή 5 για πρώτη φορά τότε freq[5]=1. Αν ξανασυναντήσω το 5 τότε freq[5]=2 κ.λ.π.

```
        else:
```

```
            freq[item] = 1
```

```
    for key, value in freq.items(): //η συνάρτηση items() επιστρέφει το πλήθος στοιχείων ενός dictionary
```

```
        print("% d : % d"%(key, value))
```

```
#Creating an empty dictionary
```

```
if __name__ == "__main__":
```

```
    my_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2, 2]
```

```
    CountFrequency(my_list)
```

5.15.25 Πρόγραμμα Python για τον υπολογισμό του αθροίσματος όλων των στοιχείων ενός λεξικού

Παραδείγματα

Input : {'a': 100, 'b':200, 'c':300}

Output : 600

#1^{ος} Τρόπος. Με επανάληψη

```
def returnSum(myDict):
    sum = 0
    for i in myDict:
        sum = sum + myDict[i] #αθροίζονται τα values των κλειδιών myDict[i]
    return sum
```

dict = {'a': 100, 'b':200, 'c':300}

print("Sum:", returnSum(dict))

Έξοδος Προγράμματος Sum : 600

#2^{ος} Τρόπος. Με εντολή for για εκτέλεση επανάληψης στα στοιχεία με τη συνάρτηση values()

```
def returnSum(dict):
    sum = 0
    for i in dict.values(): #η συνάρτηση values() επιστρέφει μόνο τις τιμές των κλειδιών (values)
        sum = sum+i #αθροίζονται τα values των κλειδιών myDict[i]
    return sum
```

dict = {'a': 100, 'b':200, 'c':300} #κάθε στοιχείο ενός λεξικού είναι ένα ζεύγος της μορφής {key, value}

print("Sum :", returnSum(dict))

Έξοδος Προγράμματος Sum : 600

5.15.26 Θέμα 5α Σεπτέμβριος 2019

Έστω το παρακάτω πρόγραμμα Python που ταξινομεί αλφαριθμητικά τις λέξεις (words) μια μπρος μια που δίνει ο χρήστης. Συμπληρώστε στο παρακάτω πρόγραμμα τις κενές γραμμές _____

my_str=input("Enter a string")

#breakdown the string into a list of words

words=my_str._____

#sort the list

words._____

#display the sorted words

print("The sorted words are:")

for word _____ words

 print(word)

Απάντηση

```

my_str=input("Enter a string")

#breakdown the string into a list of words. Οι λέξεις αναγνωρίζονται με βάση το χαρακτήρα κενού και τοποθετούνται στη λίστα word
words=my_str.split()

#sort the list
words.sort()

#display the sorted words
print("The sorted words are:")

for word in words:
    print(word)

```

5.15.27 Θέμα 2α Ιούνιος 2020

Συμπληρώστε τον κώδικα ώστε να δίνει στην έξοδο: 123

```

def f(x):
    for a in x:
        .....
f([1, 2, 3])

```

- a) `print(a+1, end = ' ')`
- β) `print(a, end = '\n')`
- γ) `print(a, end = ' ')`
- δ) `print(123)`
- ε) `print([1, 2, 3])`

Απάντηση

γ) `print(a, end = ' ')`

Παρατήρηση

```

def f(x):
    for a in x: #σε κάθε επανάληψη στο a μπαίνει ένα στοιχείο της λίστας x
        print(a, end=' ')
f([1,2,3])

```

5.15.28 Θέμα 2β Ιούνιος 2020

Ποιο το αποτέλεσμα του προγράμματος:

```

a = 8.0
b = (8/a)/(a-8) + a**8 / (a **7)

```

- α) 8.0
- β) ZeroDivisionError
- γ) 8E8
- δ) 8
- ε) 1.142857

Απάντηση

ZeroDivisionError

Παρατήρηση

Όταν συμβαίνει σφάλμα κατά την εκτέλεση του κώδικα η Python εμφανίζει εξαιρέσεις όπως και η Java που είναι μηνύματα κατά την εκτέλεση του κώδικα σχετικών με το σφάλμα που συνέβη

5.15.29 Θέμα 3 Ιούνιος 2020

Ποιο θα είναι το αποτέλεσμα του κώδικα:

```
a = [[1, 2, 3], [4, 5, 6, 7]]  
print(len(a[1][1:]))
```

- α) 1 2 3
- β) 1 2 3 4 5 6 7
- γ) 4 5 6
- δ) 3

Απάντηση

δ) 3

Παρατήρηση

Η συνάρτηση `len(a[1][1:])` πηγαίνει στο 2^o στοιχείο της λίστας που είναι στη θέση 1 (και είναι μια υπολίστα) και υπολογίζει το πλήθος των στοιχείων της από το 2^o μέχρι και το τελευταίο

Αν θέλαμε και το πλήθος στοιχείων στην 1^η υπολίστα θα γράφαμε `len(a[0][1:])`

5.15.30 Θέμα 4 Ιούνιος 2020

Τι θα επιστρέψει ο παρακάτω κώδικας:

```
def f(x,y):  
    return x + y  
print(type(f(2, 3)))
```

- α) <class 'NoneType'>
- β) <class 'int'>
- γ) 5
- δ) <class 'function'>

Απάντηση

β) <class 'int'>

Παρατήρηση

Η έτοιμη συνάρτηση `type` στην Python επιστρέφει τον τύπο ενός αντικειμένου. Εδώ το επιστρεφόμενο αντικείμενο από τη συνάρτηση `f` είναι το άθροισμα των ακέραιων μεταβλητών `x`, `y` άρα ο τύπος του αποτελέσματος είναι <class 'int'>

5.15.31 Θέμα 1 Σεπτέμβριος 2020

Τι θα τυπώσει ο παρακάτω κώδικας Python

```
i = 3
while i > 0:
    i = i + 1
    if i < 5: continue
    if i > 10: break
    print(i, end = ' ')
```

- a) 6 7 8
- β) 6 7 8 9
- γ) 5 6 7 8 9 10
- δ) 5 6 7 8

Απάντηση

γ) 5 6 7 8 9 10

Παρατήρηση

Για i=4 κάνει continue στην επόμενη επανάληψη του while απευθείας

Για i=5 τυπώνει 5

Για i=6 τυπώνει 6

Για i=7 τυπώνει 7

Για i=8 τυπώνει 8

Για i=9 τυπώνει 9

Για i=10 τυπώνει 10

Για i=11 κάνει break

5.15.32 Θέμα 2 Σεπτέμβριος 2020

Ποιο θα είναι το αποτέλεσμα:

```
for x in range(12, 2, -2):
    print(x+1, end = ' ')
    if x == 9: break
```

- α) 13 11 9 7 5
- β) 12 10 8
- γ) -10 -12 -14

Απάντηση

α) 13 11 9 7 5

Παρατήρηση

Το x παίρνει τις τιμές 12, 10, 8, 6, 4 και τυπώνει 13, 11, 9, 7, 5

Η επανάληψη για x=2 ΔΕΝ εκτελείται

5.15.33 Θέμα 3 Σεπτέμβριος 2020

Ποιο θα είναι το αποτέλεσμα της εκτέλεσης του κώδικα Python

x = 4

```
for x in "1234":  
    print (x*x, end = ' ')
```

- a) 1 3 6 9
- β) 1 33 55 9999
- γ) 11 33 66 99
- δ) 1
- ε) TypeError

Απάντηση

ε) TypeError

Παρατήρηση

Η εκτέλεση εμφανίζει το ακόλουθο μήνυμα:

Traceback (most recent call last):

File "C:/Users/jiane/AppData/Local/Programs/Python/Python39/1.py", line 4, in <module>

```
print (x*x, end = ' ')
```

TypeError: can't multiply sequence by non-int of type 'str'

Προσοχή

Ο επόμενος κώδικας είναι σωστός

x = 4

```
for x in "1234":
```

```
    print (x*4, end = ' ') #o κάθε χαρακτήρας που πάει στο x τυπώνεται 4 φορές
```

και τυπώνει:

1111 2222 3333 4444

5.15.34 Θέμα 3 Σεπτέμβριος 2020

Ποιο το αποτέλεσμα

b = [[1, 2, 3, 4], [5, 6, 7]]

```
print (len (b[1] [1:]))
```

- α) 2
- β) 3
- γ) 7
- δ) 4

Απάντηση

a) 2

Παρατήρηση

Με τη ένδειξη `b[1]` αρχίζουμε από το 2^o στοιχείο της λίστας `b` (διότι τα στοιχεία μιας λίστα αριθμούνται από το μηδέν) που είναι η υπολίστα [5, 6, 7] και με το συμβολισμό [1:] αναφερόμαστε σε όλα τα στοιχεία της από το 2^o μέχρι και το τελευταίο συμπεριλαμβανομένου και του τελευταίου στοιχείου. Η συνάρτηση `len` υπολογίζει το πλήθος σε αυτό που αναφέραμε που είναι 2 το 6 και το 7.

5.15.35 Θέμα 4 και 5 Σεπτέμβριος 2020

Έστω το λεξικό `d`. Ποιο το αποτέλεσμα της εντολής: `print(len(d.values()))`

```
d = {1:'Δε', 2:'Τρ', 3: 'Τε', 4:'Πε', 5:'Πα', 6:'Σα', 7:'Κυ'}  
print(len(d.values()))
```

- a) 7
- β) 7 7
- γ) 7 7 7
- δ) NameError

Απάντηση

a) 7

Παρατήρηση

- Η συνάρτηση `keys()` επιστρέφει μόνο τα κλειδιά ενός λεξικού
- Η συνάρτηση `values()` επιστρέφει μόνο τις τιμές ενός λεξικού
- Η συνάρτηση `items()` επιστρέφει όλα τα στοιχεία ενός λεξικού δηλ. τα ζεύγη `key, values`
- Η συνάρτηση `len(values())` επιστρέφει το πλήθος των τιμών ενός λεξικού

5.15.36 Θέμα 6 Σεπτέμβριος 2020

Ποιο είναι το αποτέλεσμα της εκτέλεσης του κώδικα Python

```
b=1  
def x(n):  
    b=5  
    print(b, n, end = ' ')  
    return b  
print(x(2), b)
```

- α) 1 2 5 5
- β) 5 3 1 1
- γ) 5 2 5 1
- δ) 5 2 1 5

Απάντηση

γ) 5 2 5 1

Παρατήρηση

- Αρχικά γίνεται η κλήση της `x(2)`
- Το 2 μεταβιβάζεται στο `n`.

- Στην τοπική μεταβλητή b καταχωρείται η τιμή 5. Με την εντολή b=5 ορίζουμε μια νέα τοπική μεταβλητή b μέσα στη συνάρτηση που δεν έχει σχέση με τη μεταβλητή b του main
- Τυπώνεται η τιμή 5 για το b, η τιμή 2 για το n μέσα στη συνάρτηση
- Με το **end = ''** καταργείται η αλλαγή γραμμής από το print
- με το **return** b επιστρέφεται το τοπικό b=5 στην εντολή κλήσης x(2) ως αποτέλεσμα και τυπώνεται 5 ως επιστρεφόμενο αποτέλεσμα
- μετά τυπώνεται το b του main που είναι 1

5.15.37 Θέμα με Dictionary

Ποιο το αποτέλεσμα του παρακάτω κώδικα Python

```
d={1: 'a', 2: 'v', 3: 't', 4: 'e'}
```

```
print(sorted (d, key = d.get))
```

```
for i in sorted (d, key = d.get):
```

```
    print(d[i], end = ' ')
```

else:

```
    print ('γεια')
```

Απάντηση

[1, 4, 2, 3]

α ε ν τ γεια

Παρατήρηση

- Η συνάρτηση sorted σε ένα λεξικό το ταξινομεί σε αύξουσα σειρά με βάσει τις τιμές του. Η αλφαριθμητική σειρά των τιμών του είναι **α ε ν τ και τα αντίστοιχα κλειδιά είναι 1, 4, 2, 3**. Η ένδειξη key = d.get **επιστρέφει κλειδί (key)**
- Στην εντολή **print(sorted (d, key = d.get))** τυπώνονται MONO τα κλειδιά που αντιστοιχούν στις ταξινομημένες τιμές του λεξικού δηλ. τα κλειδιά με τη σειρά 1 (α), 4 (ε), 2 (ν) και 3 (τ)
- Στο **for i in sorted (d, key = d.get):** στο i μπαίνουν κατά σειρά οι τιμές 1, 4, 2, 3 και τυπώνονται οι τιμές d[1]=α, d[4]=ε, d[2]=ν και d[3]=τ
- Όταν το for τελειώσει εκτελείται το else και τυπώνει το αλφαριθμητικό 'γεια'

5.15.38 Θέμα 5 Σεπτέμβριος 2021 με Dictionary

Τι θα εκτυπώσει ο επόμενος κώδικας;

```
d={1:'γ', 2:'ε', 3:'ν', 4:'α'}
```

```
for i in sorted(d, key=d.get):
```

```
    print(d[i], end="")
```

else:

```
    print('σας')
```

Απάντηση

αγεισας

Παρατήρηση

- Η συνάρτηση sorted σε ένα λεξικό το ταξινομεί σε αύξουσα σειρά με βάσει τις τιμές του. Η αλφαριθμητική σειρά των τιμών του είναι **α γ ε ι και τα αντίστοιχα κλειδιά είναι 4, 1, 2 , 3**. Η ένδειξη key = d.get **επιστρέφει κλειδί (key)**
- Στο **for i in sorted (d, key = d.get)**: στο i μπαίνουν κατά σειρά οι τιμές 4, 1, 2, 3 και τυπώνονται οι τιμές d[4]=α, d[1]=γ, d[2]=ε και d[3]=ι
- Όταν το for τελειώσει εκτελείται το else και τυπώνει το αλφαριθμητικό 'γεια'

5.15.39 Θέμα 7 Σεπτέμβριος 2020 με Dictionary

Ποιο το αποτέλεσμα του παρακάτω κώδικα Python

```
d={1: 'α', 2: 'ν', 3: 'τ', 4: 'ε'}
for i in sorted (d, key = d.get):
    print(d[i], end = ' ')
else: print ('γεια')
```

- a) αντε γεια
 β) αεντ γεια
 γ) αεντγεια
 δ) ν α ε τ γεια

Απάντηση

γ) α ε ν τ γεια

Παρατήρηση

- Η συνάρτηση sorted σε ένα λεξικό **ταξινομεί σε αύξουσα σειρά τις τιμές του** (values). Η αλφαριθμητική σειρά των τιμών του είναι **α ε ν τ** και τα αντίστοιχα κλειδιά είναι **1, 4, 2, 3**. Η καταχώριση key= d.get **επιστρέφει το κλειδί (key) και το καταχωρεί στη μεταβλητή key**
- Στο **for i in sorted (d, key = d.get)**: στο i μπαίνουν κατά σειρά τα κλειδιά **1, 4, 2, 3** όπως αντιστοιχούν στις τιμές **α ε ν τ** και τυπώνονται οι τιμές d[1]=**α**, d[4]=**ε**, d[2]=**ν** και d[3]=**τ**

5.15.40 Θέμα 1 Σεπτέμβριος 2021 με Dictionary

Ποιο το αποτέλεσμα του παρακάτω κώδικα Python:

d={1:'α', 2:'ν', 3:'υ', 4:'ι', 5:'ο'}

for i in sorted(d, key=d.get):

print(d[i], end='')

else: print('σου')

Απάντηση

αινουσου

Παρατήρηση

- Η συνάρτηση sorted σε ένα λεξικό ταξινομεί σε ανέουσα σειρά τις τιμές του (values). Η αλφαριθμητική σειρά των τιμών του είναι **a i v o u** και τα αντίστοιχα κλειδιά είναι **1, 4, 2, 5, 3**. Η καταχώριση key=d.get επιστρέφει το κλειδί (key) και το καταχωρεί στη μεταβλητή key
- Στο for i in sorted (d, key = d.get): στο i μπαίνουν κατά σειρά τα κλειδιά **1, 4, 2, 5, 3** όπως αντίστοιχούν στις τιμές **a i v o u** και τυπώνονται οι τιμές d[1]=**a**, d[4]=**i**, d[2]=**v**, d[5]=**o** και d[3]=**u**
- Όταν το for τελειώσει εκτελείται το else και τυπώνει το αλφαριθμητικό 'σου'

5.15.41 Θέμα 3 Σεπτέμβριος 2021 με Dictionary

Έστω λεξικό: d={1:'Δε', 2:'Τρ', 3:'Τε', 4:'Πε', 5:'Πα', 6:'Σα', 7:'Κυ'}. Ποιο το αποτέλεσμα της εντολής: print(len(d.values()))

Απάντηση

7

Παρατήρηση

Η d.values() επιστρέφει τις τιμές του λεξικού και η συνάρτηση len υπολογίζει το πλήθος των τιμών που είναι 7.

5.15.42 Θέμα 1 Φεβρουάριος 2021

Ποιο θα είναι το αποτέλεσμα του παρακάτω κώδικα;

'c' + 'd' if '356'.isdigit() else 'y1'+'b1'

- cdb1
- cd
- cdb1
- cdy1b1

Απάντηση

cd

Παρατήρηση 1

Ελέγχεται ο κάθε γαρακτήρας του αλφαριθμητικού '356' αν είναι αριθμητικός με την εξορισμού συνάρτηση isdigit() και επειδή όλοι οι γαρακτήρες του αλφαριθμητικού '356' είναι όντως αριθμητικοί τότε στο αλφαριθμητικό 'c' + συνενώνεται το αλφαριθμητικό 'd' και εμφανίζεται το cd

Παρατήρηση 2

'c' + 'd' if '35b'.isdigit() else 'y1'+'b1'

Ελέγχεται ο κάθε γαρακτήρας του αλφαριθμητικού '35b' αν είναι αριθμητικός με την εξορισμού συνάρτηση isdigit() και εφόσον όλοι οι γαρακτήρες του αλφαριθμητικού είναι αριθμητικοί (κάτι που είναι ψευδές) τότε στο αλφαριθμητικό 'y1' + συνενώνεται το αλφαριθμητικό 'b1' και εμφανίζεται το y1b1

Παρατήρηση 3



5.15.43 Θέμα 2 Φεβρουάριος 2021

Ποια από τις παρακάτω μεθόδους ΔΕΝ μπορούμε να χρησιμοποιήσουμε για να κάνουμε διάτρεξη των στοιχείων σε ένα λεξικό

- filter()
- cycle()
- iter()
- map()
- chain()

Απάντηση

filter()

Παρατήρηση

Παράδειγμα 1

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

mapped_values = **map(lambda x: x ** 2, my_dict.values())** #Η συνάρτηση map() εφαρμόζει σε κάθε τιμή του λεξικού που επιστρέφεται από τη συνάρτηση my_dict.values μια lambda συνάρτηση. Στο x μπαίνει η κάθε τιμή του λεξικού και υπολογίζεται το τετράγωνο κάθε τιμής του λεξικού και αποθηκεύεται κάθε αποτέλεσμα στη μεταβλητή mapped_values που λειτουργεί ως λίστα

```
for value in mapped_values:
```

```
    print(value)
```

1

4

9

Παράδειγμα 2

```
from itertools import cycle
```

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

cycle_iterator=**cycle(my_dict.items())** #μέσω της συνάρτησης my_dict.items() παίρνουμε όλα τα στοιχεία του λεξικού και μέσω της συνάρτησης cycle() δημιουργούμε μια κυκλική λίστα και τα αποθηκεύουμε. Το όνομα της είναι cycle_iterator

```
print(next(cycle_iterator)) # Output: ('a', 1)
```

```
print(next(cycle_iterator)) # Output: ('b', 2)
```

```
print(next(cycle_iterator)) # Output: ('c', 3)
```

```
print(next(cycle_iterator)) # Output: ('a', 1) επαναφορά από την αρχή
```

('a', 1)

('b', 2)

('c', 3)

('a', 1)

Παράδειγμα 3

```
from itertools import chain
my_dict = {'a': 1, 'b': 2, 'c': 3}
chained_values = chain(my_dict.values()) #μέσω της συνάρτησης my_dict.values() παίρνουμε τις τιμές του λεξικού και μέσω της συνάρτησης chain() δημιουργούμε μια λίστα και τις αποθηκεύουμε. Το όνομα της είναι chained_values
```

for value in chained_values:

```
    print(value)
```

1

2

3

Παράδειγμα 4

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
dict_iterator=iter(my_dict)
```

```
print(next(dict_iterator)) # Output: 'a'  
print(next(dict_iterator)) # Output: 'b'  
print(next(dict_iterator)) # Output: 'c'
```

a

b

c

```
dict_iterator = iter(my_dict.items())
```

('a', 1)

('b', 2)

('c', 3)

```
dict_iterator = iter(my_dict.values())
```

1

2

3

Παράδειγμα 5

```
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

filtered_dict = dict(filter(lambda item: item[1] % 2 == 0, my_dict.items())) #με τη συνάρτηση my_dict.items() λαμβάνουμε όλα τα στοιχεία του λεξικού. Τα τοποθετούμε ένα προς ένα στη μεταβλητή item. Στη συνέχεια με το item[1] ελέγχουμε το 2^o μέλος του

ζεύγους (key, value) δηλ. το value. Αν το value κάθε τιμής του λεξικού είναι άρτια τότε επιλέγεται από τη συνάρτηση filter. Μετά καλείται ο δημιουργός dict() και φτιάχνει ένα νέο λεξικό με τα στοιχεία του my_dict που έχουν άρτια value

```
print(filtered_dict) # Output: {'b': 2, 'd': 4}terator)) # Output: 'c'
```

5.15.44 Θέμα 1 Σεπτέμβριος 2021

Ποιο είναι το αποτέλεσμα του παρακάτω κώδικα:

```
d_dict={'color': 'yellow', 'fruit': 'banana', 'pet': 'cat'}
```

for key **in** d_dict:

```
    print(key)
```

- 'color'
- 'fruit'
- 'pet'
- color
- fruit
- pet
- yellow
- banana
- cat
- 'yellow'
- 'banana'
- 'cat'

Απάντηση

- **color**
- **fruit**
- **pet**

Παρατήρηση

Με το επόμενο for τυπώνονται μόνο τα keys (κλειδιά) του λεξικού χωρίς τις αποστρόφους' ' Ακόμα και σε διπλές αποστρόφους να ήταν τα κλειδιά τυπώνονται χωρίς αυτές

```
d_dict={'color': 'yellow', 'fruit': 'banana', 'pet': 'cat'}
```

for key **in** d_dict:

```
    print(key)
```

- **color**
- **fruit**
- **pet**

Παράδειγμα

```
mydict = { "brand": "Ford", "model": "Mustang", "year": 1964 }
```

```
for x in mydict:
```

```
    print(x) #τυπώνονται μόνο τα κλειδιά (keys) χωρίς αποστρόφους
```

brand

model

year

Με το επόμενο for τυπώνονται οι τιμές (values) του λεξικού χωρίς τις ''

```
for key in d_dict:
```

```
    print(d_dict[key])
```

Ford

Mustang

1964

Ενναλακτικά μπορούμε να τυπώσουμε τις τιμές (values) του λεξικού χωρίς τις '' με τον επόμενο κώδικα:

```
for key in d_dict.values():
```

```
    print(key)
```

yellow

banana

cat

Αν θέλουμε να τυλώσουμε και τα κλειδιά και τις τιμές του λεξικού (key. Values) τότε εφαρμόζουμε τον επόμενο κώδικα:

```
for x, y in d_dict.items(): #στο x καταχωρείται σε κάθε επανάληψη ένα κλειδί (key) και στο y καταχωρείται η αντίστοιχη τιμή του (value)
```

```
print(x, y) #τυπώνονται ταντόχρονα και τα κλειδιά και οι τιμές
```

color yellow

fruit banana

pet cat

Αν γράψουμε την εντολή:

```
print(d_dict)
```

τυπώνεται το λεξικό με τη μορφή των { } δηλ.

```
{'color': 'yellow', 'fruit': 'banana', 'pet': 'cat'}
```

5.15.45 Θέμα 2 Σεπτέμβριος 2021

Ποιο είναι το αποτέλεσμα του παρακάτω κώδικα:

```
{'a', 'n', 's'} & set('fup')  
• { 'b', 'r', 'a'}  
• set()  
• {}  
• {'q', 'r', 'x', 'u', 'b', 'a'}
```

Απάντηση

set()

5.15.46 Θέμα 3 Σεπτέμβριος 2021

Η συγκεκριμένη δήλωση θα εμφανίσει μήνυμα λάθους ή όχι:

```
d={'a1':0, 'b1':1, 'c1':0}  
if d['a1']>0:
```

```
    print('ΣΩΣΤΑ')
```

```
elif d['b1']>0:
```

```
    print('ΣΩΣΤΑ')
```

```
elif d['c1']>0:
```

```
    print('ΣΩΣΤΑ')
```

```
elif d['d1']>0:
```

```
    print('ΣΩΣΤΑ')
```

```
else:
```

```
    print('ΛΑΘΟΣ')
```

Απάντηση

Τυπώνει το μήνυμα 'ΣΩΣΤΑ' και δεν εμφανίζει μήνυμα λάθους

Παρατήρηση

Ο λόγος που τυπώνεται η τιμή ΣΩΣΤΑ είναι διότι αληθεύει η 2^n συνθήκη:

```
elif d['b1']>0:
```

```
    print('ΣΩΣΤΑ')
```

όπου η τιμή του λεξικού στη θέση 'b1' είναι 1>0 άρα τυπώνεται το μήνυμα ΣΩΣΤΑ

5.15.47 Θέμα 4 Σεπτέμβριος 2021

Ποιο είναι το αποτέλεσμα της εκτέλεσης του κώδικα Python;

(*lambda x:(x+3)*5/2(3)*)

- 0
- 15.0
- 30.0
- SyntaxError
- Κανένα από τα παραπάνω

Απάντηση

15.0

Παρατήρηση

Καλείται η ανώνυμη συνάρτηση με όρισμα 3 και αυτό μπαίνει στο x και γίνεται η πράξη $(3+3)*5/2=15.0$

5.15.48 Θέμα 6 Σεπτέμβριος 2021

Ποιο θα είναι το αποτέλεσμα του παρακάτω κώδικα:

for i in [2, 1]:

 for j in [1, 3]:

 if i+j>3: break

 print(i-1, end=")

 print(i-1, end="")

Απάντηση

100

Παρατήρηση

- i=2
- j=1
- 1+2>3 OXI
- j=3
- 2+3>3 NAI
- Τερματίζει το εσωτερικό for
- **Τυπώνει 1**
- i=1
- j=1
- 1+1>3 OXI
- j=3
- 1+3>3 NAI
- Τερματίζει το εσωτερικό for
- **Τυπώνει 0**
- Τερματίζει το εξωτερικό for
- **Τυπώνει 0**

5.15.49 Θέμα 7 Σεπτέμβριος 2021

Ποιο θα είναι το αποτέλεσμα του παρακάτω κώδικα Python:

`for u in range(5):`

```
    if u%3==1: pass  
    else: print(u//2, end=' ')
```

- 0 1 1
- 0 1 2 3 4
- 1 2 3 4 5
- 0 1 2 3 4 5
- 0 1 2 3

Απάντηση

0 1 1

Παρατήρηση

- $u=0$
- $0 \% 3 == 1$ OXI. Το $\%$ είναι ο τελεστής του ακέραιου υπολοίπου
- **Τυπώνει $0//2=0$ διότι το ακέραιο πηλίκο είναι 0. Ο Τελεστής $//$ υπολογίζει το πηλίκο με στρογγυλοποίηση στον πλησιέστερο ακέραιο προς τα κάτω**
- $u=1$
- $1 \% 3 == 1$ NAI. To pass είναι η κενή εντολή στην Python. Σημαίνει do nothing
- $u=2$
- $2 \% 3 == 1$ OXI
- **Τυπώνει $2//2=1$ διότι το ακέραιο πηλίκο είναι 1**
- $u=3$
- $3 \% 3 == 1$ OXI
- **Τυπώνει $3//2=1.5 \rightarrow 1$ διότι υπολογίζει το πηλίκο με στρογγυλοποίηση στον πλησιέστερο ακέραιο προς τα κάτω που είναι το 1**
- $u=4$
- $4 \% 3 == 1$ NAI. To pass είναι η κενή εντολή στην Python. Σημαίνει do nothing

Παρατήρηση

- `print(3//2) →` υπολογίζει το ακέραιο πηλίκο με στρογγυλοποίηση προς τα κάτω και τυπώνει 1
- `print(3/2) →` υπολογίζει το πραγματικό πηλίκο και τυπώνει 1.5
- `print(3%2) →` υπολογίζει το ακέραιο υπόλοιπο και τυπώνει 1

5.15.50 Θέμα 7 Σεπτέμβριος 2021

Ποιο θα είναι το αποτέλεσμα της εντολής: $a=15 // (7\%4)$

- 5
- 8.5
- 8.0
- 8

Απάντηση

5

Παρατήρηση

- Πρώτα υπολογίζεται το υπόλοιπο $7 \% 4 = 3$
- Μετά υπολογίζεται το ακέραιο πηλίκο: $a=15 // 3 = 5$

5.15.51 Θέμα 7 Σεπτέμβριος 2021

Τι θα τυπώσει το παρακάτω πρόγραμμα:

```
def f():
    a=1
    print(a, end=' ')
a=5
f()
print(a, end=' ')
```

- 5 1
- 1
- 1 5
- 1
- UnboundLocalError

Απάντηση

1 5

Παρατήρηση

- Η εκτέλεση αρχίζει από την εντολή $a=5$. Μετά καλείται η $f()$. Ορίζεται ένα τοπικό a με τιμή 1 και τυπώνεται το τοπικό a με τιμή 1.
- Τελειώνει η $f()$ και επιστρέφει πίσω και μετά τυπώνεται το a του $main$ που έχει την τιμή 5.

5.15.52 Θέμα 11 Σεπτέμβριος 2021

Τι θα τυπώσει το παρακάτω πρόγραμμα:

```
def f():
    global a
    a=a+1
    print(a, end=' ')
a=1
f()
print(a, end=' ')
```

- 1 2
- 2 1
- 1 1
- 2 2
- UnboundLocalError

Απάντηση

2 2

Παρατήρηση

- Η εκτέλεση αρχίζει από την εντολή `a=1`. Μετά καλείται η `f()`.
- Το `global a` δηλώνει ότι το `a` μέσα το σώμα της συνάρτησης δεν είναι μια τοπική μεταβλητή της συνάρτησης αλλά είναι το **που έχει δηλωθεί στο main**
- Το `a` αυξάνεται κατά 1 στην εντολή `a=a+1` και γίνεται 2 και τυπώνεται το καθολικό `a` με τιμή 2
- Τελειώνει η `f()` και επιστρέφει πίσω και μετά τυπώνεται το καθολικό `a` του `main` που έχει την τιμή 2

5.15.53 Θέμα 8 Σεπτέμβριος 2021

Ποιο είναι το αποτέλεσμα της εκτέλεσης του κώδικα Python:

```
i=2
for i in "1234":
    print(i*i, end=' ')
```

- 1 4 9 16
- 1 22 333 4444
- 11 22 33 44
- 4
- TypeError

Απάντηση

- TypeError

Παρατήρηση

Traceback (most recent call last):

```
File "C:/Users/USER/AppData/Local/Programs/Python/Python310/ex10.py", line 3, in <module>
    print(i*i, end=' ')
TypeError: can't multiply sequence by non-int of type 'str'
```

5.15.54 Θέμα 9 Σεπτέμβριος 2021

Ποιο θα είναι το αποτέλεσμα του παρακάτω κώδικα;

```
a=[[1, 2, 3], [4, 5, 6, 7]]
```

```
print(len(a[1][1:]))
```

- 3
- 3.0
- 456
- 234
- 4

Απάντηση

3

5.15.55 Θέμα 10 Σεπτέμβριος 2021

Συμπληρώστε τον κώδικα ώστε να δίνει στην έξοδο 123:

```
def f(x):
```

```
    for a in x:  
        .....
```

```
f([1,2,3])
```

Απάντηση

```
def f(x):
```

```
    for a in x:  
        print(a, end="")
```

```
f([1,2,3])
```

5.15.56 Θέμα 12 Σεπτέμβριος 2021

Ποιο θα είναι το αποτέλεσμα της εκτέλεσης του κώδικα Python:

```
def f(x):
```

```
    for a in x:  
        print(a, end=' ')  
    return a
```

```
f([1, 2, 4])
```

- [1, 2, 4]
- 1 2 4
- 1
- [1]

- 124

Απάντηση

1

Παρατίθηση

- Καλείται η `f([1, 2, 4])` και η λίστα μεταβιβάζεται στο `x`
- Εκτελείται η επανάληψη `for a in x` εκτελείται για κάθε στοιχείο της λίστας. Στο `a` μπαίνει αρχικά το 1
- Με την εντολή `print(a, end='')` τυπώνεται η τιμή 1
- Μετά εκτελείται η εντολή `return` a η οποία τερματίζει τη συνάρτηση και επιστρέφει την τιμή `a=1` στο `main`
- Η τιμή αυτή στο `main` χάνεται και δεν τυπώνεται κάτι άλλο

5.15.57 Θέμα 13 Σεπτέμβριος 2021

Τι θα τυπώσει το παρακάτω πρόγραμμα:

```
def f1():
    fruit='μήλο'
    f2()

def f2():
    print(fruit)
```

fruit='αχλάδι'

f1()

- Μήλο
- Αχλάδι
- Error
- Μήλο Αχλάδι

Απάντηση

- **Αχλάδι**

Παρατίθηση

- Η μεταβλητή στο `main` `fruit` αρχικοποιείται με το string '`αχλάδι`' δηλ. `fruit='αχλάδι'`
- Καλείται η `f1()`
- Ορίζεται η τοπική μεταβλητή με τιμή `='μήλο'`
- Η `f1()` καλεί την `f2()`
- Στη συνάρτηση `f2()` τυπώνεται με την εντολή `print(fruit)` η καθολική μεταβλητή `fruit` διότι στην `f2()` ΔΕΝ ορίζεται μεταβλητή `fruit` άρα η μεταβλητή `fruit` που τυπώνεται είναι η καθολική

5.15.58 Θέμα 14 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα εκτέλεσης του παρακάτω κώδικα:

a=8.0

b=(8/a)/(a-8)+a**8/(a**7)

- 8.0
- ZeroDivisionError
- 8E8
- 8
- 1.142857

Απάντηση

- **ZeroDivisionError**

Παρατήρηση

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

 b=(8/a)/(a-8)+a**8/(a**7)

ZeroDivisionError: float division by zero

5.15.59 Θέμα 15 Σεπτέμβριος 2021

Τι θα τυπώσει το παρακάτω πρόγραμμα:

s='Καλή σας μέρα'

print(s.capitalize())

- Καλή σας μέρα
- Καλή σας Μέρα
- ΚΑΛΗ ΣΑΣ ΜΕΡΑ
- StringError

Απάντηση

- **Καλή σας μέρα**

Παρατήρηση

- Η συνάρτηση **capitalize()** επιστρέφει ένα αντίγραφο του αρχικού string με κεφαλαίο μόνο τον αρχικό γαρακτήρα του string
- Η συνάρτηση **upper()** επιστρέφει ένα αντίγραφο του αρχικού string με κεφαλαίους ΟΛΟΥΣ τους γαρακτήρες του string
- Η συνάρτηση **lower()** επιστρέφει ένα αντίγραφο του αρχικού string με πεζούς ΟΛΟΥΣ τους γαρακτήρες του string

5.15.60 Θέμα 16 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα εκτέλεσης του παρακάτω κώδικα Python:

`for i in [2, 1]:`

`for j in [1, 3]:`

`if i+j>3: break`

`print (i+j, end='')`

`print(i-1, end='')`

`print(i-1, end='')`

- 3 0 0
- 3 0
- 3 5 1 2 4 0
- 3 5 1 2 4 0 0
- 3 1 2 0
- 3 1 2 0 0

Απάντηση

3 1 2 0 0

Παρατήρηση

- Στο εξωτερικό for το `i=2`
- Στο εσωτερικό for το `j=1`
- Η συνθήκη `2+1>3` είναι ψευδής
- **Εκτελείται η εντολή `print (i+j, end='')` και τυπώνει 3**
- Στο εσωτερικό for το `j=3`
- Η συνθήκη `2+5>3` είναι αληθής και κάνει `break` από το for `j`
- **Εκτελείται η εντολή `print (i-1, end='')` στο for `i` και τυπώνει 1**
- Στο εξωτερικό for το `i=1`
- Στο εσωτερικό for το `j=1`
- Η συνθήκη `1+1>3` είναι ψευδής
- **Εκτελείται η εντολή `print (i+j, end='')` και τυπώνει 2**
- Στο εσωτερικό for το `j=3`
- Η συνθήκη `1+3>3` είναι αληθής και κάνει `break` από το for `j`
- **Εκτελείται η εντολή `print (i-1, end='')` στο for `i` και τυπώνει 0**
- Τελειώνει το εξωτερικό for `i`

- Στο main **εκτελείται η εντολή *print* (i-1, end=' ') και τυπώνει**

5.15.61 Θέμα 1 Σεπτέμβριος 2021

Συμπληρώστε τον κώδικα ώστε να δίνει στην έξοδο: **1234**

```
def f(x):
    for a in x:
        .....
f([1,2, 3, 4])
```

Απάντηση

```
def f(x):
    for a in x:
        print(a, end="")
f([1,2, 3, 4])
```

Παρατήρηση

Με την εντολή *print(a, end="")* τυπώνονται στην οθόνη 1234 χωρίς κενό λόγω του *end=""* διότι η λίστα μεταβιβάζεται στο x. Με την επανάληψη *for a in x:* στο a μπαίνει σε κάθε επανάληψη μια τιμή της λίστας x και τυπώνεται

5.15.62 Θέμα 4 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα της εκτέλεσης του παρακάτω κώδικα Python:

```
a=3
while a>=0:
    a=a-2
    b=b+a
print(b)
```

- 1
- 0
- 2
- NameError
- IndexError
- -1

Απάντηση

- **NameError**

Παρατήρηση

Ισχύει η συνθήκη a=3>0

Με την εντολή a=a-2 το a γίνεται 1

- Στην εντολή *b=b+a* η b ΔΕΝ έχει αρχικοποιηθεί οπότε ΔΕΝ μπορεί να εκτελεστεί η πρόσθεση αυτή και εμφανίζεται NameError δηλ. σφάλμα στο όνομα της μεταβλητής

Traceback (most recent call last):

File "C:/Users/USER/AppData/Local/Programs/Python/Python310/ex4.py", line 4, in <module>

b=b+a

NameError: name 'b' is not defined

Μια σωστή έκδοση του κώδικα θα ήταν:

```
a=3  
b=0 #οποιαδήποτε τιμή μπαίνει στο b  
while a>=0:  
    a=a-2  
    b=b+a  
print(b)
```

5.15.63 Θέμα 5 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα του παρακάτω κώδικα:

```
for i in range(15, 3, -3):
```

```
    print(i+1, end="")  
    if i == 12: break
```

- 161396
- 1613
- 6 9 12 15 18
- -8 10 -12
- Καμία από τις απαντήσεις

Απάντηση

1613

Παρατήρηση

- Στο i καταχωρούνται οι τιμές από το 15 (αρχική τιμή) μέχρι το 3 (τελική τιμή) με βήμα μείωσης -3
- Αρχικά i=15 τυπώνεται 16. Δεν ισχύει η συνθήκη i == 12 και το for συνεχίζει
- Το end="" σημαίνει δεν υπάρχει αλλαγή γραμμής και μεταξύ των τιμών που τυπώνονται δεν υπάρχει ούτε κενό
- Το i από το βήμα μείωσης -3 γίνεται i=12 και τυπώνεται 13
- Ισχύει η συνθήκη i == 12 και το for τερματίζει με το break

5.15.64 Θέμα 6 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα του παρακάτω κώδικα Python:

```
a=1  
def x(n):  
    a=4  
    print(a, n, end=' ')  
    return a  
  
print(x(3), a)
```

- 1 2 4 4
- 4 4 2 1
- 2 4 1 1
- 4 3 4 1
- 4 2 1 4
- 4 2 1

Απάντηση

4 3 4 1

Παρατήρηση

- Καλείται η συνάρτηση x με τιμή 3 η οποία μεταβιβάζεται στην παράμετρο n
- ΠΡΟΣΟΧΗ: ΣΤΗΝ ΕΝΤΟΛΗ a=4 ΟΡΙΖΕΤΑΙ ΜΙΑ ΝΕΑ ΤΟΠΙΚΗ ΜΕΤΑΒΑΛΗΤΗ ΜΕΣΑ ΣΤΗ ΣΥΝΑΡΤΗΣΗ ΠΟΥ ΔΕΝ ΕΧΕΙ ΣΧΕΣΗ ΜΕ ΤΟ a ΤΟΥ MAIN. Στο τοπικό a μπαίνει η τιμή 4
- Τυπώνεται το τοπικό a που είναι 4 και η τιμή του n που είναι 3 και την οποία λαμβάνει από το main κατά την κλήση x(3)
- Η συνάρτηση τερματίζει με την εντολή return και επιστρέφει το 4.
- Στο main τυπώνεται το 4 και η τιμή του καθολικού a που είναι 1

5.15.65 Θέμα 7 Σεπτέμβριος 2021

Ποιο το αποτέλεσμα του επόμενου κώδικα:

```
a=16 // (5%4)
```

```
print(a);
```

- 4
- 8
- 16.0
- 16

Απάντηση

16

Παρατήρηση

- Πρώτα υπολογίζεται το ακέραιο υπόλοιπο της διαιρεσης $5\%4$ που είναι ίσο με 1 διότι είναι σε παρένθεση
- Ο τελεστής // υπολογίζει το ακέραιο πηλίκο της διαιρεσης με στρογγυλοποίηση στον πλησιέστερο ακέραιο προς τα κάτω αν το πηλίκο είναι πραγματικός

Παρατήρηση

$a=16 / (5\%4)$

`print(a);`

Ο κώδικας αυτός τυπώνει 16.0 διότι το / είναι ο τελεστής του πραγματικού πηλίκου

5.15.66 Θέμα 8 Σεπτέμβριος 2021

Τι θα τυπώσει ο παρακάτω κώδικα Python:

`x=5`

```
while x>0:    ←
    x=x+1
    if x<8: continue
    if x>10: break
    print(x, end = ' ')
```

- 5 6 7 8
- 5 6 7 8 9 10
- 6 7 8
- 6 7 8 9 10
- 9 10
- Καμία από τις απαντήσεις

Απάντηση

Καμία από τις απαντήσεις (διότι τυπώνονται οι τιμές 8 9 10)

Παρατήρηση

- Για $x=5$ ισχύει η συνθήκη $x>0$
- Το x γίνεται 6
- Για $x=6$ ισχύει η συνθήκη $x<8$ και εκτελεί `continue` στον επόμενο βρόγχο δηλ πάει απευθείας στο while
- Για $x=6$ ισχύει η συνθήκη $x>0$
- Το x γίνεται 7
- Για $x=7$ ισχύει η συνθήκη $x<8$ και εκτελεί `continue` στον επόμενο βρόγχο δηλ πάει απευθείας στο while
- Για $x=7$ ισχύει η συνθήκη $x>0$
- Το x γίνεται 8

- Για $x=8$ ΔΕΝ ισχύει η συνθήκη $x < 8$
- Για $x=8$ ΔΕΝ ισχύει η συνθήκη $x > 10$
- Για $x=8$ εκτελεί `print` και τυπώνει 8
- Το x γίνεται 9
- Για $x=9$ ΔΕΝ ισχύει η συνθήκη $x < 8$
- Για $x=9$ ΔΕΝ ισχύει η συνθήκη $x > 10$
- Για $x=9$ εκτελεί `print` και τυπώνει 9
- Το x γίνεται 10
- Για $x=10$ ΔΕΝ ισχύει η συνθήκη $x < 8$
- Για $x=10$ ΔΕΝ ισχύει η συνθήκη $x > 10$
- Για $x=10$ εκτελεί `print` και τυπώνει 10
- Για $x=11$ ο βρόγχος τερματίζει διότι ισχύει η συνθήκη `if x>10` και κάνει `break` από το βρόγχο

5.15.67 Θέμα 9 Σεπτέμβριος 2021

Να συμπληρώσετε τον παρακάτω κώδικα Python:

```
print ([x for x in range(18) .....])
```

ώστε να παράγει το εξής αποτέλεσμα:

```
[4, 5, 6, 11, 12, 13]
```

Απάντηση

```
print ([x for x in range(18) if x%7>=4])
```

Παρατήρηση

- Στο x μπαίνουν οι τιμές από 0 έως 17
- Κάθε τιμή διαιρείται με το 7 και λαμβάνεται το υπόλοιπο της με τον τελεστή %
- Αν αυτό το υπόλοιπο είναι $>=4$ η τιμή τυπώνεται
- Αν $x=0$ τότε $0\%7=0$ και δεν ισχύει η συνθήκη $0\geq4$
- Αν $x=1$ τότε $1\%7=1$ και δεν ισχύει η συνθήκη $1\geq4$
- Αν $x=2$ τότε $2\%7=2$ και δεν ισχύει η συνθήκη $2\geq4$
- Αν $x=3$ τότε $3\%7=3$ και δεν ισχύει η συνθήκη $3\geq4$
- **Αν $x=4$ τότε $3\%7=3$ και ισχύει η συνθήκη $4\geq4$ και τυπώνεται 4**
- **Αν $x=5$ τότε $5\%7=5$ και ισχύει η συνθήκη $5\geq4$ και τυπώνεται 5**
- **Αν $x=6$ τότε $6\%7=6$ και ισχύει η συνθήκη $6\geq4$ και τυπώνεται 6**

- Av x=7 τότε $7\%7=0$ και δεν ισχύει η συνθήκη $0 \geq 4$
- Av x=8 τότε $8\%7=1$ και δεν ισχύει η συνθήκη $1 \geq 4$
- Av x=9 τότε $9\%7=2$ και δεν ισχύει η συνθήκη $2 \geq 4$
- Av x=10 τότε $10\%7=3$ και δεν ισχύει η συνθήκη $3 \geq 4$
- **Av x=11 τότε $11\%7=4$ και ισχύει η συνθήκη $4 \geq 4$ και τυπώνεται 11**
- **Av x=12 τότε $12\%7=5$ και ισχύει η συνθήκη $5 \geq 4$ και τυπώνεται 12**
- **Av x=13 τότε $13\%7=6$ και ισχύει η συνθήκη $6 \geq 4$ και τυπώνεται 13**
- Av x=14 τότε $14\%7=0$ και δεν ισχύει η συνθήκη $0 \geq 4$
- Av x=15 τότε $15\%7=0$ και δεν ισχύει η συνθήκη $1 \geq 4$
- Av x=16 τότε $17\%7=0$ και δεν ισχύει η συνθήκη $2 \geq 4$
- Av x=17 τότε $17\%7=0$ και δεν ισχύει η συνθήκη $3 \geq 4$

5.15.68 Θέμα 4 Ιούνιος 2022

(a) Με τι πρέπει να συμπληρώστε τον κώδικα ώστε να δίνει στην έξοδο 1 2 3 4

```
def f(x):  
    for a in x:  
        .....
```

`f([1, 2, 3, 4])`

Απάντη

```
def f(x):  
    for a in x:
```

print(

(b) Ποιο το αποτελεσμα της εκτενων προγραμματος που διαβαζει μια λιστα απο την εισοδηματικη γραμμη και την εκτελει μια διαδικαση σε αυτην.

```
for i in [3, 4]:  
    for j in [1, 3]:  
        if i+j>3: break  
        print(i+j, end=' ')  
    print(i-1, end=' ')  
print(i-1, end=' ')
```

Απάντηση

233

Αιτιολόγηση

Γ_{1α}, j=3

Για $j=1$ if $i+j=3+1=4>3$ Ισχύει, κάνεται break από το βρόγχο του i και συνεχίζει στο βρόγχο του i και τυπώνει $i-1$ δηλ.

3-1=2

$\Gamma_{1\alpha, j=4}$

Για $i=1$ if $i+i=4+1=5>3$ Ισχύει, κάνει break από το βρόγχο του i και συνεχίζει στο βρόγχο του i και τυπώνει $i-1$ δηλ.

4-1=3

Το εξωτερικό for με το i τελειώνει

Στο τελευταίο print που είναι έξω και από τα 2 for τυπώνει i-1 δηλ. 4-1=3

(c) Με τι πρέπει να συμπληρώσετε τον κώδικα ώστε να δίνει στην έξοδο του [5, 6, 12, 13];

```
print([x for x in range(15) ....])
```

Απάντηση

```
print([x for x in range(15) if x==5 or x==6 or x==12 or x==13])
```

1 2

3

Πιθανό Θέμα

Ποιο το αποτέλεσμα της εκτέλεσης του τμήματος προγράμματος;

```
for i in [3, 4]:  
    for j in [1, 3]:  
        print(i+j, end=' ')  
    print(i-1, end=' ')
```

```
print(i-1, end=' ')
```

Απάντηση

4 6 2 5 7 3 3

Παρατήρηση

i=3

j=1 Τυπώνει 1+3=4

j=3 Τυπώνει 3+3=6

Τυπώνει 3-1=2

i=4

j=1 Τυπώνει 4+1=5

j=3 Τυπώνει 4+3=7

Τυπώνει 4-1=3

Τυπώνει 4-1=3

5.15.69 Θέμα 3 Σεπτέμβριος 2022

(a) Ποιο το αποτέλεσμα της εκτέλεσης του τμήματος προγράμματος;

`for x in range(12, 2, -2):`

`print(x+2, end='')`

`if x==4:`

`break`

Απάντηση

14 12 10 86

Παρατήρηση- Επεξήγηση Κώδικα

- Γενικά στην εντολή `for x in range(12, 2, -2)` το x αρχίζει από 12, τελειώνει στο 2 και σε κάθε επανάληψη το x μειώνεται κατά 2. Αφού εκτελεστεί μια επανάληψη γίνεται έλεγχος αν $x \geq 2$. Αν είναι αληθής η συνθήκη, εκτελείται η επόμενη επανάληψη και τερματίζει η επανάληψη όταν το x=2 (κανονικά εκτελείται και για x=2)
- Αρχικά το x=12. Ελέγχεται αν $x \geq 2$. **Είναι αληθής η συνθήκη και μπαίνει στην επανάληψη και τυπώνει 14**
- Μετά ΔΕΝ ισχύει η συνθήκη x=4 και γυρίζει στο for
- Το x μειώνεται κατά 2 και γίνεται 10. Ελέγχεται αν $x \geq 2$. **Είναι αληθής η συνθήκη και μπαίνει στην επανάληψη και τυπώνει 12**
- Μετά ΔΕΝ ισχύει η συνθήκη x=4 και γυρίζει στο for
- **Το x μειώνεται κατά 2 και γίνεται 8. Ελέγχεται αν $x \geq 2$. Είναι αληθής η συνθήκη και μπαίνει στην επανάληψη και τυπώνει 10**
- Μετά δεν ισχύει η συνθήκη x=4 και γυρίζει στο for
- **Το x μειώνεται κατά 2 και γίνεται 6. Ελέγχει αν $x \geq 2$. Είναι αληθής η συνθήκη και μπαίνει στην επανάληψη και τυπώνει 8**
- Μετά ΔΕΝ ισχύει η συνθήκη x=4 και γυρίζει στο for
- **Το x μειώνεται κατά 2 και γίνεται 4. Ελέγχεται αν $x \geq 2$. Είναι αληθής η συνθήκη και μπαίνει στην επανάληψη και τυπώνει 6**
- Μετά ισχύει η συνθήκη x=4 και κάνει break και εξέρχεται από το for διότι η εντολή break τερματίζει το for

Παρατήρηση

Η εντολή `print(x+2, end='')` καταργεί την προεπιλεγμένη αλλαγή γραμμής στην Python και βάζει ως χαρακτήρας ανάμεσα στις τιμές που τυπώνονται ένα κενό διάστημα

(β) Ποιο το αποτέλεσμα της εκτέλεσης του τμήματος προγράμματος;

`x=4`

`for x in "12345678":`

`print(x*x, end='')`

Απάντηση

Ο ΚΩΔΙΚΑΣ ΔΕΝ ΕΚΤΕΛΕΙΤΑΙ ΣΩΣΤΑ ΚΑΙ ΠΡΟΚΥΠΤΕΙ ΣΥΝΤΑΚΤΙΚΟ ΣΦΑΛΜΑ ΚΑΘΩΣ Η ΜΕΤΑΒΛΗΤΗ x EINAI ΤΥΠΟΥ STRING ΚΑΙ ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΕΦΑΡΜΟΣΤΕΙ Ο ΤΕΛΕΣΤΗΣ * ΣΕ ΑΥΤΗ

Παρατίμηση

Στην εντολή `for x in "12345678"`: σε κάθε επανάληψη στο x μπαίνει ένας χαρακτήρας και πολύζεται χαρακτήρας με χαρακτήρας πρόγια που ΔΕΝ γίνεται και γιαντό εμφανίζεται συντακτικό σφάλμα

Μια σωστή έκδοση του κώδικα θα πορούσε να είναι η επόμενη:

x=4

`for x in "12345678":`

`print(x*4, end='')`

που δίνει ως αποτέλεσμα:

1111 2222 3333 4444 5555 6666 7777 8888

(γ) Ποιο το αποτέλεσμα της εκτέλεσης του τμήματος προγράμματος;

i=5

`while i>0:`

`i=i+1`

`if i<7: continue`

`if i>12: break`

`print(i, end='')`

Απάντηση

7 8 9 10 11 12

5.15.70 Θέμα Φεβρουάριος 2023

Το παραγοντικό ενός φυσικού αριθμού ν είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων με ν: $v! = 1 \cdot 2 \cdot 3 \cdots \cdot v$
Για παράδειγμα το $4!$ υπολογίζεται ως:

$$1 * 2 * 3 * 4 = 24$$

ενώ το $5!$ υπολογίζεται ως:

$$1 * 2 * 3 * 4 * 5 = 120$$

Σε αυτήν την άσκηση πρέπει να γράψετε μια **συνάρτηση σε Python με όνομα factorial** που θα επιστρέφει το παραγοντικό του αριθμού που έχει περαστεί ως όρισμα.

Χρησιμοποιήστε ένα **βρόγχο for για να καλέσετε την συνάρτηση factorial έτσι ώστε να επιστρέφει τα πρώτα 36 παραγοντικά (0 έως 35)**

Απάντηση

Αναδρομική Συνάρτηση	Επαναληπτική Συνάρτηση
<code>def factorial(n):</code>	<code>def factorial(n):</code>
<code> if n == 0:</code>	<code> p=1</code>
<code> return 1</code>	<code> for i in range(1, n+1):</code>
<code> else:</code>	<code> p=p*i</code>
<code> return factorial(n-1)*n</code>	<code> return p</code>
<code>for i in range(36):</code>	<code>for i in range(36):</code>
<code> print(factorial(i))</code>	<code> print(factorial(i))</code>

Παράδειγμα Αναδρομικής Συνάρτησης για n=3

`factorial(3)`
{



```
return factorial(2)*3;  
}  
print(factorial(3))  
  
} } } } return 1;
```

Αποτελέσματα Εκτέλεσης

1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
640237370572800
12164510040883
24329020081766
51090942171709
11240007277776
25852016738884
62044840173323
15511210043330
40329146112660
10888869450418
30488834461171
88417619937397
26525285981219
82228386541779
26313083693369
86833176188118
29523279903960
10333147966386

5.15.71 Θέμα Ιούνιος 2023

Υλοποιήστε πρόγραμμα σε Python που να εξομοιώνει τη λειτουργία ενός ηλεκτρονικού καλαθιού αγορών με χρήση πλειάδων ή λεξικού. Οι απαραίτητες συναρτήσεις που πρέπει – τουλάχιστον – να υλοποιήσετε, είναι οι παρακάτω:

- Add_to_basket (προσθήκη προϊόντος στο καλάθι)
- Remove_from_basket (αφαίρεση προϊόντος από το καλάθι)
- Calculate_total_cost (υπολογισμός συνολικού κόστους του καλαθιού)
- Πραγματοποιήστε την προσθήκη τουλάχιστον τριών προϊόντων, και εκτυπώστε το περιεχόμενο του καλαθιού και το συνολικό κόστος. Στη συνέχεια αφαιρέστε ένα προϊόν και εκτυπώστε πάλι το περιεχόμενο του καλαθιού.

Απάντηση

Πρόγραμμα σε Python που εξομοιώνει τη λειτουργία ενός ηλεκτρονικού καλαθιού αγορών γρηγοροποιώντας λεξικά.

Αρχικοποίηση του καλαθιού αγορών ως ένα κενό λεξικό

```
basket = {}
```

Συνάρτηση για προσθήκη προϊόντος στο καλάθι

```
def add_to_basket(product, price, quantity):  
    if product in basket:  
        basket[product]['quantity'] += quantity  
    else:  
        basket[product] = {'price': price, 'quantity': quantity}
```

Συνάρτηση για αφαίρεση προϊόντος από το καλάθι

```
def remove_from_basket(product):  
    if product in basket:  
        del basket[product]
```

Συνάρτηση για υπολογισμό συνολικού κόστους του καλαθιού

```
def calculate_total_cost():  
    total_cost = sum(item['price'] * item['quantity'] for item in basket.values())  
    return total_cost
```

Προσθήκη προϊόντων στο καλάθι

```
add_to_basket("Προϊόν1", 10.0, 2)  
add_to_basket("Προϊόν2", 5.0, 3)  
add_to_basket("Προϊόν3", 8.0, 1)
```

Εκτύπωση περιεχομένου του καλαθιού και συνολικού κόστους

```
print("Περιεχόμενο του καλαθιού:")  
for product, info in basket.items():  
    print(f"Προϊόν: {product} Τιμή: {info['price']} Ευρώ, Ποσότητα: {info['quantity']}")
```

```
total_cost = calculate_total_cost()
```

```
print(f"Συνολικό κόστος: {total_cost} Ευρώ")
```

```
# Αφαίρεση ενός προϊόντος από το καλάθι
remove_from_basket("Προϊόν2")

# Εκτύπωση περιεχομένου του καλαθιού και νέου συνολικού κόστους
print("\n Περιεχόμενο του καλαθιού μετά την αφαίρεση ενός προϊόντος:")
for product, info in basket.items():
    print(f" {product}: Τιμή: {info['price']} Ευρώ, Ποσότητα: {info['quantity']}")

total_cost = calculate_total_cost()
print(f"Νέο συνολικό κόστος: {total_cost} Ευρώ")
```

5.16 ΘΕΜΑΤΑ ΣΕ PYTHON ΑΠΟ ΗΛΕΚΤΡΟΛΟΓΟΥΣ

5.16.1 Θέμα 1

Τι εκτυπώνει ο ακόλουθος κώδικας Python;

```
myItems=[1, 2, 3, 4, 5, 6]
for i in range(1, 6):
    myItems[i-1]=myItems[i]
for i in range(0, 6):
    print(myItems[i], end=' ')
```

Απάντηση

2 3 4 5 6 6

5.16.2 Θέμα 2

Ποια η έξοδος του ακόλουθου κώδικα Python;

```
x="abcdef"
i="a"
while i in x:
    print(i, end=' ')
```

Απάντηση

Τυπώνει το a άπειρες φορές

5.16.3 Θέμα 3

Ποιο το αποτέλεσμα του ακόλουθου κώδικα Python και ποια δομή χρησιμοποιήσατε;

```
welcome="ela mesa"
print(welcome)
first=welcome[4:]
second=welcome[:3]
both=(first, second)
print(both)
both.append("now")
```

Απάντηση

ela mesa

('mesa', 'ela')

Traceback (most recent call last):

```
both.append("now")
```

AttributeError: 'tuple' object has no attribute 'append' #Εμφανίζεται λάθος στη γραμμή αντή διότι ο τύπος tuple δεν έχει μέθοδο append

Χρησιμοποιήσαμε τη δομή tuple

5.16.4 Θέμα 4

Έστω ότι έχουμε τον παρακάτω κώδικα Python. Ποιο αποτέλεσμα θα έχετε και γιατί; Έχετε κάποια πρόταση να βελτιωθεί;

class Student:

```
def say_something(self):
    print("I passes the exams")
```

Student.say_something()

Απάντηση

Ο κώδικας όπως είναι γραμμένος έχει συντακτικό λάθος και δεν θα εκτελεστεί

Το λάθος είναι στη συνάρτηση say_something() και είναι το ακόλουθο:

Traceback (most recent call last):

```
File "C:\Users\USER\PycharmProjects\Eisagogi\test.py", line 4, in <module>
```

```
    Student.say_something()
```

TypeError: Student.say_something() missing 1 required positional argument: 'self'

Η διόρθωση του κώδικα είναι η εξής:

```
class Student:
```

```
    def say_something(self):  
        print("I passes the exams")
```

```
s=Student()
```

```
s.say_something()
```

Η έξοδος του κώδικα αυτού είναι το μήνυμα **I passed the exams**

5.16.5 Θέμα 5-OXI

Να γράψετε πρόγραμμα που ζητάει ένα κείμενο από το χρήστη και εκτυπώνει ένα λεξικό με το πλήθος κάθε χαρακτήρα του κειμένου, να αγνοήσετε τα κενά στο κείμενο. Όταν ο χρήστης δώσει <enter> τερματίζει το πρόγραμμα, αλλιώς ζητάει κείμενο ξανά.

Παράδειγμα

>>>κείμενο:καλή σας μέρα

{'κ': 1, 'α': 3, 'λ': 1, 'ή': 1, 'σ': 1,

>>>κείμενο:καληνύχτα

{'κ': 1, 'α': 2, 'λ': 1, 'ή': 1, 'ν': 1,

>>>κείμενο:

Απάντηση

while True:

 freq = {} #δημιουργία κενού λεξικού

 txt = **input**("κείμενο:")

if not txt: **break**

for ch **in** txt:

if ch == " ": **continue**

 freq[ch] = freq.get(ch, 0) + 1

print(freq)

5.16.6 Θέμα 6

Να γράψετε πρόγραμμα που ρωτάει το όνομα του χρήστη και απαντάει με χαιρετισμό στα κεφαλαία - χωρίς τόνους, θεωρήστε ότι τα τονούμενα γράμματα είναι τα α, ο, ε, η, υ, ι, ω. Το πρόγραμμα τερματίζει όταν ο χρήστης δώσει <enter>

Παράδειγμα

>>> Πώς σε λένε; Κώστα ΓΕΙΑ ΣΟΥ ΚΩΣΤΑ

>>> Πώς σε λένε; Μαρία ΓΕΙΑ ΣΟΥ ΜΑΡΙΑ

>>> Πώς σε λένε;

Απάντηση

tonoi = {"ά": "α", "έ": "ε", "ή": "η", "ί": "ι", "ό": "ο", "ύ": "υ", "ώ": "ω"}

while True:

 name = **input**("Πώς σε λένε; ") **if not** name: **break**

 cap_name = "" **for** ch **in** name:

if ch **in** tonoi: cap_name += tonoi[ch] **else**: cap_name += ch

print(f"ΓΕΙΑ ΣΟΥ {cap_name.upper()}")

5.17 Θέμα 7

Να γράψετε πρόγραμμα που δέχεται ένα ακέραιο αριθμό από τον χρήστη και απαντάει αν είναι άρτιος ή περιττός, Αν ο χρήστης δεν δώσει ακέραιο αριθμό να δίνει σχετικό μήνυμα, όταν ο χρήστης πατήσει <enter> το πρόγραμμα να τερματίζει

Παράδειγμα

>>> δώσε ακέραιο αριθμό: 123 περιττός αριθμός

>>> δώσε ακέραιο αριθμό: -2 άρτιος αριθμός

>>> δώσε ακέραιο αριθμό: abc Παρακαλώ δώστε ακέραιο αριθμό!

>>> δώσε ακέραιο αριθμό:

Απάντηση

while True:

 n = **input**('δώσε ακέραιο αριθμό: ')

if not n: **break**

try:

```
n = int(n)
```

except:

```
print('Παρακαλώ δώστε ακέραιο αριθμό!')
```

continue

```
if int(n)%2: print('περιττός αριθμός')
```

```
else: print('άρτιος αριθμός')
```

5.18 Θέμα 8

Να γράψετε πρόγραμμα που ζητάει ένα κείμενο από τον χρήστη και εκτυπώνει ένα λεξικό με το πλήθος κάθε ελληνικού χαρακτήρα του κειμένου. Όταν ο χρήστης δώσει <enter> τερματίζει το πρόγραμμα, αλλιώς ζητάει κείμενο ξανά.

Σημείωση: οι ελληνικοί χαρακτήρες στον πίνακα utf-8 έχουν τιμές από 902 μέχρι 980

Παράδειγμα

κείμενο: Καλημέρα Mr Jhonson

```
{'Κ': 1, 'α': 2, 'λ': 1, 'η': 1, 'μ': 1, 'έ': 1, 'ρ': 1}
```

κείμενο: Goodmoning Κώστα

```
{'Κ': 1, 'ώ': 1, 'σ': 1, 'τ': 1, 'α': 1}
```

κείμενο:

Απάντηση

while True:

```
freq = {}  
txt = input("κείμενο:")  
if not txt: break  
for ch in txt:  
    if ord(ch) < 902 or ord(ch) > 980: continue  
    freq[ch] = freq.get(ch, 0) + 1  
print(freq)
```

5.19 Θέμα 9

Γράψτε πρόγραμμα που ζητάει από τον χρήστη να βάλει ένα προϊόν και την αντίστοιχη ποσότητα +προϊόν, ποσότητα ή να βγάλει ένα -προϊόν από το ψυγείο, το πρόγραμμα απαντάει με ταξινομημένη λίστα των προϊόντων που υπάρχουν στο ψυγείο με τις αντίστοιχες ποσότητες. Το πρόγραμμα τερματίζει με πάτημα <enter>

Παράδειγμα

Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν: +μπύρες,5 Το ψυγείο περιέχει:

('μπύρες', '5')

Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν: +γιαούρτια,4 Το ψυγείο περιέχει:

('γιαούρτια', '4')

('μπύρες', '5')

Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν: +γάλα,1 Το ψυγείο περιέχει:

('γάλα', '1')

('γιαούρτια', '4')

('μπύρες', '5')

Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν: -μπύρες Το ψυγείο περιέχει:

('γάλα', '1')

('γιαούρτια', '4')

Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν:

Απάντηση

```
fridge = []
```

while True:

```
proion = input("Προσθέστε (+) προϊόν, ποσότητα ή αφαιρέστε (-) προϊόν:")
if not proion: break
if proion[0] == "+": fridge.append(tuple(proion[1:]).split(","))
elif proion[0] == "-": fridge = [x for x in fridge if x[0] != proion[1:]]
print("Το ψυγείο περιέχει:")
for p in sorted(fridge):
    print(p)
```

5.20 Θέμα 10

Γράψτε πρόγραμμα που επιτρέπει την εγγραφή σε μια εφαρμογή (επιλογή 1) ή είσοδο του χρήστη στην εφαρμογή (επιλογή 2). Το πρόγραμμα τερματίζει με πάτημα <enter>. Για την εγγραφή ζητάει όνομα χρήστη και κωδικό δύο φορές, αν το όνομα δεν υπάρχει ήδη και οι δύο κωδικοί είναι ίδιοι, αποθηκεύει τον χρήστη και απαντάει "Επιτυχής εγγραφή" Για την είσοδο του χρήστη, ζητάει το όνομα του χρήστη και τον μυστικό κωδικό, αν ο χρήστης υπάρχει και ο κωδικός είναι σωστός του απαντάει "Καλωσήρθατε!" αν όχι του επιστρέφει μιμήνυμα σφάλματος όπως στο παράδειγμα.

Παράδειγμα

1. Εγγραφή, 2. Είσοδος. Η επιλογή σας:1 Όνομα:Nikos

Κωδικός:123

Κωδικός(επανάληψη):123 Επιτυχής εγγραφή

1. Εγγραφή, 2. Είσοδος. Η επιλογή σας:2 Όνομα:nikos

Κωδικός:123

Λάθος χρήστης/κωδικός

1. Εγγραφή, 2. Είσοδος. Η επιλογή σας:2 Όνομα:Nikos

Κωδικός:123 Καλωσήρθες

1. Εγγραφή, 2. Είσοδος. Η επιλογή σας:

users = [] while True:

```
    reply = input("1. Εγγραφή, 2. Είσοδος. Η επιλογή σας:") if not reply: break
```

```
    if reply == "1":
```

```
        name = input("Όνομα:") pass1 =
```

```
        input("Κωδικός:")
```

```
        pass2 = input("Κωδικός (επανάληψη):")
```

```
        if name not in [x[0] for x in users] and pass1 == pass2: users.append((name, pass1))
```

```
        print("Επιτυχής εγγραφή") else: print('Σφάλμα
```

```
        εγγραφής')
```

```
    elif reply == '2':
```

```
        name = input("Όνομα:") passw =
```

```
        input("Κωδικός:")
```

```
        if (name, passw) in users: print("Καλωσήρθες") else: print("Λάθος χρή-
```

```
        στης/κωδικός")
```

6 λ-Λογισμός

Το μοντέλο υπολογισμού του Church ονομάζεται **λογισμός Λάμβδα**

- βασίζεται στην έννοια των παραμετρικών εκφράσεων (κάθε παράμετρος εισάγεται με μια εμφάνιση του γράμματος λ –από όπου και το όνομα της σημειογραφίας)
- ο λογισμός Λάμβδα ήταν η έμπνευση για το συναρτησιακό προγραμματισμό
- ο υπολογισμός γίνεται με την αντικατάσταση παραμέτρων σε εκφράσεις, ακριβώς όπως σε ένα συναρτησιακό πρόγραμμα υψηλού επιπέδου ο υπολογισμός γίνεται με τη μεταβίβαση ορισμάτων σε συναρτήσεις

6.1 “Έννοιες Συναρτησιακού Προγραμματισμού

Οι γλώσσες συναρτησιακού προγραμματισμού όπως η Lisp, η Scheme, η FP, η ML, η Miranda, και η Haskell είναι μια προσπάθεια να πραγματωθεί ο λογισμός Λάμβδα του Church σε μια πρακτική μορφή σαν γλώσσα προγραμματισμού. Έχουν αρκετές δυνατότητες, αρκετές οι οποίες λείπουν από κάποιες προστακτικές γλώσσες.

- **συναρτήσεις πρώτης κατηγορίας και υψηλότερης τάξης**
- ισχυρός πολυμορφισμός
- **ισχυρές λειτουργίες για λίστες**
- **αναδρομή**
- δομημένα αποτελέσματα συναρτήσεων
- πραγματικά γενικές σταθερές σύνθετου τύπου
- συλλογή σκουπιδιών

Τα **Πλεονεκτήματα των συναρτησιακών γλωσσών** είναι:

- η έλλειψη παρενεργειών διευκολύνει την κατανόηση των προγραμμάτων
- η έλλειψη ρητής σειράς αποτίμησης (σε μερικές γλώσσες) προσφέρει την πιθανότητα παράλληλης αποτίμησης (π.χ. MultiLisp)
- η έλλειψη παρενεργειών και ρητής σειράς αποτίμησης απλοποιούν κάποια πράγματα για το μεταγλωττιστή
- τα προγράμματα συχνά είναι εκπληκτικά μικρά
- η γλώσσα μπορεί να είναι εξαιρετικά μικρή αλλά ισχυρή

Η βασική Ιδέα:

- τα πάντα γίνονται με τη σύνθεση συναρτήσεων
- δεν υπάρχει μεταβλητή κατάσταση
- δεν υπάρχουν παρενέργειες

Πώς κάνουμε κάτι σε μια συναρτησιακή γλώσσα; **Η αναδρομή αντικαθιστά την επανάληψη.** Γενικά, μπορούμε να έχουμε το ίδιο αποτέλεσμα με μια σειρά αναθέσεων

x := 0 ...

x := expr1 ...

x := expr2 ...

με την $f3(f2(f1(0)))$, όπου κάθε f περιμένει την τιμή του x σαν παράμετρο, η $f1$ επιστρέφει $expr1$ και η $f2$ επιστρέφει $expr2$

Πιθανό Θέμα

Η αναδρομή αντικαθιστά με επιτυχία ακόμα και τους βρόχους

Ο επόμενος βρόχος:

```
x:=0; i:=1; j:=100;  
while i<j do  
    x :=x+ i*j; i :=i+1; j:= j-1  
end while  
return x
```

γίνεται αναδρομικά ως εξής:

$f(0, 1, 100)$ όπου

$f(x, i, j) == \text{if } i < j \text{ then } f(x+i*j, i+1, j-1) \text{ else } x$

Το να θεωρούμε όμως ότι η αναδρομή είναι μια απενθείας, μηχανική αντικατάσταση της επανάληψης είναι ο λάθος τρόπος να βλέπουμε τα πράγματα. Πρέπει να συνηθίσουμε να σκεπτόμαστε σε αναδρομικό στυλ

Πιο σημαντική έννοια και από την αναδρομή είναι οι συναρτήσεις υψηλότερης τάξης

- Παίρνουν μια συνάρτηση ως παράμετρο ή επιστρέφουν μια συνάρτηση ως αποτέλεσμα
- Χρήσιμες στην κατασκευή πραγμάτων

6.2 Λογισμός -λ- Θεωρητικές Έννοιες

Πιθανό Θέμα: Παράδειγμα εκφράσεων Λάμβδα-Πιθανό Θέμα

Όνομα	Περιεχόμενο	Περιγραφή
id	$\lambda x.x$	Η μεταβλητή x
const	$\lambda x.2$	Η σταθερά 2
plus	$\lambda x.\lambda y.x+y$	Η πράξη της πρόσθεσης $x+y$
square	$\lambda x.x*x$	Η πράξη του πολλαπλασιασμού $x*x$
hypot	$\lambda x.\lambda y.sqrt(plus(square x) (square y))$	Η πράξη υπολογισμού της υποτείνουσας

Αναδρομικά, μια έκφραση Λάμβδα είναι:

- (1) ένα όνομα
- (2) μια αφαίρεση που αποτελείται από ένα Λάμβδα, ένα όνομα, μια τελεία και μια έκφραση Λάμβδα
- (3) μια εφαρμογή που αποτελείται από δυο διπλανές Λάμβδα εκφράσεις (η παράθεση σημαίνει εφαρμογή συνάρτησης)

(4) μια έκφραση Λάμβδα σε παρενθέσεις

Παράδειγμα – Πιθανό Θέμα

Να γραφεί αναλυτικά ο υπολογισμός της υποτείνουσας ορθογωνίου τριγώνου με πλευρές 4 και 3 αντίστοιχα που καλείται ως εξής:

$$\lambda x. \lambda y. \text{sqrt}(\text{plus}(\text{square } x)(\text{square } y)) \ 4 \ 3$$

Απάντηση

$$\lambda x. \lambda y. \text{sqrt}(\text{plus}(\text{square } x)(\text{square } y)) \ 4 \ 3 =$$

$$= \lambda x. \lambda y. \text{sqrt}(\text{plus}(\lambda x. x^* x)(\lambda y. y^* y))$$

$$= \lambda x. \lambda y. \text{sqrt}(\lambda x. \lambda y. x^* x + y^* y) \ 4 \ 3 =$$

$$= \lambda y. \text{sqrt}(\lambda y. 4^* 4 + y^* y) \ 3 =$$

$$\text{sqrt}(4^* 4 + 3^* 3) =$$

$$\text{sqrt}(16 + 9) = 5$$

Ο λ-λογισμός είναι θεωρία συναρτήσεων και έχει δύο κύριες λειτουργίες

-**Εφαρμογή συνάρτησης** F πάνω σε ένα όρισμα A, που συμβολίζεται με F A

-**Αφαίρεση.** Έστω ότι x μεταβλητή και E[x] έκφραση που εξαρτάται από τη x. Η έκφραση $\lambda x. E[x]$ συμβολίζει τη συνάρτηση

$$x \mapsto E[x]$$

Η συνάρτηση δέχεται ως όρισμα μία τιμή x και επιστρέφει ως αποτέλεσμα την τιμή E[x]. Η x δεν εμφανίζεται απαραίτητα στην έκφραση E[x]. Αν αυτό δεν συμβαίνει, τότε η $\lambda x. E[x]$ είναι μία σταθερή συνάρτηση.

Παράδειγμα – Πιθανό Θέμα

Να υπολογιστεί η παράσταση $(\lambda x. x^2 - 3 \cdot x + 2) \ 8$ που καλείται με όρισμα 8

Απάντηση

Η $\lambda x. x^2 - 3x + 2$ συμβολίζει μία συνάρτηση που σε κάθε τιμή x απεικονίζει την τιμή $x^2 - 3x + 2$.

Αν η συνάρτηση εφαρμοσθεί στο όρισμα 8, προκύπτει $(\lambda x. x^2 - 3 \cdot x + 2) \ 8 = 8^2 - 3 \cdot 8 + 2 = 42$

Η τιμή του ορίσματος αντικαθιστά την παράμετρο x

- Συνήθως η εφαρμογή προσεταιρίζει από αριστερά προς τα δεξιά, επομένως η f A B είναι (f A) B και όχι f (A B)
- Επίσης, η εφαρμογή έχει υψηλότερη προτεραιότητα από την αφαίρεση, επομένως η λx. A B είναι λx.(A B) και όχι (λx. A) B
- Οι παρενθέσεις χρησιμοποιούνται για σαφήνεια ή για την παράβαση των κανόνων. Εξορισμό τις χρησιμοποιούμε σε κάθε αφαίρεση που χρησιμοποιείται ως συνάρτηση ή παράμετρος: $(\lambda f. f \ 2) (\lambda x. \text{plus} \ x \ x)$ ή σε κάθε εφαρμογή που χρησιμοποιείται ως παράμετρος: $\text{double} (\text{minus} \ 5 \ 2)$
- Στην $(\lambda x. \lambda y. \lambda z. e) \ a \ b \ c$ η αρχική συνάρτηση δέχεται μια παράμετρο και επιστρέφει μια συνάρτηση (μιας παραμέτρου) που επιστρέφει μια συνάρτηση (μιας παραμέτρου)

- Για την αναγωγή της έκφρασης, αντικαθιστούμε την a σε κάθε x στη λ.y.λ.z.e, στη συνέχεια αντικαθιστούμε την b σε κάθε y στην υπόλοιπη έκφραση και την c σε κάθε z σε ό,τι μένει στο τέλος

Παράδειγμα – Πιθανό Θέμα

Να υπολογιστεί η παράσταση **(λ.x.λ.y.x+y) 3 4**

Απάντηση

(λ.x.λ.y.x+y) 3 4 =

λ.y.(3+y) 4 =

(3+4)=

7

6.3 Ελεύθερη και Δεσμευμένη Μεταβλητή

- Η αφαίρεση $\lambda x.E[x]$ δεσμεύει τη μεταβλητή x μέσα στην έκφραση $E[x]$.
- Μεταβλητή μη δεσμευμένη ονομάζεται ελεύθερη

Παράδειγμα: $\lambda x. x^2 - 3y + 2$. **x δεσμευμένη και y ελεύθερη**

Παράδειγμα: $(\lambda x. x^2 - 3y + 2) (4x + 1)$. Η πρώτη εμφάνιση του x (στο x^2) είναι δεσμευμένη, γιατί είναι στο εσωτερικό της αφαίρεσης, ενώ η δεύτερη εμφάνιση του x (στο $4x + 1$) είναι ελεύθερη)

Παράδειγμα: Στη $\lambda x. \lambda y. (* x y)$ έχουμε δυο ένθετες εκφράσεις Λάμβδα: η x είναι ελεύθερη στην εσωτερική $(\lambda y. (* x y))$, αλλά δεσμευμένη στην εξωτερική

Η αποτίμηση των εκφράσεων Λάμβδα γίνεται μέσω:

(1) αντικατάστασης των παραμέτρων (**βήτα αναγωγή**) $(\lambda x. \text{times } x x) y \Rightarrow \text{times } y y$

(2) μετονομασίας των μεταβλητών (**άλφα μετατροπή**) (συχνά για να αποφύγουμε τις συγκρούσεις ονομάτων) $(\lambda x. \text{times } x x) y = (\lambda z. \text{times } z z) y$

(3) απλοποίηση «εκτός σειράς»(**ήτα αναγωγή**) $(\lambda x. f x) \Rightarrow f$

Ο τελευταίος κανόνας είναι δυσνόητος: ΔΕΝ είναι ο ίδιος με τη βήτα αναγωγή

7 Γλώσσα Προγραμματισμού Scala

7.1 Βασικά Χαρακτηριστικά

- Scala: ακρώνυμο του “Scalable Language”, δηλώνει ότι έχει σχεδιαστεί για να μπορεί να μεγαλώνει παράλληλα με τις ανάγκες των χρηστών της
- Χαρακτηριστικά αντικειμενοστραφή και συναρτησιακό προγραμματισμού, δίνει την αίσθηση scripting γλώσσας
- Συντακτικά ευέλικτη γλώσσα
- Εκτελείται στο JVM (Java Virtual Machine)
- Μοντέλο μεταγλώττισης ίδιο με Java και C#. Χρησιμοποιεί βιβλιοθήκες Java
- Λειτουργικά χαρακτηριστικά: παράγει κώδικα Byte (σχεδόν ίδιος με Java)
- Μπορεί να από-μεταγλωττιστεί σε κώδικα Java
- Μπορεί να τρέξει σε Android smartphones
- Εναλλακτική υλοποίηση σε .NET πλατφόρμα

7.2 Προγραμματιστικά Χαρακτηριστικά

Συντακτική Ευελιξία

- Semicolons δεν είναι απαραίτητα
- () στις κλήσεις συναρτήσεων δεν είναι απαραίτητα

Unified type system

- Όλοι οι τύποι δεδομένων κληρονομούν από την κλάση Any (που χωρίζεται σε AnyVal και AnyRef)

Αμεταβλητότητα

- Διαχωρισμός ανάμεσα σε σταθερές (val) και μεταβλητές που μπορεί να αλλάξει η τιμή τους (var)

Υποστηρίζει patterns κανονικών εκφράσεων

7.3 Παραδείγματα Scala

7.3.1 Σύνταξη Κώδικα σε Scala

AS A FIRST EXAMPLE

```
object HelloWorld {  
    def main(args: Array[String]) {  
        println("Hello, world!")  
    }  
}
```

7.3.2 Βασικές Εντολές Scala

IF STATEMENT

As in java
If (statement)
action

FOR LOOP

Similar to java
for(i <= 1 to 100){
 actions
}

IF-ELSE STATEMENT

As in java
If (statement1)
action1
else
action2

7.3.3 Διαφορές Scala και Java

SCALA METHOD DEFINITIONS

```
def fun (x: Int) = {  
    result  
}
```

JAVA METHOD DEFINITIONS

```
int fun (int x){  
    return result;  
}
```

7.3.4 Παράδειγμα 4

DEFAULT PARAMETER VALUES

```
def hello(foo:Int = 0, bar:Int = 0) {  
    println("foo: "+foo+" bar: "+bar)  
}
```

hello() → foo: 0 bar: 0
hello(1) → foo: 1 bar: 0
hello(1,2) → foo: 1 bar: 2

NAMED PARAMETERS

```
def hello(foo:Int = 0, bar:Int = 0) {  
    println("foo: "+foo+" bar: "+bar)  
}
```

hello(bar=6) → foo: 0 bar: 6
hello(foo=7) → foo: 7 bar: 0
hello(foo=8,bar=9) → foo: 8 bar: 9

7.3.5 Διερμηνευτής Εντολών Scala

Η Scala ενσωματώνει ένα διερμηνευτή εντολών (interpreter) που μπορεί να θεωρηθεί ότι λειτουργεί ως calculator. Ο χρήστης δίνει αριθμητικές παραστάσεις από το πληκτρολόγιο και ο calculator επιστρέφει το αντίστοιχο αποτέλεσμα όπως στο επόμενο παράδειγμα:

```
scala> 87 + 145
unnamed0: Int = 232

scala> 5 + 2 * 3
unnamed1: Int = 11

scala> "hello" + " world!"
unnamed2: java.lang.String = hello world!
```

Είναι εφικτό να ονομάσουμε μια υπο-έκφραση και να χρησιμοποιήσουμε το όνομα αυτή για την έκφραση όπως στο επόμενο παράδειγμα:

```
scala> def scale = 5
scale: Int
```

δηλώνουμε τη μεταβλητή scale και της καταχωρούμε ακέραια τιμή ως περιεχόμενο άρα ο τύπος της γίνεται int

```
scala> 7 * scale
unnamed3: Int = 35
```

χρησιμοποιούμε τη μεταβλητή scale και την πολλαπλασιάζουμε με την ακέραια τιμή 7 άρα προκύπτει ανώνυμη μεταβλητή τύπου int με περιεχόμενο

```
scala> def pi = 3.141592653589793
pi: Double
```

δηλώνουμε τη συνάρτηση square που λαμβάνει ένα όρισμα τύπου double και επιστρέφει ως αποτέλεσμα το τετράγωνο του ορίσματος

Δεν απαιτείται η εντολή return ώστε η συνάρτηση να επιτρέψει αποτέλεσμα. Οι συναρτήσεις στη Scala που επιστρέφουν αποτέλεσμα έχουν το σύμβολο = μετά την παρένθεση με τα ορίσματα

```
scala> def square(x: Double) = x * x
square: (Double)Double
```

```
scala> square(2)
unnamed0: Double = 4.0
```

καλείται η συνάρτηση square με όρισμα 2

```
scala> square(5 + 3)
unnamed1: Double = 64.0
```

καλείται η συνάρτηση square με όρισμα 8

```
scala> square(square(4))
unnamed2: Double = 256.0
```

Καλείται η συνάρτηση square δύο φορές. Την 1^η φορά με όρισμα 4 και επιστρέφει αποτέλεσμα 16 και τη 2^η φορά με όρισμα 16 και επιστρέφει $256=16*16$

```
scala> def sumOfSquares(x: Double, y: Double) = square(x) + square(y)
sumOfSquares: (Double,Double)Double
```

```
scala> sumOfSquares(3, 2 + 2)
unnamed3: Double = 25.0
```

Πιθανό Θέμα: Να γίνει Αναλυτικός Υπολογισμός Αποτελέσματος της sumOfSquares(3, 4)

Απάντηση

```
sumOfSquares(3, 2+2)
→ sumOfSquares(3, 4)
→ square(3) + square(4)
→ 3 * 3 + square(4)
→ 9 + square(4)
→ 9 + 4 * 4
→ 9 + 16
→ 25
```

Πιθανό Θέμα: Να γραφεί συνάρτηση στη Scala που να επιστρέφει την απολυτή τιμή του ορίσματος της

Απάντηση

```
def abs(x: Double) = if (x >= 0) x else -x
```

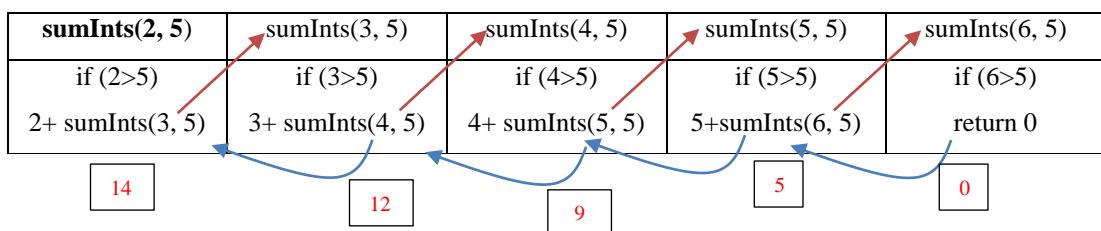
- Ορίζουμε τη συνάρτηση με όνομα abs. Το (x: Double) σημαίνει ότι λαμβάνει ένα όρισμα τύπου Double και επιστρέφει το x αν $x \geq 0$ ή τον αντίθετο του αν $x < 0$.
- Το (Double) Double σημαίνει ότι το όρισμα είναι τύπου Double και καλείται με call-by-value και ο τύπος του επιστρεφόμενου αποτελέσματος είναι τύπου Double

Πιθανό Θέμα: Να γραφεί αναδρομική συνάρτηση στη Scala που να αθροίζει όλους τους ακέραιους που βρίσκονται μεταξύ δύο δοθέντων αριθμών a και b

Απάντηση

```
def sumInts(a: Int, b: Int): Int = if (a > b) 0 else a + sumInts(a + 1, b)
```

Για παράδειγμα η κλήση sumInts(2, 5) υπολογίζει το άθροισμα $2+3+4+5=14$



Αναδρομική Συνάρτηση στη Scala για Άθροισμα Τετραγώνων Τιμών σε ένα Εύρος

Να γραφεί αναδρομική συνάρτηση σε Scala που **αθροίζει τα τετράγωνα των ακέραιων που βρίσκονται μεταξύ δύο δοθέντων αριθμών a και b**

Απάντηση

```
def square(x: Int): Int = x * x
def sumSquares(a: Int, b: Int): Int =
  if (a > b) 0 else square(a) + sumSquares(a + 1, b)
```

Η κλήση sumSquares(2, 5) υπολογίζει και επιστρέφει το άθροισμα $2^2+3^2+4^2+5^2$

Αναδρομική Συνάρτηση στη Scala για Άθροισμα Δυνάμεων σε ένα Εύρος

Να γραφεί αναδρομική συνάρτηση που **αθροίζει τις δυνάμεις 2^n όλων των ακέραιων μεταξύ δύο δοθέντων αριθμών a και b**

Απάντηση

```
def powerOfTwo(x: Int): Int = if (x == 0) 1 else 2 * powerOfTwo(x - 1)
def sumPowersOfTwo(a: Int, b: Int): Int =
  if (a > b) 0 else powerOfTwo(a) + sumPowersOfTwo(a + 1, b)
```

Η συνάρτηση powerOfTwo(x) επιστρέφει το 2^x

Η κλήση sumPowersOfTwo(2, 5) την καλεί και υπολογίζει το άθροισμα $2^2+2^3+2^4+2^5$

7.3.6 Ανώνυμες Συναρτήσεις (Anonymous Functions)

Μια ανώνυμη συνάρτηση είναι μια έκφραση η οποία αξιολογείται ως συνάρτηση και ορίζεται χωρίς να της δοθεί όνομα. Ως παράδειγμα μπορούμε να θεωρήσουμε την **ανώνυμη συνάρτηση τετραγώνου**:

```
(x: Int) => x * x
```

Το τμήμα μπροστά από το \Rightarrow είναι οι παράμετροι της συνάρτησης ενώ το τμήμα μετά το \Rightarrow είναι το σώμα της συνάρτησης. Για παράδειγμα μια ανώνυμη συνάρτηση που πολλαπλασιάζει τα δύο ορίσματα της είναι η ακόλουθη:

```
(x: Int, y: Int) => x * y
```

Χρησιμοποιώντας ανώνυμες συναρτήσεις μπορούμε να επανασχηματίσουμε τις δύο πρώτες συναρτήσεις αθροίσματος χωρίς ονομασμένες βοηθητικές συναρτήσεις

```
def sumInts(a: Int, b: Int): Int =  
    if (a > b) 0 else a + sumInts(a + 1, b)  
  
↓  
def sumInts(a: Int, b: Int): Int = sum((x: Int) => x, a, b)
```

Η συνάρτηση sumInts έχει ως σώμα το \Rightarrow x, λαμβάνει το όρισμα x, στο x μπαίνουν οι παράμεροι a, b και με την έτοιμη συνάρτηση sum αθροίζονται όλες τις τιμές που είναι στο εύρος [a, b]. Στο x τοποθετούνται όλες οι τιμές από a έως και b. Η έτοιμη συνάρτηση sum προσθέτει τις τιμές.

```
def square(x: Int): Int = x * x  
def sumSquares(a: Int, b: Int): Int =  
    if (a > b) 0 else square(a) + sumSquares(a + 1, b)  
  
↓  
def sumSquares(a: Int, b: Int): Int = sum((x: Int) => x * x, a, b)
```

Η συνάρτηση sumSquares έχει ως σώμα το \Rightarrow x*x, λαμβάνει το όρισμα x, στο x μπαίνουν οι παράμεροι a, b και με την έτοιμη συνάρτηση sum αθροίζονται όλα τα τετράγωνα των τιμών που είναι στο εύρος [a, b]. Στο x τοποθετούνται όλες οι τιμές από a έως και b. Η έτοιμη συνάρτηση sum προσθέτει τις τιμές.

7.3.7 Currying

Το Currying στη Scala είναι μια τεχνική που μετασχηματίζει μια συνάρτηση που λαμβάνει **πολλαπλά ορίσματα σε μια νέα συνάρτηση η οποία λαμβάνει ένα όρισμα**.

Σύνταξη Συνάρτησης με πολλαπλά ορίσματα: `def function_name(argument1, argument2) = operation`

Παράδειγμα

Πρόγραμμα Scala που αθροίζει 2 αριθμούς (Χωρίς τη μέθοδο currying)

```
object Curry
{
    //Ορισμός currying function
    def add(x: Int, y: Int) = x + y;

    def main(args: Array[String])
    {
        println(add(20, 19));
    }
}
```

Αποτέλεσμα Εκτέλεσης: 39

Η συνάρτηση add λαμβάνει 2 ορίσματα (x, y) και τα αθροίζει και δίνει ως αποτέλεσμα το άθροισμα τους

Α τρόπος δήλωσης της προηγούμενης συνάρτησης ως Curried function

Θέλουμε να μετατρέψουμε τη συνάρτηση add που λαμβάνει δύο ορίσματα (δηλ. πολλαπλά ορίσματα) σε μια συνάρτηση η οποία λαμβάνει ένα (μοναδικό) όρισμα:

Σύνταξη Συνάρτησης με 1 όρισμα: `def function_name(argument1) = (argument2) => operation`

Παράδειγμα 1

Πρόγραμμα Scala που αθροίζει 2 αριθμούς χρησιμοποιώντας currying function

```
object Curry
{
    def add2(a: Int) = (b: Int) => a + b; // μετατρέπουμε τη συνάρτηση που λαμβάνει δύο (δηλ. πολλαπλά) ορίσματα σε μια συνάρτηση η οποία λαμβάνει ένα (δηλ. μοναδικό) όρισμα

    //Main
    def main(args: Array[String])
    {
        println(add2(20)(19));
    }
}
```

Αποτέλεσμα 39

Επεξήγηση: Εδώ ορίζουμε τη συνάρτηση add2 που λαμβάνει μόνο ένα όρισμα αρχικά (το a). Η 2^η κλήση θα πάρει ένα όρισμα (το b) και θα επιστραφεί ως αποτέλεσμα το a+b. **Άρα έχουμε κάνει curried τη συνάρτηση add**, που σημαίνει ότι την έχουμε μετατρέψει από συνάρτηση που λαμβάνει 2 ορίσματα σε συνάρτηση που λαμβάνει ένα όρισμα και στο τέλος η συνάρτηση add2 επιστρέφει το άθροισμα των ορισμάτων της

Β τρόπος δήλωσης της προηγούμενης ως Curried function -OXI

Μερικώς Εφαρμοζόμενη Curried Συνάρτηση (Currying Function Using Partial Application)

Υπάρχει ακόμα ένας τρόπος να ορίσουμε την Curried function και αυτό ονομάζεται **Partially Applied function** (μερικώς εφαρμοζόμενη συνάρτηση). Ας δούμε το επόμενο παράδειγμα όπου έχουμε ορίσει μια μεταβλητή sum στο main.

Παράδειγμα 2

Πρόγραμμα Scala program που αθροίζει 2 αριθμούς χρησιμοποιώντας τη currying function

object Curry

{

```
def add2(a: Int) = (b: Int) => a + b; // μετατρέπουμε τη συνάρτηση που λαμβάνει δύο (δηλ. πολλαπλά) ορίσματα σε μια συνάρτηση  
η οποία λαμβάνει ένα (δηλ. μοναδικό) όρισμα
```

//Main

```
def main(args: Array[String])
```

{

//μερικώς εφαρμοζόμενη συνάρτηση

```
val sum=add2(29);
```

```
println(sum(5));
```

}

}

Αποτέλεσμα 34

Επεξήγηση Εδώ μεταβιβάζεται μόνο ένα όρισμα όταν η συνάρτηση add2 καλείται. Το 2^o όρισμα μεταβιβάζεται με την τιμή και αυτά τα ορίσματα αθροίζονται και εκτυπώνεται το τελικό αποτέλεσμα

Γ τρόπος δήλωσης της προηγούμενης ως Curried function -OXI

Σύνταξη def function name(argument1) (argument2) = operation

Παράδειγμα 3

Πρόγραμμα Scala που αθροίζει 2 αριθμούς χρησιμοποιώντας την currying function.

object Curry

{

```
def add2(a: Int) (b: Int) = a + b; //Δήλωση Currying function
```

```
def main(args: Array[String])
```

{

```
    println(add2(29)(5));
```

}

}

Αποτέλεσμα 34

Γιαυτή τη σύνταξη η μερικώς εφαρμοζόμενη συνάρτηση (Partial Application function) γράφεται διαφορετικά.

Παράδειγμα 4

Πρόγραμμα Scala program που αθροίζει 2 αριθμούς χρησιμοποιώντας την currying function.

object Curry

{

//Δήλωση Currying function

```
def add2(a: Int) (b: Int) = a + b;
```

```
def main(args: Array[String])
```

{

```
    val sum=add2(29)_;
```

```
    println(sum(5));
```

}

}

Αποτέλεσμα 34

Εδώ μόνο το ‘_’ αθροίζεται μετά την κλήση της συνάρτησης add2 για την τιμή sum.

Πως εφαρμόζονται οι συναρτήσεις που επιστρέφουν συνάρτηση;

Απάντηση

Αν τα args1 και args2 είναι λίστες ορισμάτων τότε η έκφραση $f(args1)(args2)$ είναι ισοδύναμη με την έκφραση $(f(args1))(args2)$. Για παράδειγμα στην έκφραση:

sum(x => x * x)(1, 10)

η συνάρτηση sum εφαρμόζεται στη συνάρτηση τετραγώνου ($x \Rightarrow x * x$). Η συνάρτηση αποτέλεσμα εφαρμόζεται στη $2^{\text{η}}$ λίστα ορισμάτων (1, 10). Αυτός ο συμβολισμός είναι δυνατός διότι η εφαρμογή των συναρτήσεων σχετίζεται με το αριστερό όρισμα.

Στο παράδειγμα μας η έκφραση $\text{sum}(x \Rightarrow x * x)(1, 10)$ είναι ισοδύναμη με την έκφραση $(\text{sum}(x \Rightarrow x * x))(1, 10)$

Υπολογίζεται η παράσταση $1^2 + 2^2 + 3^2 + \dots + 10^2$

8 Συναρτησιακός Προγραμματισμός (Functional Programming)

8.1 Μειώσεις-Ελαττώσεις (Reductions)

Εκτελώντας ένα συναρτησιακό πρόγραμμα (functional program) δηλ. αξιολογώντας μια έκφραση σημαίνει ότι εφαρμόζονται επαναληπτικά ορισμοί συναρτήσεων μέχρις ότου επεκταθούν όλες οι εφαρμογές συναρτήσεων. Οι υλοποιήσεις των μοντέρνων γλωσσών συναρτησιακού προγραμματισμού βασίζονται σε μια τεχνική απλοποίησης που ονομάζεται αναγωγή (reduction).

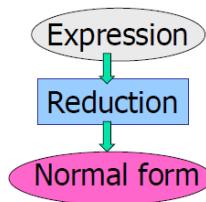
Τι είναι η Μείωση;

Δοθέντων ενός συνόλου κανόνων R_1, R_2, \dots, R_n (που ονομάζονται κανόνες αναγωγής) και μιας έκφρασης e , Αναγωγή είναι η διαδικασία της επαναληπτικής απλοποίησης της e χρησιμοποιώντας τους δοσμένους κανόνες αναγωγής:

$$\begin{aligned} e &\Rightarrow e_1 \quad R_{i1} \\ &\Rightarrow e_2 \quad R_{i2} \\ &\dots \\ &\Rightarrow e_k \quad R_{ik}, \quad R_{ij} \in \{R_1, R_2, \dots, R_n\} \end{aligned}$$

μέχρι να μην μπορεί να εφαρμοστεί κανένας κανόνας.

Το e_k ονομάζεται **κανονική μορφή** (normal form) της e . Είναι η πιο απλή μορφή της e .



Υπάρχουν δύο τύποι κανόνων αναγωγής:

Ενσωματωμένοι Κανόνες

Πρόσθεση, Αφαίρεση, Πολλαπλασιασμός, Διαιρεση

Κανόνες Χρήστη

$$\begin{aligned} \text{square } x &= x * x \\ \text{double } x &= x + x \\ \text{sum []} &= 0 \\ \text{sum (x:xs)} &= x + \text{sum xs} \\ f x y &= \text{square } x + \text{square } y \end{aligned}$$

Κάθε βήμα αναγωγής αντικαθιστά μια υποέκφραση με μια ισοδύναμη έκφραση εφαρμόζοντας ένα από δύο τύπους κανόνων:

$$\begin{aligned} f x y &= \text{square } x + \text{square } y \\ f 3 4 &\Rightarrow (\text{square } 3) + (\text{square } 4) && (f) \\ &\Rightarrow (3 * 3) + (\text{square } 4) && (\text{square}) \\ &\Rightarrow 9 + (\text{square } 4) && (*) \\ &\Rightarrow 9 + (4 * 4) && (\text{square}) \\ &\Rightarrow 9 + 16 && (*) \\ &\Rightarrow 25 && (+) \end{aligned}$$

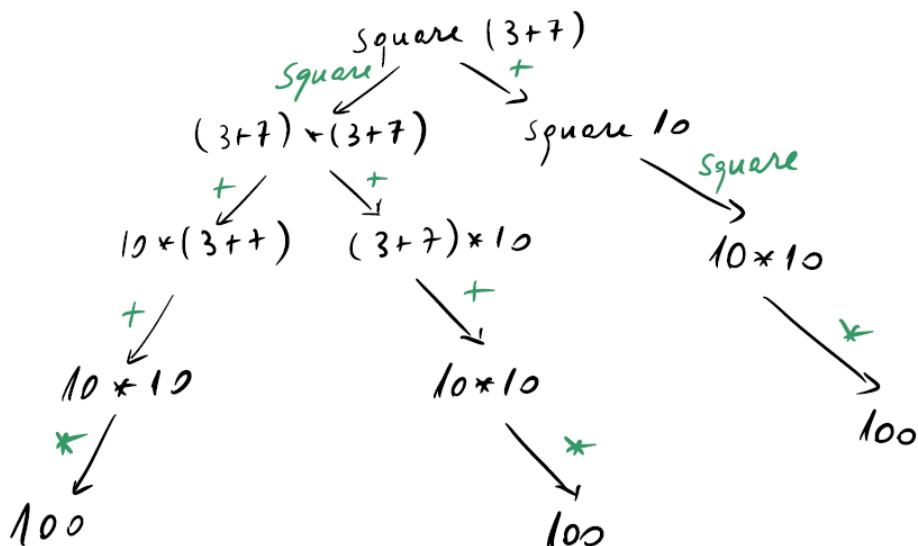
Κάθε αναγωγή αντικαθιστά μια υποέκφραση που ονομάζεται αναγώγιμη έκφραση (reducible expression ή redex) με μια ισοδύναμη είτε με κλήση συνάρτησης (π.χ. τη `square`) είτε χρησιμοποιώντας μια ενσωματωμένη συνάρτηση (π.χ. τη `+`)

Μια έκφραση χωρίς redexes λέμε ότι βρίσκεται σε κανονική μορφή (normal form). Όσο λιγότερες είναι οι αναγωγές που πρέπει να εκτελεστούν τόσο πιο γρήγορα τρέχει το πρόγραμμα.

8.2 Εναλλακτικές Μειώσεις

$\text{square}(3+7)$	$\text{square}(3+7)$
$\text{square}(3+7)$ $= \text{square}(10)$ (+) $= 10 * 10$ square $= 100$ (*) Normal form	$\text{square}(3+7)$ $= (3+7) * (3+7)$ (square) $= 10 * (3+7)$ (+) $= 10 * 10$ (*) $= 100$ Normal form
	Reduction rules

8.3 Διαφορετικοί Τρόποι Υπολογισμού του square



Ποια είναι η βέλτιστη διαδρομή; Αυτή που βρίσκει τη συντομότερη διαδρομή δηλαδή αυτή που κάνει τις λιγότερες αναγωγές

Υπάρχει αλγόριθμος που να βρίσκει πάντα τη βέλτιστη διαδρομή; Υπάρχουν δύο τέτοιες στρατηγικές:

- **H Eager Evaluation Strategy**
- **H Lazy Evaluation Strategy**

8.4 Eager Evaluation Strategy vs Lazy Evaluation

Δοθείσης μια έκφρασης της μορφής: f α όπου f συνάρτηση και α όρισμα, η στρατηγική Eager Evaluation μειώνει μια τέτοια έκφραση προσπαθώντας να εφαρμόσει πρώτα τον ορισμό της f . Αντίθετα η στρατηγική Lazy Evaluation μειώνει την έκφραση προσπαθώντας να απλοποιήσει πρώτα το όρισμα α

8.4.1 Πιθανό Θέμα: Παράδειγμα 1 με Eager Evaluation και Lazzy Evaluation

Eager Evaluation	Lazzy Evaluation
$= \text{square}(3+7) \rightarrow 1$ αναγωγή $=\text{square}(10) \rightarrow 1$ αναγωγή $10*10 \rightarrow 1$ αναγωγή $=100$	$= \text{square}(3+7) \rightarrow 1$ αναγωγή $(3+7)*(3+7) \rightarrow 1$ αναγωγή $= 10*(3+7) \rightarrow 1$ αναγωγή $= 10*10 \rightarrow 1$ αναγωγή 100
Η στρατηγική Eager Evaluation εκτελεί 3 μειώσεις Η στρατηγική Eager Evaluation είναι πιο αποδοτική διότι Η στρατηγική Eager Evaluation ακολουθεί inner-most μείωση δηλαδή μειώνει πρώτα τα πιο εσωτερικά redexes. Ένα εσωτερικό redex (innermost redex) είναι ένα redex που δεν περιλαμβάνει άλλο redex ως εσωτερική υποέκφραση	Η στρατηγική Lazy Evaluation απαιτεί 4 μειώσεις Η στρατηγική Lazy Evaluation ακολουθεί outer-most μείωση δηλαδή μειώνει πρώτα τα πιο εξωτερικά redexes. Ένα εξωτερικό redex (Outermost redex) είναι ένα redex που δεν βρίσκεται μέσα σε άλλο redex.

8.4.2 Πιθανό Θέμα: Παράδειγμα 2 με Eager Evaluation και Lazzy Evaluation

Να υπολογιστεί η έκφραση $\text{square}(\text{sum}[1:100])$ με Eager και Lazzy Evaluation και να υπολογιστεί σε κάθε περίπτωση το πλήθος των μειώσεων

Eager Evaluation	Lazzy Evaluation
$= \text{square}(\text{sum}[1:100]) \rightarrow 100$ αναγωγές $=\text{square}(5050) \rightarrow 1$ αναγωγή $5050*5050 \rightarrow 1$ αναγωγή $=25502500$	$= \text{square}(\text{sum}[1:100])$ $\text{sum}[1:100]*\text{sum}[1:100] \rightarrow 1$ αναγωγή $= 5050*\text{sum}[1:100] \rightarrow 100$ αναγωγές $= 5050*5050 \rightarrow 100$ αναγωγές $25502500 \rightarrow 1$ αναγωγή
Η στρατηγική Eager Evaluation εκτελεί 102 μειώσεις Η στρατηγική Eager Evaluation είναι πιο αποδοτική διότι δεν επανέλαβε τη μείωση της υποέκφρασης $\text{sum}[1..100]$	Η στρατηγική Lazy Evaluation απαιτεί 202 μειώσεις

8.4.3 Πιθανό Θέμα: Παράδειγμα 3-Εκτέλεση Περιττών Υπολογισμών

first(2+2, square 15)

Eager Evaluation	Lazzy Evaluation
first(2+2, square 15) $= \text{first}(4, \text{square } 15) (+)$ $= \text{first}(4, 225) (\text{square})$ $4 (\text{first})$	first(2+2, square 15) $= 2+2 (\text{first})$ $= 4 (+)$
Η στρατηγική Eager Evaluation εκτελεί 3 μειώσεις	Η στρατηγική Lazy Evaluation απαιτεί 2 μειώσεις
	Η στρατηγική Lazy Evaluation εδώ είναι πιο αποδοτική διότι αποφεύγει την εκτέλεση περιττών υπολογισμών

8.4.4 Πλεονεκτήματα-Μειονεκτήματα

Eager Evaluation	Πλεονεκτήματα Αποφεύγει τις επαναλαμβανόμενες μειώσεις Υπο-εκφράσεων
	Μειονεκτήματα Πρέπει να υπολογίσει όλες τις παραμέτρους σε μια κλήση συνάρτησης ανεξάρτητα από το αν αυτές χρειάζονται ή όχι στον υπολογισμό του τελικού αποτελέσματος Μπορεί να μην τερματίζει
Lazy Evaluation	Πλεονεκτήματα Μια υποέκφραση δεν μειώνεται παρά μόνο αν είναι εντελώς απαραίτητη για την παραγωγή του τελικού αποτελέσματος Αν μια υπάρχει μια οποιαδήποτε σειρά μειώσεων η οποία τερματίζει τότε και η Lazy Evaluation τερματίζει
	Μειονεκτήματα Οι μειώσεις κάποιων υποεκφράσεων μπορεί να επαναλαμβάνονται αχρείαστα

8.4.5 Διπλές Μειώσεις Υποεκφράσεων

- Η μείωση της υποέκφρασης (3+4) είναι διπλή αν εφαρμόσουμε τη στρατηγική lazy function για τη μείωση της square (3+4)
- Το πρόβλημα εμφανίζεται σε οποιοδήποτε ορισμό όπου μια μεταβλητή στο αριστερό του μέρος εμφανίζεται περισσότερες από μια φορές στο δεξιό μέρος

square x = x * x

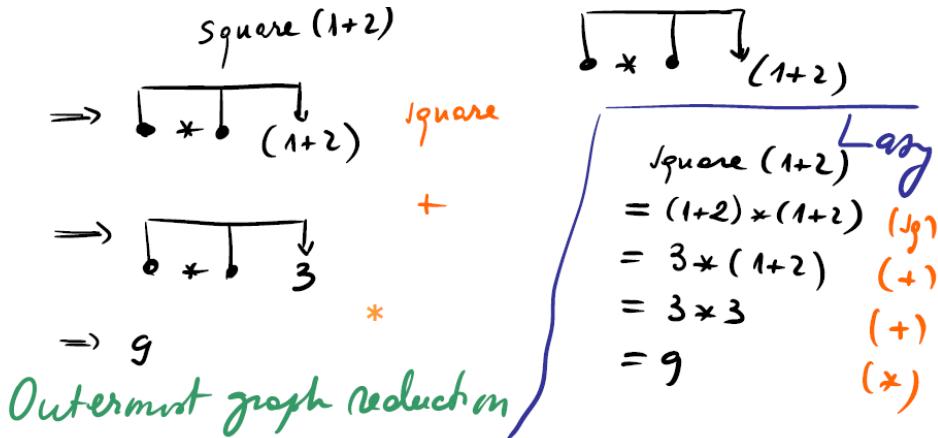
cube x = x * x * x

8.5 Στρατηγική Μείωσης Γράφου (Graph Reduction)

Σκοπός: Η Διατήρηση όλων των καλών χαρακτηριστικών της Lazy Evaluation με ταυτόχρονη αποφυγή διπλών μειώσεων υποεκφράσεων

Μέθοδος: Η αναπαράσταση εκφράσεων με τη μορφή Γράφου έτσι ώστε όλες οι εμφανίσεις μιας μεταβλητής να δείχνουν στην ίδια τιμή

8.5.1 Παράδειγμα με μείωση Γράφου (Graph Reduction)



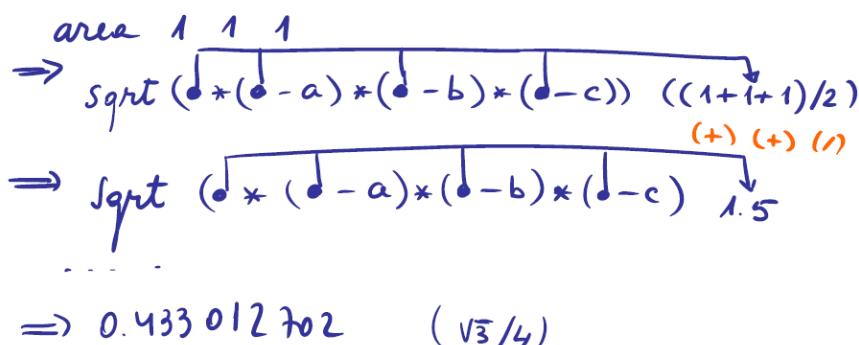
Η στρατηγική Μείωσης Γράφου (Graph Reduction) συνδυάζει όλα τα πλεονεκτήματα και των 2 στρατηγικών (Eager και Lazy evaluation) χωρίς κανένα από τα μειονεκτήματα τους

Η εξωτερική μείωση του γράφου (outermost graph reduction) του $\text{square}(3+4)$ μειώνει κάθε όρισμα το πολύ κατά ένα. Γιαυτό το λόγο απαιτεί λιγότερα βήματα μείωσης από την εσωτερική μείωση.

8.5.2 Μείωση Γράφου για την εντολή let

Ο τύπος του Ήρωα για το εμβαδόν τριγώνου με πλευρές a , b , και c .

$$\begin{aligned}
 \text{area } a \ b \ c = & \text{let } s = (a+b+c)/2 \text{ in} \\
 & \text{sqrt}(s*(s-a)*(s-b)*(s-c))
 \end{aligned}$$



Οι συνδέσεις let (let-bindings) απλά δίνουν ονόματα στους κόμβους του γράφου.

Παρατήρηση

Ο αριθμός των βημάτων μείωσης για να φτάσουμε στην κανονική μορφή (normal form) αντιστοιχεί στο χρόνο εκτέλεσης και το μέγεθος των όρων του γράφου που αντιστοιχούν στην χρησιμοποιούμενη μνήμη.

8.5.3 Μειώσεις για συναρτήσεις υψηλότερης τάξης και currying

$$\begin{aligned} \text{id } x &= x \\ a &= \text{id } (+1) 41 \\ \text{twice } f &= f . f \\ b &= \text{twice } (+1) (13^*3) \end{aligned}$$

To id και το twice ορίζονται μόνο με ένα όρισμα (+1). Η λύση είναι να δούμε πως τα πολλαπλά ορίσματα απλοποιούνται σε ένα όρισμα-curryng

Currying

$$\begin{aligned} \text{id } x &= x \\ a &= \text{id } (+1) 41 \\ \text{twice } f &= f . f \\ b &= \text{twice } (+1) (13^*3) \end{aligned}$$

$$a = (\text{id } (+1)) 41$$

$$b = (\text{twice } (+1)) (13^*3)$$

Για να μειώσουμε μια τυχαία εφαρμογή $expression1\ expression2$ πρώτα μειώνουμε την $expression1$ μέχρις ότου να γίνει μια συνάρτηση της οποίας ο ορισμός μπορεί να ξεδιπλωθεί με το όρισμα $expression2$.

$$a = (\text{id } (+1)) 41$$

a

$$\begin{aligned} &\Rightarrow (\text{id } (+1)) 41 && (\text{a}) \\ &\Rightarrow (+1) 41 && (\text{id}) \\ &\Rightarrow 42 && (+) \end{aligned}$$

$$b = (\text{twice } (+1)) (13^*3)$$

b

$$\begin{aligned} &\Rightarrow (\text{twice } (+1)) (13^*3) && (\text{b}) \\ &\Rightarrow ((+1).(+1)) (13^*3) && (\text{twice}) \\ &\Rightarrow (+1) ((+1) (13^*3)) && (.) \\ &\Rightarrow (+1) ((+1) 39) && (*) \\ &\Rightarrow (+1) 40 && (+) \\ &\Rightarrow 41 && (+) \end{aligned}$$

9 Συναρτησιακός Προγραμματισμός – Lisp -Scheme

9.1 Έννοιες Συναρτησιακού Προγραμματισμού

Οι γλώσσες συναρτησιακού προγραμματισμού όπως η Lisp, η Scheme, η FP, η ML, η Miranda, και η Haskell είναι μια προσπάθεια να πραγματωθεί ο λογισμός Λάμβδα του Church σε μια πρακτική μορφή σαν γλώσσα προγραμματισμού.

Η βασική ιδέα: τα πάντα γίνονται με τη σύνθεση συναρτήσεων

- ◆ δεν υπάρχει μεταβλητή κατάσταση
- ◆ δεν υπάρχουν παρενέργειες

Πώς κάνουμε κάτι σε μια συναρτησιακή γλώσσα;

Η αναδρομή (ειδικά η αναδρομή ουράς) αντικαθιστά την επανάληψη

Γενικά, μπορούμε να έχουμε το ίδιο αποτέλεσμα με μια σειρά αναθέσεων

- x := 0 ...
- x := expr1 ...
- x := expr2 ...

με την f3(f2(f1(0))) όπου κάθε f περιμένει την τιμή του x σαν παράμετρο, η f1 επιστρέφει expr1, και η f2 επιστρέφει expr2

Πιθανό Θέμα

Η αναδρομή αντικαθιστά με επιτυχία ακόμα και τους βρόχους

```
x=0; i:=1; j:=100;  
while i <j do  
    x :=x +i*j; i:=i +1; j:=j-1  
end while  
return x  
  
γίνεται f(0, 1, 100) όπου  
f(x, i, j) == if i < j then  
    f (x+i*j, i+1, j-1)  
else x
```

Η Lisp έχει επίσης τα εξής (δεν τα έχουν όλες οι συναρτησιακές γλώσσες)

- ◆ ομοεικονική
- ◆ αυτό-ορισμός
- ◆ ανάγνωση-αποτίμηση-εκτύπωση

Εκδόσεις της LISP

- ◆ Αμιγής Lisp (η πρώτη Lisp)
- ◆ Interlisp, MacLisp, Emacs Lisp
- ◆ Common Lisp
- ◆ Scheme

Όπως αναφέρθηκε, η Scheme είναι μια ιδιαίτερα κομψή Lisp

- ♦ Ο διερμηνέας εκτελεί έναν βρόχο ανάγνωσης- αποτίμησης-εκτύπωσης
- ♦ Ότι εισάγεται στο διερμηνέα αποτιμάται (αναδρομικά) μόνο μια φορά
- ♦ Αν κάτι είναι μέσα σε παρενθέσεις, είναι κλήση συνάρτησης (εκτός αν είναι σε παράθεση)
- ♦ Οι παρενθέσεις ΔΕΝ υπάρχουν μόνο για ομαδοποίηση, όπως στις γλώσσες της οικογένειας της Algol

Η προσθήκη ενός επιπέδου παρενθέσεων αλλάζει τη σημασία

(+ 3 4) $\Rightarrow 7$

((+ 3 4)) $\Rightarrow \text{error}$

(το βέλος ' \Rightarrow ' σημαίνει «αποτιμάται σε»)

9.2 Σύνοψη της Scheme

- Λογικές τιμές #t και #f
- Αριθμοί
- Εκφράσεις Λάμβδα
- Παράθεση

(+ 3 4) $\Rightarrow 7$

(quote (+ 3 4)) $\Rightarrow (+ 3 4)$

'(+ 3 4) $\Rightarrow (+ 3 4)$

- Μηχανισμοί για τη δημιουργία νέων εμβελειών

(let ((square (lambda (x) (* x x))) (plus +))

(sqrt (plus (square a) (square b))))

let*

letrec

Εκφράσεις συνθήκης στη Scheme

Πιθανό Θέμα

(if (< 2 3) 4 5) $\Rightarrow \textcolor{red}{4}$

Πιθανό Θέμα

(cond

((< 3 2) 1)

((< 4 3) 2)

(else 3)) $\Rightarrow \textcolor{red}{3}$

9.3 Θεωρητικές Βάσεις: Λογισμός - λ

Λογισμός Λάμβδα: Μια σημειογραφία/μοντέλο υπολογισμών που βασίζεται στον αμιγή συντακτικό χειρισμό συμβόλων, τα πάντα είναι συναρτήσεις

9.4 Επεξεργασία Big Data και MapReduce

Καταγραφή

Web Requests Logs → ποιες ήταν σήμερα οι πιο δημοφιλείς queries στο internet;

Πόσα κλικ σε ads έκαναν σήμερα οι χρήστες;

Απλοποίηση

Η **MapReduce** είναι το μοντέλο απλοποίησης για επεξεργασία δεδομένων υψηλής κλίμακας που εισάγεται από τη γλώσσα LISP

9.5 Συναρτησιακός Προγραμματισμός (Functional Programming)

- FP=υπολογισμός ως εφαρμογή συναρτήσεων
- Πως Διαφοροποιείται από τον κλασσικό προγραμματισμό;
 - Δεν υπάρχουν οι έννοιες των δεδομένων και των εντολών
 - Εκτέλεση του προγράμματος σημαίνει υπολογισμός των συναρτήσεων
 - Οι συναρτήσεις είναι ως επί το πλείστο μαθηματικές συναρτήσεις
 - Η κλήση των συναρτήσεων με τα ίδια ορίσματα θα δίνει πάντα το ίδιο αποτέλεσμα
 - Οι ροές δεδομένων είναι εσωτερικές
 - επιτρέπονται διαφορετικοί τρόποι εκτέλεσης
- Συναρτησιακές γλώσσες
 - Scala, ML, Haskell, Lisp κ.λ.π.

9.6 LISP

Οι λίστες είναι πρωτεύοντες τύποι δεδομένων

```
'(1 2 3 4 5)
'((a 1) (b 2) (c 3))
```

Οι συναρτήσεις γράφονται σε συμβολισμό prefix

```
(+ 1 2) → 3
(* 3 4) → 12
(sqrt (+ (* 3 3) (* 4 4))) → 5
(define x 3) → x
(* x 5) → 15
```

9.6.1 Συναρτήσεις

- Συναρτήσεις=Λάμβδα εκφράσεις περιορισμένες σε μεταβλητές

```
(define foo  
  (lambda (x y)  
    (sqrt (+ (* x x) (* y y)))))
```

- Η προηγούμενη έκφραση είναι ισοδύναμη με την:

```
(define (foo x y)  
  (sqrt (+ (* x x) (* y y))))
```

Πιθανό Θέμα: Ποιο το αποτέλεσμα της κλήσης (foo 3 4);

Απάντηση

(foo 3 4)→5

9.6.2 Άλλα Χαρακτηριστικά

- Στη Lisp/Scheme το καθετί αποτελεί μια s-έκφραση (s-expression)
 - Δεν υπάρχει καμία διάκριση ανάμεσα στον κώδικα και στα δεδομένα
 - Είναι εύκολο να γραφεί αυτο-τροποποιούμενος κώδικας
- Συναρτήσεις Υψηλότερης Τάξης (High-order Function)

Συναρτήσεις που λαμβάνουν ως όρισμα άλλες συναρτήσεις

(**define** (bar f x) (**f (f x)**))

Η bar είναι όνομα συνάρτησης που λαμβάνει 2 ορίσματα: **To 1^o όρισμα της συνάρτησης bar είναι η συνάρτηση f** και **to 2^o όρισμα της συνάρτησης bar είναι η παράμετρος x**

Το σώμα της bar είναι το εξής: **πρώτα καλείται η f(x)**. Επιστρέφει αποτέλεσμα και μετά καλείται πάλι η f(αποτέλεσμα) ή f(f(x)) δηλ. καλείται μετά η εξωτερική f

(**define** (baz x) (* x x))

Η baz είναι όνομα συνάρτησης που λαμβάνει 1 ορίσμα (το x)

Το σώμα της baz είναι το εξής: Επιστρέφει ως αποτέλεσμα το x*x

Πιθανό Θέμα: Πόσο κάνει η παράσταση (bar baz 2)→?

Απάντηση

Η έκφραση (bar baz 2)→16 υπολογίζεται ως εξής:

- Η bar εκτελείται αρχικά με παραμέτρους **baz, 2**. Το baz μεταβιβάζεται στο f άρα η συνάρτηση f είναι η baz. Το 2 μεταβιβάζεται στο x
- Μετά γίνεται η 1^η εκτέλεση f x δηλ. **baz 2**. Επιστρέφεται η τιμή 2*2=4
- Μετά γίνεται η 2^η εκτέλεση f x δηλ. **baz 4**. Το baz μεταβιβάζεται στο f άρα η συνάρτηση f είναι η baz. Το 4 μεταβιβάζεται στο x
- Επιστρέφεται η τιμή 4*4=16

9.6.3 Αναδρομή σε Lisp

Πιθανό Θέμα: Δίνεται ο ακόλουθος αναδρομικός κώδικας σε Lisp για τον υπολογισμό του n!

```
(define (factorial n)
  (if (= n 1)
    1
    (* n (factorial (- n 1)))))
```

Να τον εκτελέσετε αναλυτικά για την κλήση (factorial 6)

Απάντηση

$ \begin{aligned} & (\text{factorial } 6) \rightarrow 6 * (\text{factorial } 5) \\ & 6 * 5 * (\text{factorial } 4) \\ & 6 * 5 * 4 * (\text{factorial } 3) \\ & 6 * 5 * 4 * 3 * (\text{factorial } 2) \\ & 6 * 5 * 4 * 3 * 2 * (\text{factorial } 1) \\ & 6 * 5 * 4 * 3 * 2 * 1 = 720 \end{aligned} $	Ισοδύναμος Κώδικας σε C <pre> factorial(int n) { if (n==1) return 1; else return factorial(n-1)*n; }</pre>
---	---

Πιθανό Θέμα: Δίνεται ο ακόλουθος επαναληπτικός κώδικας σε Lisp για τον υπολογισμό του n!

```
(define (factorial n)
  (define (aux n top product)
    (if (= n top)
        (* n product) //then return product*n
        (aux (+ n 1) top (* n product)))) //else product=product*n)
  (aux 1 n 1))
(factorial 6) → 720
```

Να τον εκτελέσετε αναλυτικά για την κλήση (factorial 6)

Απάντηση

$ \begin{aligned} & (\text{aux } 1 \text{ n } 1) \\ & (\text{factorial } 6) \rightarrow 1 * 2 * 3 * 4 * 5 * 6 = 720 \end{aligned} $	<pre> factorial(int top) { int product=1; for (n=1;n<=top; n++) product=product*n; return product; }</pre>
--	---

9.6.4 MapReduce

Δύο βασικές αρχές στο συναρτησιακό προγραμματισμό (Συναρτήσεις Υψηλότερης Τάξης Α κλάσης-first class high order functions):

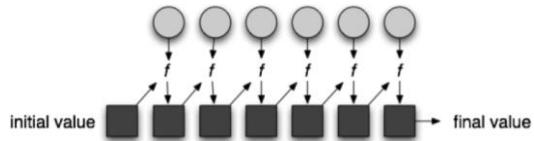
- **Map:** εκτελείται μια ενέργεια σε κάθε στοιχείο μιας λίστας
- **Fold:** συνδυάζονται με κάποιο τρόπο τα αποτελέσματα μιας λίστας

MAP

- Ο τρόπος λειτουργίας της Map είναι ότι εφαρμόζεται σε κάθε στοιχείο μιας λίστας και το αποτέλεσμα της είναι μια νέα λίστα

FOLD

- Ο τρόπος λειτουργίας της Fold είναι εφαρμόζει την πράξη σε όλη τη λίστα και επιστρέφει ένα τελικό αποτέλεσμα



Πιθανό Θέμα: Παραδείγματα Εφαρμογής MAP και FOLD

Παράδειγμα εφαρμογής της MAP	Παραδείγματα εφαρμογής της FOLD
<p><u>Η map εφαρμόζει τη συνάρτηση lambda σε κάθε στοιχείο της λίστας και επιστρέφει λίστα</u></p> <p style="color: red;">(map (lambda (x) (* x x)))</p> <p><u>Με είσοδο '(1 2 3 4 5) ποιο το αποτέλεσμα</u></p> <p style="color: red;"><u>Απάντηση</u></p> <p style="color: red;">(map (lambda (x) (* x x)) '(1 2 3 4 5) → '(1 4 9 16 25)</p> <p>Η γραμμή '(1 2 3 4 5) είναι τα δεδομένα εισόδου και σε κάθε στοιχείο της λίστας εκτελείται η πράξη x^*x και υπολογίζεται η λίστα εξόδου '(1 4 9 16 25)</p>	<p><u>Η fold εφαρμόζει την πράξη σε όλη τη λίστα και επιστρέφει ένα τελικό αποτέλεσμα.</u></p> <p style="color: red;">(fold + 0 '(1 2 3 4 5))--?</p> <p style="color: red;"><u>Απάντηση</u></p> <p style="color: red;">(fold + 0 '(1 2 3 4 5)) → 15</p> <p>Το + στην 1^η fold είναι η πράξη και το 0 είναι η αρχικοποίηση του αθροιστή,</p> <p style="color: red;">(fold * 1 '(1 2 3 4 5))--?</p> <p style="color: red;"><u>Απάντηση</u></p> <p style="color: red;">(fold * 1 '(1 2 3 4 5)) → 120</p> <p>Το * στη 2^η fold είναι η πράξη και το 1 είναι η αρχικοποίηση του πολλαπλασιαστή</p>

Πιθανό Θέμα: Παράδειγμα με συνδυασμό MAP και FOLD

```
(define (sum-of-squares y)
  (fold + 0 (map (lambda (x) (* x x)) y)))
```

(sum-of-squares '(1 2 3 4 5))--?

Απάντηση

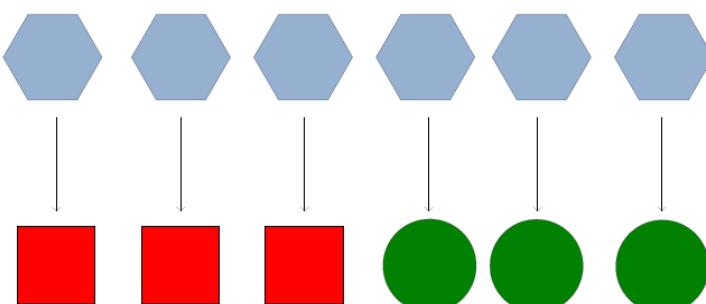
(sum-of-squares '(1 2 3 4 5)) → 55

- Καλείται η συνάρτηση με όνομα sum-of-squares και η λίστα (1 2 3 4 5) μεταβιβάζεται στο y.
- Μετά εκτελείται η map που εφαρμόζει την πράξη x^2 σε κάθε στοιχείο της λίστας εισόδου. Κάθε στοιχείο της λίστας εισόδου τοποθετεί κάθε αποτέλεσμα στο v και στο τέλος επιστρέφει τη λίστα (1 4 9 16 25)
- Στη συνέχεια η fold εφαρμόζει την πράξη $+ \text{ σε όλη τη λίστα}$ (1 4 9 16 25) που τη λαμβάνει ως όρισμα από τη map με αρχική τιμή αθροιστή 0 και υπολογίζει το συνολικό αθροισμα των τετραγώνων της λίστας αυτής που είναι $1+4+9+16+25=55$

Ο μηχανισμός MapReduce χρησιμοποιείται για επεξεργασία ενός μεγάλου όγκου δεδομένων (>1 TB) και για τη χρήση παραλληλισμού εκατοντάδων/χιλιάδων CPU. Προσφέρει αυτόματο παραλληλισμό και κατανομή δεδομένων.

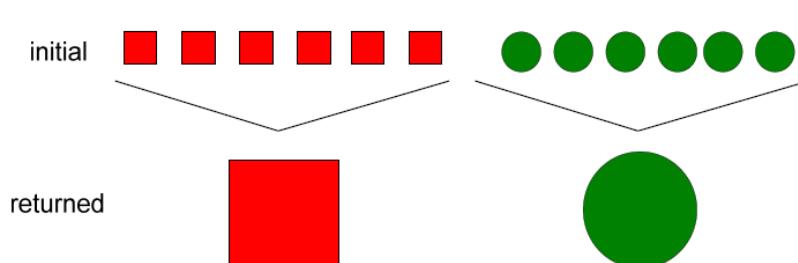
Συνάρτηση MAP

- Η συνάρτηση map απεικονίζει δεδομένα (π.χ. γραμμές από αρχεία, γραμμές από βάσεις δεδομένων κ.λ.π.) μέσω της map function ως **ζεύγη κλειδί, τιμή π.χ. (όνομα αρχείου, γραμμή)**



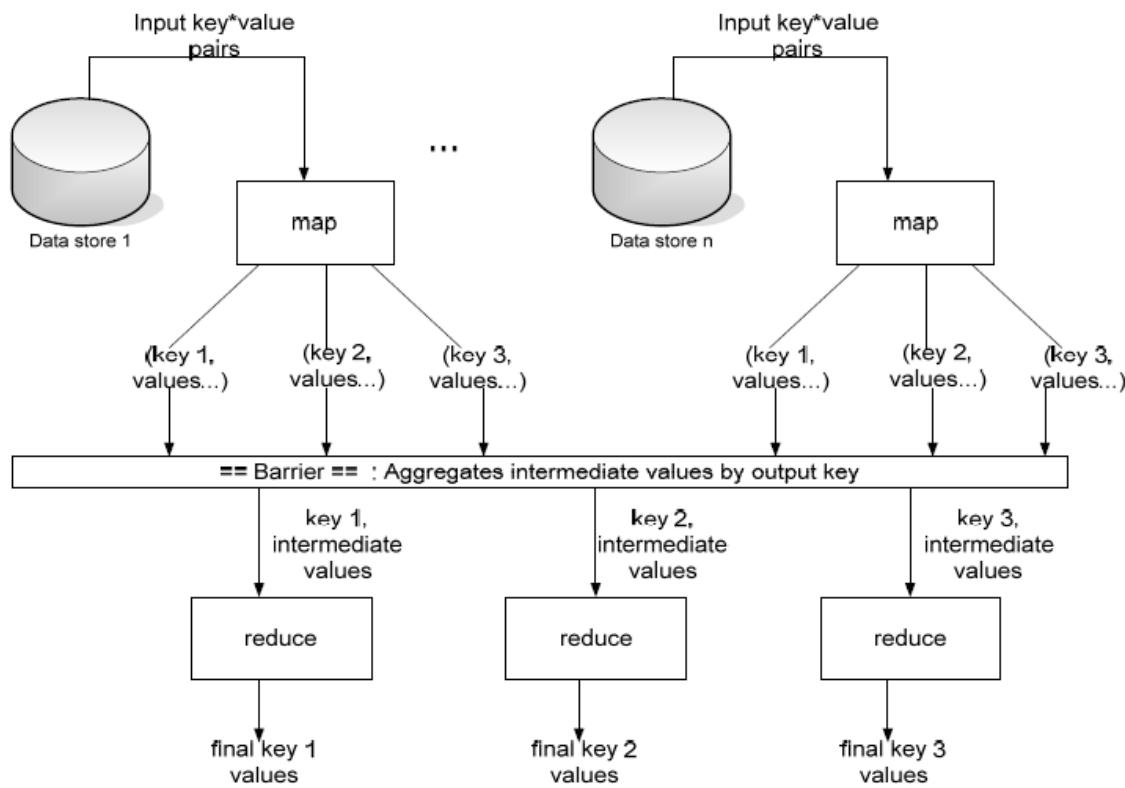
- Όταν τελειώσει η φάση του map όλες οι ενδιάμεσες τιμές για ένα δοθέν output key συνδυάζονται μαζί σε μια λίστα

Συνάρτηση Reduce



- Η συνάρτηση `reduce()` συνδυάζει αυτές τις ενδιάμεσες τιμές σε μια ή περισσότερες τελικές τιμές για το ίδιο κλειδί εξόδου (συνήθως μόνο μια τελική τιμή για κάθε κλειδί)

Γενικό Σχήμα Map-Reduce



Παραλληλισμός

- Οι συναρτήσεις `map()` λειτουργούν **παράλληλα** παράγοντας διαφορετικές ενδιάμεσες τιμές από διαφορετικά σύνολα δεδομένων εισόδου
- Οι συναρτήσεις `reduce()` λειτουργούν **παράλληλα**, η καθεμία σε ένα διαφορετικό κλειδί εξόδου
- Όλες οι τιμές επεξεργάζονται ανεξάρτητα
- Συμφόρηση (Bottleneck): Η φάση `reduce` δεν μπορεί να ξεκινήσει αν η φάση `map` δεν έχει ολοκληρωθεί πλήρως

9.7 ΘΕΜΑΤΑ ΣΕ ΣΥΝΑΡΤΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

9.7.1 Θέματα με Αποτίμηση Εκφράσεων με Fold και Map

9.7.1.1 Θέμα 5β Σεπτέμβριος 2019

define (sum-of-doubles y)

(**fold** + 0 (**map**(**lambda** (x) (* x x)) v))

(sum-of-doubles '(1 2 3 4 5)) → ?

Συμπληρώστε τις αποτιμήσεις των παραπάνω εκφράσεων

Απάντηση

sum-of-squares '(1 2 3 4 5) → 55

Παρατήρηση

Αιτιολόγηση: Η συνάρτηση sum-of-doubles λαμβάνει τη λίστα '(1 2 3 4 5), υπολογίζει με τη συνάρτηση lambda(x) τα τετράγωνα των στοιχείων της λίστας και στη συνέχεια η map επιστρέφει τη λίστα των τετραγώνων στη fold η οποία τα αθροίζει και υπολογίζει το συνολικό άθροισμα 55

Αναλυτική Αιτιολόγηση

1. Καλείται η συνάρτηση με όνομα **sum-of-doubles** και η λίστα (1 2 3 4 5) μεταβιβάζεται στο y
2. Μετά καλείται η συνάρτηση **lambda** (x) (* x x)) για κάθε στοιχείο της λίστας που είναι στο y
3. Κάθε στοιχείο της λίστας μεταβιβάζεται στο x και εκτελείται η πράξη x*x σε αυτό
4. Κάθε τετράγωνο καταχωρείται στο v (άρα στο v πάνε διαδοχικά οι τιμές 1*1=1, 2*2=4, 3*3=9, 4*4=16, 5*5=25) και δημιουργείται μια νέα λίστα με τη συνάρτηση map δηλ. δημιουργείται η λίστα τετραώνων (1 4 9 16 25) και στο τέλος επιστρέφεται η λίστα
5. Στη συνέχεια η fold εφαρμόζει την πράξη + σε όλη τη λίστα (1 4 9 16 25) που λαμβάνει ως όρισμα από τη map με αρχική τιμή αθροιστή 0 και υπολογίζει το συνολικό άθροισμα των στοιχείων της λίστας που λαμβάνει δηλ. 1+4+9+16+25=55

9.7.1.2 Θέμα 5α Ιούνιος 2022

a) Δίνεται ο κώδικας:

(**map** (**lambda** (x) (* x x)))

'(1 2 3 4 5))

→ ?

Τι θα εκτυπωθεί στο ?

Απάντηση

? → '(1 4 9 16 25)

Παρατήρηση

Αιτιολόγηση: Η συνάρτηση lambda λαμβάνει τη λίστα '(1 2 3 4 5), υπολογίζει τα τετράγωνα των στοιχείων της λίστας και στη συνέχεια η map επιστρέφει τη λίστα των τετραγώνων. Άρα το επιστρεφόμενο αποτέλεσμα είναι η λίστα '(1 4 9 16 25)

Αναλυτική Αιτιολόγηση

Η λίστα (1 2 3 4 5) μεταβιβάζεται στην παράμετρο x της συνάρτησης **lambda** (x) (* x x))

Στο κάθε στοιχείο της λίστας εφαρμόζεται η πράξη x^*x

Κάθε τετράγωνο καταχωρείται σε μια νέα λίστα λόγω της συνάρτησης map δηλ. στη νέα λίστα πάνε διαδοχικά οι τιμές $1*1=1$, $2*2=4$, $3*3=9$, $4*4=16$, $5*5=25$)

Στο τέλος επιστρέφεται η λίστα : '(1 4 9 16 25) από τη map

9.7.1.3 Θέμα 5a Ιούνιος 2023

Δίνεται ο κώδικας:

(fold + 0 '(1 2 3 4 5))→?

(fold * 1 '(1 2 3 4 5))→?

Τι θα εκτυπωθεί στα ?

Απάντηση

(fold + 0 '(1 2 3 4 5))→15

(fold * 1 '(1 2 3 4 5))→120

Αιτιολόγηση

Η fold εφαρμόζει την εκάστοτε πράξη σε όλη τη λίστα και υπολογίζει ένα τελικό αποτέλεσμα

Η (fold + 0 '(1 2 3 4 5)) εκτελεί την πράξη πρόσθεσης στη λίστα (1, 2, 3, 4, 5) με αρχική τιμή αθροιστή το 0 και υπολογίζει το συνολικό άθροισμα των στοιχείων της που είναι **15 διότι $0+1+2+3+4+5=15$**

Η ((fold * 1 '(1 2 3 4 5)) εκτελεί την πράξη του πολλαπλασιασμού στη λίστα (1, 2, 3, 4, 5) με αρχική τιμή πολλαπλασιαστή το 1 και υπολογίζει το συνολικό γινόμενο των στοιχείων της που είναι **120 διότι εκτελείται ο πολλαπλασιασμός $1*2*3*4*5=120$**

Παράδειγμα

fold + 2 '(1 2 3 4 5)=**2+1+2+3+4+5**

(fold * 2 '(1 2 3 4 5))= **$2*1*2*3*4*5$**

9.7.1.4 Θέμα 5b Ιούνιος 2022

define (sum-of-double v)

(fold + 0 (map(lambda (x) (* x x)) v)))

(sum-of-doubles '(1 2 3 4 5))→**55**

Παρατήρηση

Αιτιολόγηση: Η συνάρτηση sum-of-doubles λαμβάνει τη λίστα '(1 2 3 4 5), υπολογίζει με τη συνάρτηση lambda(x) τα τετράγωνα των στοιχείων της λίστας και στη συνέχεια η map επιστρέφει τη λίστα των τετραγώνων στη fold η οποία τα αθροίζει και υπολογίζει το 55

Αναλυτική Αιτιολόγηση

1. Καλείται η συνάρτηση με όνομα sum-of-doubles και η λίστα (1 2 3 4 5) μεταβιβάζεται στο y.

2. Μετά εκτελείται η map που καλεί και εκτελεί τη συνάρτηση **lambda** (x) (* x x)) σε κάθε στοιχείο της λίστας. Το κάθε στοιχείο της λίστας που έχει μεταβιβαστεί στο y καταχωρείται στο x και εφαρμόζεται η πράξη x^*x σε αυτό και το κάθε αποτέλεσμα καταχωρείται στο v (άρα στο v πάνε διαδοχικά οι τιμές $1*1=1$, $2*2=4$, $3*3=9$, $4*4=16$, $5*5=25$) και δημιουργείται μια νέα λίστα με τα αποτελέσματα αυτά δηλ. δημιουργείται η λίστα **(1 4 9 16 25)** και στο τέλος επιστρέφεται από τη map
3. Στη συνέχεια η fold εφαρμόζει την πράξη + σε όλη τη λίστα **(1 4 9 16 25)** που λαμβάνει ως όρισμα από τη map με αρχική τιμή αθροιστή 0 και υπολογίζει το συνολικό άθροισμα των στοιχείων της λίστας που λαμβάνει **$1+4+9+16+25=55$**

9.7.1.5 Θέμα 5c Ιούνιος 2022

Έστω σε Lisp οι παρακάτω Map/Fold (Reduce) high-order function:

a)

map(lambda (x) (^ x 3))

'(1 2 3 4))

→?

b)

(fold + 0 '(1 2 3 4)) →?

(fold * 1 '(1 2 3 4)) →?

c)

define (sum-of-cubes v)

(fold + 0 (map(lambda (x) (^ x 3)) v)))

(sum-of-cubes '(1 2 3 4)) →?

Με τι θα συμπληρώστε της αποτιμήσεις? Των παραπάνω εκφράσεων; Το \wedge συμβολίζει ύψωσε σε δύναμη

Απάντηση

a)

→(1^3, 2^3, 3^3, 4^3)= (1 8 27 16)

Παρατήρηση

Η λίστα μεταβιβάζεται στο x. Σε κάθε στοιχείο της λίστας εκτελείται η πράξη x^3

Η συνάρτηση map δημιουργεί μια νέα λίστα με της κύβους της αρχικής λίστας η οποία επιστρέφεται ως απάντηση

b)

Απάντηση

(fold + 0 '(1 2 3 4)) →**10**

(fold * 1 '(1 2 3 4)) →**24**

Παρατήρηση

Η fold εκτελεί την πράξη + στη λίστα **'(1 2 3 4)** με αρχική τιμή αθροιστή το 0 και υπολογίζει το άθροισμα της λίστας που είναι **$1+2+3+4=10$**

Η fold εκτελεί την πράξη * στη λίστα '(1 2 3 4) με αρχική τιμή πολλαπλασιαστή 1 και υπολογίζει το γινόμενο της λίστας που είναι $1*2*3*4=24$

c)

Απάντηση

(sum-of-cubes '(1 2 3 4)) → ?100

Παρατήρηση

Η λίστα (1 2 3 4) μεταβιβάζεται στην παράμετρο ν της συνάρτησης sum-of-cubes

Σε κάθε στοιχείο της λίστας εκτελείται η συνάρτηση lambda και υπολογίζει τον κύβο του από την πράξη x^3

Κάθε κύβος στοιχείου καταχωρείται σε μια νέα λίστα λόγω της συνάρτησης map δηλ. στη νέα λίστα πάνε διαδοχικά οι τιμές $1^3=1$, $2^3=8$, $3^3=27$, $4^3=64$)

Στο τέλος επιστρέφεται η λίστα '(1 8 27 64) από τη map

Η συνάρτηση fold εκτελεί την πράξη + στη λίστα '(1 8 27 64) με αρχική τιμή αθροιστή το 0 και υπολογίζει το άθροισμα της λίστας που είναι $1+8+27+64=100$

9.7.1.6 Θέμα 5c Ιούνιος 2021

Έστω σε Lisp οι παρακάτω Map/Fold (Reduce) high order functions:

```
(map (lambda (x) (* x x))
      '(1 2 3 4 5))
→ ?
(fold + 0 '(1 2 3 4 5)) → ?
(fold * 1 '(1 2 3 4 5)) → ?
(define (sum-of-squares v)
  (fold + 0 (map (lambda (x) (* x x)) v)))
(sum-of-squares '(1 2 3 4 5)) → ?
```

Συμπληρώστε τις αποτιμήσεις ? των παραπάνω εκφράσεων

Απάντηση

- στο 1^o fold το αποτέλεσμα είναι (fold + 0 '(1 2 3 4 5)) → $1+2+3+4+5=15$
- στο 2^o fold το αποτέλεσμα είναι (fold * 1 '(1 2 3 4 5)) → $1*2*3*4*5=120$
- στο 3^o fold το αποτέλεσμα είναι (sum-of-squares '(1 2 3 4 5)) → $1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$

9.7.1.7 Θέμα Ιούνιος 2022

Έστω σε Lisp οι παρακάτω Map/Fold (Reduce) high-order functions:

(**map** (*lambda* (x) (* x 2)))

'(1 2 3 4 5))

?

(**fold** + 0 '(1 2 3 4 5)) → ?

(**fold** * 1 '(1 2 3 4 5)) → ?

(**define** (sum-of-doubles v)

(**fold**+ 0 (**map** (*lambda* (x) (*x 2)) v)))

(sum-of-doubles '(1 2 3 4 5)) → ?

Τι θα συμπληρώνατε στις αποτιμήσεις των παραπάνω εκφράσεων;

Απάντηση

'(1 2 3 4 5) → (1 2 6 8 10)

(fold + 0 '(1 2 3 4 5)) → 15

(fold * 0 '(1 2 3 4 5)) → 120

(sum-of-doubles '(1 2 3 4 5)) → 2+4+6+8+10=30

9.7.1.8 Πιθανό Θέμα - Σχήματα σε Lisp (Lis Scheme) με lambda expressions

(**define** foo

(*lambda* (x y)

(**sqrt** (+ (* x x)(* y y))))))

(**define** (foo x y)

(**sqrt** (+ (* x x)(* y y))))

Δώστε το αποτέλεσμα εκτέλεσης της (foo 3 4) → ?

Απάντηση

(foo 3 4) → 5

Αιτιολόγηση Αποτελέσματος

Αν η foo έχει την 1^η μορφή με τη λambda συνάρτηση τότε:

το 3 μεταβιβάζεται στην παράμετρο x και το 4 μεταβιβάζεται στην παράμετρο y δηλ. στην κλήση (foo 3 4) το 3 → x και το 4 → y εκτελείται η συνάρτηση (lambda(3, 4)

(* 3 3) → 9

(* 4 4) → 16

(+ (* 3 3)(* 4 4)) → 16+9=25

(**sqrt** (+ (* 3 3)(* 4 4))) = 5

Αν η foo έχει τη 2^η μορφή χωρίς τη lambda συνάρτηση τότε:

το 3 μεταβιβάζεται στην παράμετρο x και το 4 μεταβιβάζεται στην παράμετρο y δηλ. στην κλήση (foo 3 4) το 3 →x και το 4→y

(* 3 3)→9

(* 4 4)→16

(+ (* 3 3)(* 4 4))→16+9=25

(sqrt (+ (* 3 3)+ (* 4 4)))=5

9.7.1.9 Πιθανό Θέμα - Σχήματα σε Lisp (List Scheme) με Map/Fold

<p>□ Simple map example:</p> <pre>(map (lambda (x) (* x x)) '(1 2 3 4 5)) → '(1 4 9 16 25)</pre>	<p>Η map εκτελείται σε όλες τις τιμές της λίστας</p> <p>Για κάθε τιμή της λίστας εκτελείται η lambda(x) που υπολογίζει και επιστρέφει το γινόμενο x^2</p> <p>Η map υπολογίζει τη λίστα ($1 \times 1 \rightarrow 1$, $2 \times 2 \rightarrow 4$, $3 \times 3 \rightarrow 9$, $4 \times 4 \rightarrow 16$, $5 \times 5 \rightarrow 25$).</p> <p>Στο τέλος θα δημιουργηθεί η λίστα '(1 4 9 16 25)</p>
<p>□ Fold examples:</p> <pre>(fold + 0 '(1 2 3 4 5)) → 15 (fold * 1 '(1 2 3 4 5)) → 120</pre>	<p>Η fold εφαρμόζει την πράξη σε όλη τη λίστα και επιστρέφει ένα τελικό Αποτέλεσμα.</p> <p>Το + στην 1η fold είναι η πράξη και το 0 είναι η αρχικοποίηση του αθροιστή και υπολογίζεται το αποτέλεσμα $1+2+3+4+5=15$</p> <p>Το * στη 2^η fold είναι η πράξη και το 1 είναι η αρχικοποίηση του πολλαπλασιαστή και υπολογίζεται το αποτέλεσμα $1 \times 2 \times 3 \times 4 \times 5=120$</p>
<p>□ Sum of squares:</p> <pre>(define (sum-of-squares v) (fold + 0 (map (lambda (x) (* x x)) v))) (sum-of-squares '(1 2 3 4 5)) → 55</pre>	<p>Καλείται η συνάρτηση sum-of-squares με όρισμα τη λίστα '(1 2 3 4 5). Η λίστα μεταβιβάζεται στην παράμετρο v.</p> <p>Μετά εκτελείται η lambda(x) που υπολογίζει για κάθε στοιχείο της λίστας v το τετράγωνο του στοιχείου από τον κώδικα x^2 και επιστρέφει τη λίστα '(1 4 9 16 25) στην fold. Μετά εφαρμόζεται η fold στη λίστα '(1 4 9 16 25) με την πράξη + και αρχική τιμή του αθροιστή στο 0 και υπολογίζει το άθροισμα $0+1+4+9+16+25=55$</p>

9.7.2 Διάφορα Θέματα

9.7.2.1 Θέμα 5γ Σεπτέμβριος 2019

a) Πως εκφράζεται η τεχνική currying στην παρακάτω εντολή σε γλώσσα scala;

sum(x=>x*x*x) (1, 1000000)

b) Πως σχετίζεται η τεχνική αυτή με το Map-Reduce μοντέλο προγραμματισμού που εφαρμόζεται στα συστήματα διαχείρισης δεδομένων μεγάλου όγκου

Απάντηση

```
def sum(f: Int => Int)(a: Int, b: Int): Int =
```

```
{
```

```
  def sumHelper(x: Int, acc: Int): Int = {
```

```
    if (x > b) acc
```

```
    else sumHelper(x + 1, acc + f(x))
```

```
}
```

```
  sumHelper(a, 0)
```

```
}
```

```
val result = sum(x => x * x * x) _
```

```
val finalResult = result(1, 1000000)
```

Ας δούμε πώς λειτουργεί αυτός ο κώδικας:

1. Η συνάρτηση sum δέχεται δύο ορίσματα:

- Μια συνάρτηση f, που παίρνει έναν ακέραιο και επιστρέφει έναν ακέραιο.
 - Δύο ακέραιους, a και b, που περιορίζουν το εύρος των ακεραίων που θα εφαρμοστεί η συνάρτηση f.
2. Στο εσωτερικό της συνάρτησης sum, υπάρχει η βιοηθητική συνάρτηση sumHelper που χρησιμοποιεί αναδρομή για να υπολογίσει το άθροισμα των τιμών της συνάρτησης f εφαρμοσμένης σε όλους τους ακέραιους από a έως b.
3. Η συνάρτηση sumHelper ελέγχει αν ο τρέχον ακέραιος x είναι μεγαλύτερος από το b. Αν ναι, επιστρέφει το άθροισμα acc, διαφορετικά συνεχίζει τον υπολογισμό αυξάνοντας τον ακέραιο x και ενημερώνοντας το άθροισμα acc κατά την εφαρμογή της συνάρτησης f στον τρέχοντα ακέραιο. Το acc είναι μια μεταβλητή που χρησιμοποιείται για να συγκεντρώσει τα ενδιάμεσα αποτελέσματα κατά τη διάρκεια του υπολογισμού. Αρχικά, η acc αρχικοποιείται στο 0. Καθώς η συνάρτηση sumHelper καλείται αναδρομικά, το acc αυξάνεται κατά το άθροισμα της τιμής που επιστρέφει η συνάρτηση f(x) για κάθε τιμή του x στο εύρος από a έως b. Κατά την εκτέλεση, το acc συσσωρεύει τα αποτελέσματα και τελικά επιστρέφεται ως το τελικό αποτέλεσμα της συνάρτησης sumHelper.
4. Η συνάρτηση sumHelper καλείται αρχικά με τα ορίσματα a και 0, και το αποτέλεσμα της κλήσης αυτής είναι το τελικό αποτέλεσμα της συνάρτησης sum.
5. Τέλος, δημιουργείται μια συνάρτηση result που χρησιμοποιεί τη συνάρτηση sum με τη συνάρτηση x => x * x * x ως πρώτο ορισμα και την καλεί με τα ορίσματα (1, 1000000) για να υπολογίσει το αποτέλεσμα. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή finalResult.

Η εντολή `sum(x => x * x * x)(1, 1000000)` στη γλώσσα Scala δημιουργεί μια συνάρτηση `sum` που παίρνει ως είσοδο μια λειτουργία (συνάρτηση) `x => x * x * x` και ένα εύρος ακεραίων από 1 έως 1000000. Στη συνέχεια, η συνάρτηση `sum` υπολογίζει το άθροισμα των τιμών που προκύπτουν από την εφαρμογή της λειτουργίας `x => x * x * x` σε κάθε ακέραιο από 1 έως 1000000.

Ακριβώς, η συνάρτηση `sum` εφαρμόζει τη λειτουργία `x => x * x * x` σε κάθε αριθμό από 1 έως 1000000 και στη συνέχεια πραγματοποιεί το άθροισμα όλων των αποτελεσμάτων. **Το αποτέλεσμα είναι το άθροισμα των κύβων των ακεραίων από 1 έως 1000000**

Παρατήρηση 2

Η τεχνική currying είναι ένας τρόπος γραφής συναρτήσεων στον οποίο μια συνάρτηση που αρχικά δέχεται πολλά ορίσματα τροποποιείται έτσι ώστε να δέχεται μικρότερα σύνολα ορισμάτων. Συγκεκριμένα, η currying μετατρέπει μια συνάρτηση που δέχεται δύο ή περισσότερα ορίσματα σε μια ακολουθία συναρτήσεων, κάθε μία από τις οποίες δέχεται ένα μόνο ορίσμα

b) Η τεχνική currying σχετίζεται με το Map-Reduce μοντέλο προγραμματισμού, καθώς αποτελεί ένα τρόπο διαχείρισης και επεξεργασίας δεδομένων που μπορεί να εφαρμοστεί σε συστήματα διαχείρισης δεδομένων μεγάλου όγκου. Ας εξετάσουμε πώς συσχετίζονται αυτά τα δύο:

1. **Τεχνική Currying:** Η τεχνική currying είναι ένας τρόπος που επιτρέπει στις συναρτήσεις να δέχονται ορίσματα σειριακά και να επιστρέφουν νέες συναρτήσεις ως αποτέλεσμα
2. **Map-Reduce Μοντέλο:** Το Map-Reduce είναι ένα μοντέλο προγραμματισμού που χρησιμοποιείται για την επεξεργασία μεγάλου όγκου δεδομένων σε κατανεμημένα συστήματα. Στο Map-Reduce, οι διαδικασίες χωρίζονται σε δύο κύρια στάδια:
 - **Map:** Στο στάδιο του Map, τα δεδομένα διαιρούνται σε τμήματα και εφαρμόζεται ένας αριθμός από συναρτήσεις Map σε κάθε τμήμα. Κάθε συνάρτηση Map δημιουργεί ανάλογα ζελυγή "κλειδί-τιμή" (key-value).
 - **Reduce:** Στο στάδιο του Reduce, τα αποτελέσματα από το Map συγκεντρώνονται και εφαρμόζεται μια συνάρτηση Reduce σε κάθε κλειδί, συνοψίζοντας τα δεδομένα

Σχετιζόμενα σημεία:

- Η τεχνική currying μπορεί να χρησιμοποιηθεί για να δημιουργήσει αλυσίδες επεξεργασίας δεδομένων, όπου κάθε στάδιο επεξεργασίας είναι μια συνάρτηση που δέχεται ένα ή περισσότερα ορίσματα και επιστρέφει μια νέα συνάρτηση ως αποτέλεσμα. Αυτό μπορεί να είναι χρήσιμο για τον προγραμματισμό των διαδικασιών Map και Reduce σε ένα Map-Reduce σύστημα.
- Στο Map-Reduce, η συνάρτηση Map εφαρμόζεται σε κάθε είσοδο και δημιουργεί 1 κλειδί-τιμή για κάθε είσοδο. Αυτή η διαδικασία μπορεί να γίνει πιο γενική και πιο ευέλικτη χρησιμοποιώντας την τεχνική currying για την εφαρμογή της συνάρτησης Map

9.7.3 Θέματα με σχήματα σε Lisp (Lis Scheme) με lambda expressions

9.7.3.1 Θέμα 5 Ομάδα Β Ιούνιος 2019

(α) Έστω σε γλώσσα Lisp/Scheme ο ορισμός της παρακάτω συνάρτησης υψηλής τάξης (high order function):

```
(define (bar f x) (f (f x)))  
#Doesn't matter what f is, just apply it twice  
(define (baz x) (* x x))  
(bar baz 2) → ?
```

Συμπληρώστε την αποτίμηση **?** της τελευταίας έκφρασης

(β) Ο απλός υπολογισμός **του παραγοντικού σε γλώσσα Lisp/Scheme δίνεται** από τον παρακάτω κώδικα:

```
(define (factorial n)  
  (if (= n 1)  
      1  
      (* n (factorial(- n 1))))))  
(factorial 6) → 720
```

Ακόμα και ο επαναληπτικός τρόπος γράφεται με αναδρομικές κλήσεις ως εξής:

```
(define (factorial-iter n)  
  (define (aux n top product)  
    (if (= n top)  
        1  
        (* n product)  
        (aux (+ n 1) top (* n product))))  
  (aux 1 n 1))  
(factorial-iter 6) → 720
```

Επιδείξτε τα βήματα εκτέλεσης του δεύτερου τρόπου για τον υπολογισμό του $6!=720$

Απάντηση

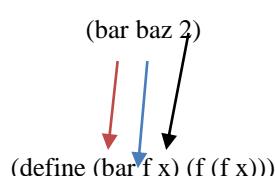
(a)

(bar baz 2) → 16

Αιτιολόγηση

Η **baz** είναι μια συνάρτηση που λαμβάνει μια τιμή στην παράμετρο **x** και επιστρέφει το $x*x$ μέσω του κώδικα **(define (baz x) (* x x))**

Στην κλήση **(bar baz 2)** γίνονται τα εξής:



Ο τρόπος λειτουργίας της **bar** είναι ο εξής: **(bar baz 2)**

Πρώτα **baz 2** → 4

Μετά **4** → 16

(b)

Εκτέλεση Α Τρόπου Υπολογισμού του $6!$ μόνο με αναδρομή (δεν ζητείται)

Στην κλήση (factorial 6) το 6 μεταβιβάζεται στο n

Η συνθήκη if ($= n 1$) είναι ψευδής γιαυτό εκτελείται το $(* 6 (\text{factorial}(5))))$

Στην κλήση (factorial 5) το 5 μεταβιβάζεται στο n

Η συνθήκη if ($= n 1$) είναι ψευδής γιαυτό εκτελείται το $(* 5 (\text{factorial}(4))))$

Στην κλήση (factorial 4) το 4 μεταβιβάζεται στο n

Η συνθήκη if ($= n 1$) είναι ψευδής γιαυτό εκτελείται το $(* 4 (\text{factorial}(3))))$

Στην κλήση (factorial 3) το 3 μεταβιβάζεται στο n

Η συνθήκη if ($= n 1$) είναι ψευδής γιαυτό εκτελείται το $(* 3 (\text{factorial}(2))))$

Στην κλήση (factorial 2) το 2 μεταβιβάζεται στο n

Η συνθήκη if ($= n 1$) είναι αληθής οπότε η συνάρτηση επιστρέφει το 1

$(* 2 (\text{factorial}(1))))$ Γίνεται η πράξη $2 * 1$ και η συνάρτηση επιστρέφει 2

$(* 3 (\text{factorial}(2))))$ Γίνεται η πράξη $3 * 2$ και η συνάρτηση επιστρέφει 6

$(* 4 (\text{factorial}(3))))$ Γίνεται η πράξη $4 * 6$ και η συνάρτηση επιστρέφει 24

$(* 5 (\text{factorial}(4))))$ Γίνεται η πράξη $5 * 24$ και η συνάρτηση επιστρέφει 120

$(* 6 (\text{factorial}(5))))$ Γίνεται η πράξη $6 * 120$ και η συνάρτηση επιστρέφει 720 που είναι το $6! = 1 * 2 * 3 * 4 * 5 * 6 = \textcolor{red}{720}$

Επεξήγηση Β Τρόπου Υπολογισμού του 6! δηλαδή του επαναληπτικού τρόπου υπολογισμού

με αναδρομικές κλήσεις (αυτό ζητείται)

1. Αρχικά, ορίζουμε μια συνάρτηση με το όνομα factorial-iter που παίρνει ένα όρισμα n. Αυτή η συνάρτηση θα χρησιμοποιηθεί για τον υπολογισμό του παραγοντικού του n.

(define (factorial-iter n)

2. Εντός της factorial-iter, ορίζουμε μια εσωτερική συνάρτηση με το όνομα aux. Αυτή η συνάρτηση θα χρησιμοποιηθεί για τον υπολογισμό του παραγοντικού και έχει τρία ορίσματα: n, top, και product.

(define (aux n top product)

3. Στη συνάρτηση aux, υπάρχει ένα if που ελέγχει εάν το n είναι ίσο με το top. Αυτό σημαίνει ότι φτάσαμε στον τελευταίο αριθμό που πρέπει να πολλαπλασιάσουμε για τον υπολογισμό του παραγοντικού. Αν είναι ίσοι, τότε επιστρέφουμε 1, δηλαδή το παραγοντικό του top

(if (= n top)

1

4. Αν το n δεν είναι ίσο με το top, τότε υπολογίζουμε το γινόμενο του n με το product και το αποτέλεσμα μεταβιβάζεται στη συνάρτηση aux με το n αυξημένο κατά 1 και το product ενημερωμένο με το νέο γινόμενο.

(* n product)

(aux (+ n 1) top (* n product))))

5. Τέλος, καλούμε τη συνάρτηση aux αρχικοποιώντας το n στο 1 (αφού ξεκινάμε τον υπολογισμό από το 1), το top στον αρχικό αριθμό n και το product στο 1.

(aux 1 n 1))

Έτσι, η συνάρτηση factorial-iter χρησιμοποιεί την εσωτερική συνάρτηση aux για να υπολογίσει το παραγοντικό του n και επιστρέφει το αποτέλεσμα. Καθώς καλείται αναδρομικά, η συνάρτηση aux αυξάνει το n κατά 1 και πολλαπλασιάζει το product με το τρέχον n μέχρι να φτάσει στον αρχικό αριθμό n.

Βηματική Εκτέλεση

1. Η aux καλείται με τις εξής τιμές:
 - n = 1
 - top = 6
 - product=1=1*1
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 1
2. Στην A επανάληψη, η aux καλείται ξανά με τις εξής τιμές:
 - n = 2
 - top = 6
 - product =2=1*2
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product)
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 2
3. Στη B επανάληψη, η aux καλείται ξανά με τις εξής τιμές:
 - n = 3
 - top = 6
 - product =6=1*2*3
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 6

4. Στη Γ επανάληψη, η αυχ καλείται ξανά με τις εξής τιμές:
- n = 4
 - top = 6
 - product = $24=1*2*3*4$
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 24
5. Στη Δ επανάληψη, η αυχ καλείται ξανά με τις εξής τιμές:
- n = 5
 - top = 6
 - product = $120=1*2*3*4*5$
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 120
6. Στην Ε επανάληψη, η αυχ καλείται ξανά με τις εξής τιμές:
- n = 6
 - top = 6
 - product= $720=1*2*3*4*5*6$
 - Η συνθήκη n=top είναι ψευδής γιαυτό επιστρέφεται το (*n product) που είναι 720
7. Τώρα, η συνθήκη (if (= n top) 1 είναι αληθής, επειδή το n είναι ίσο με το top (6). Επομένως, επιστρέφεται 1
8. Το τελικό αποτέλεσμα του παραγοντικού του 6 δηλ. 720 επιστρέφεται ως τελική απάντηση

9.7.4 Θέματα με σταδιακή αποτίμηση σε λ-λογισμό

9.7.4.1 Θέμα Ιούνιος 2022

Παρουσιάστε όλα τα βήματα σταδιακής αποτίμησης της παρακάτω έκφρασης σε λ-λογισμό:

$$(\lambda x. \lambda y. \lambda z. x + y + z) 3 4 5$$

Απάντηση
 $(\lambda x. \lambda y. \lambda z. x + y + z) 3 4 5$

$\lambda y. \lambda z. (3 + y + z) 4 5$

$\lambda z. (3 + 4 + z) 5$

$(3 + 4 + 5)$

12

9.7.4.2 Θέμα Ιούνιος 2023

Παρουσιάστε όλα τα βήματα σταδιακής αποτίμησης της παρακάτω έκφρασης σε λ-λογισμό:

$$(\lambda x. \lambda y. \lambda z. x^2 + y^2 + z^2) 1 2 3$$

Απάντηση

$(\lambda x. \lambda y. \lambda z. x^2 + y^2 + z^2) 1 2 3$

$\lambda y. \lambda z. (1 + y^2 + z^2) 2 3$

$\lambda z. (1 + 4 + z) 3$

$(3 + 4 + 9)$

16

Πιθανό Θέμα

$(\lambda x. \lambda y. x + y) 3 4$

$\lambda y. (3 + y) 4$

$(3 + 4)$

7

9.7.5 Θέματα με αντικατάσταση Αναδρομής με Επανάληψη

9.7.5.1 Θέμα Ιούνιος 2022

Είναι γνωστό ότι στις συναρτησιακές γλώσσες η αναδρομή αντικαθιστά την επανάληψη. Να κάνετε χρήση αναδρομικής συνάρτησης και πρώτη κλήση για τον παρακάτω βρόχο:

x:=0; i:=0; j:=1000;

while i<j **do**

x:=x+i*j; i:=i+1;

j:=j-1;

end while

return x

Απάντηση

Η κλήση είναι $f(0, 0, 1000)$ όπου

$f(x, i, j) == \text{if } i < j \text{ then}$

$f(x+i*j, i+1, j-1)$

else x

9.7.5.2 Θέμα Σεπτέμβριος 2022

Είναι γνωστό ότι **στις συναρτησιακές γλώσσες η αναδρομή αντικαθιστά την επανάληψη**. Κάντε ακριβώς το ίδιο (σώμα αναδρομικής συνάρτησης και πρώτη κλήση) για τον παραπάνω βρόγχο η εντολή j^i σημαίνει ύψωση της μεταβλητής j σε δύναμη με εκθέτη i.

R=0; i=0; j=1000;

while i<j

R=R+ j^i; i=i+1;

j=j-1;

end while

R=rec(0, 1000)

Απάντηση

R=rec(0, 1000)

rec(i, j)

if (i<j) **return** j^i + **rec(i+1, j-1)**

else return R

9.7.5.3 Θέμα Ιούνιος 2023

Είναι γνωστό ότι στις συναρτησιακές γλώσσες η αναδρομή αντικαθιστά την επανάληψη. Κάντε ακριβώς το ίδιο (σώμα αναδρομής συνάρτησης και πρώτη κλήση) για τον παραπάνω βρόγχο η εντολή j^2i σημαίνει ύψωση της μεταβλητής j σε δύναμη με εκθέτη $2*i$.

$K=0; i=0; j=80;$

while $i < j$

$K=K+ j^2i; i=i+3;$

$j=j-3;$

end while

return K

Απάντηση

rec(i, j)

if ($i < j$) **return** $j^2i + \text{rec}(i+3, j-3)$

else return K

Αρχική Κλήση $\text{rec}(0, 80)$

9.7.5.4 Θέμα 5 Ομάδα Α Ιούνιος 2019

Είναι γνωστό ότι στις συναρτησιακές γλώσσες η αναδρομή αντικαθιστά την επανάληψη. **Να κάνετε χρήση αναδρομικής συνάρτησης και πρώτη κλήση για τον παρακάτω βρόχο:**

$x:=0; i:=1; j:=1000;$

while $i < j$ **do**

$x:=x+i*j;$

$i:=i+1;$

$j:=j-1;$

end while

return x

Απάντηση

Η κλήση είναι $f(0, 1, 100)$ όπου

$f(x, i, j) == \text{if } i < j \text{ then}$

$f(x+i*j, i+1, j-1)$

else x

9.7.6 Θέματα με συμπλήρωση λογικών εκφράσεων σε γλώσσα scheme

9.7.6.1 Θέμα 5b Ιούνιος 2021

Με τι θα συμπληρώσετε τις αποτιμήσεις ?₁ και ?₂ των παρακάτω εκφράσεων συνθήκης σε γλώσσα scheme

(if (< 4 5) 9 10) \Rightarrow ?₁

(cond

((< 12 11) 10)

((< 10 9) 11)

(else 12)) \Rightarrow ?₂

Απάντηση

(if (< 4 5) 9 10) \Rightarrow **10** //ελέγχει τη συνθήκη $4 < 5$. Αν είναι αληθής επιστρέφει 9. Αν είναι ψευδής επιστρέφει 10. Εδώ η συνθήκη είναι αληθής και επιστρέφει 9

(cond //Η δήλωση αυτή αφορά πολλές συνθήκες

((< 12 11) 10) //ελέγχει τη συνθήκη $12 < 11$. Αν είναι αληθής επιστρέφει 10. Αν είναι ψευδής προχωράμε στην επόμενη συνθήκη. Εδώ η συνθήκη $12 < 11$ είναι ψευδής άρα προχωράμε στην επόμενη συνθήκη

((< 10 9) 11) ελέγχει τη συνθήκη $10 < 9$. Αν είναι αληθής επιστρέφει 11. Αν είναι ψευδής προχωράμε στην επόμενη συνθήκη. Εδώ η συνθήκη $10 < 9$ είναι ψευδής άρα προχωράμε στην επόμενη συνθήκη

(else 12)) \Rightarrow **12** //εδώ φτάνει αν όλες οι προηγούμενες συνθήκες είναι ψευδείς και επιστρέφει 10. Επειδή όλες οι προηγούμενες συνθήκες είναι ψευδείς επιστρέφεται τελικά 12

9.7.6.2 Θέμα Ιούνιος 5a 2022

Με τι θα συμπληρώσετε τις αποτιμήσεις ?₁ και ?₂ των παρακάτω εκφράσεων συνθήκης σε γλώσσα scheme

(if (< 5 4) 9 10) \Rightarrow ?₁

(cond

((< 10 9) 8)

((< 8 7) 9)

(else 10)) \Rightarrow ?₂

Απάντηση

(if (< 5 4) 9 10) \Rightarrow **10** //ελέγχει τη συνθήκη $5 < 4$. Αν είναι αληθής επιστρέφει 9. Αν είναι ψευδής επιστρέφει 10. Εδώ η συνθήκη είναι ψευδής και επιστρέφει 10

(cond //Η δήλωση αυτή αφορά πολλές συνθήκες

((< 10 9) 8) //ελέγχει τη συνθήκη $10 < 9$. Αν είναι αληθής επιστρέφει 8. Αν είναι ψευδής προχωρά στην επόμενη συνθήκη. Εδώ είναι ψευδής άρα προχωρούμε στην επόμενη συνθήκη

((< 8 7) 9) ελέγχει τη συνθήκη $8 < 7$. Αν είναι αληθής επιστρέφει 9. Αν είναι ψευδής προχωρά στην επόμενη συνθήκη. Εδώ είναι ψευδής άρα προχωρούμε στην επόμενη συνθήκη

(else 10)) \Rightarrow **10** //εδώ φτάνει αν όλες οι προηγούμενες συνθήκες είναι ψευδείς και επιστρέφει 10. Επειδή όλες οι προηγούμενες συνθήκες είναι ψευδείς επιστρέφεται τελικά 10