Παραρτήματα

A. Στοιχεία Προτασιακού Λογισμού 1081
B. Μαθηματικές Συναρτήσεις 1093
C. Λέξεις-Κλειδιά της C++ 1095
D. Πρότυπα Σύνολα Χαρακτήρων 1097
E. Η Προτεραιότητα των Πράξεων 1101

παράρτημα

A

Στοιχεία Προτασιακού Λογισμού

Ο στόχος μας σε αυτό το παράρτημα:

Τα παρακάτω έχουν στόχο να σου υπενθυμίσουν μερικές βασικές λογικές πράξεις που χρησιμοποιούμε.

Περιεχόμενα:

A.1 Συντακτικά **1081**
A.2 Νοηματικά **1082**
A.2.1 Λογική Σύζευξη "&&" **1083**
A.2.2 Λογική Διάζευξη "||" **1083**
A.2.3 Η Λογική Άρνηση "!" **1084**
A.2.4 Συνεπαγωγή "⇒" **1084**
A.2.5 Διπλή Συνεπαγωγή "⇔" **1085**
A.2.6 Άλλοι Σύνδεσμοι **1086**
A.2.6.1 Ο Σύνδεσμος του Sheffer "|" **1086**
A.2.6.2 Η Άρνηση της "||": "↓" **1086**
A.2.6.3 Αποκλειστική Διάζευξη "⊻" **1086**
A.2.7 Τιμή Παράστασης **1086**
A.3 Ταυτολογίες και Ισοδυναμία **1087**
A.4 Αποδείξεις **1087**
A.5 Κατηγορηματικός Λογισμός **1089**
A.6 Ισότητα **1090**
A.7 Οι Δικές μας Αποδείξεις **1091**
Ασκήσεις **1091**

Εισαγωγικές Παρατηρήσεις:

Θα πρέπει να τονίσουμε ότι

- τα περιεχόμενα του παραρτήματος δεν αποτελούν κάλυψη του Προτασιακού Λογισμού,
- ο Προτασιακός Λογισμός δεν είναι αρκετός για πλήρεις τυπικές αποδείξεις ορθότητας προγραμμάτων.

Στην §A.5 υπάρχει και μια παρουσίαση των ποσοδεικτών και της ισότητας: «γεύση» από τον Κατηγορηματικό Λογισμό.

Αν θέλεις να διαβάσεις περισσότερα για τη Μαθηματική Λογική δες τη βιβλιογραφία στο τέλος του βιβλίου.

A.1 Συντακτικά

Το αλφάβητο που θα χρησιμοποιήσουμε είναι:

$$\{ P, Q, R, P_0, Q_0, R_0, P_1, Q_1, R_1, \ldots, \mathsf{F}, \&\&, ||, \Rightarrow, \Leftrightarrow, !, (,) \}$$

Τα P, Q, R με δείκτη ή χωρίς είναι τα **προτασιακά σύμβολα** (propositional symbols). Με αυτά θα παριστάνουμε τις ατομικές προτάσεις.

Η "`F`" είναι μια προτασιακή **σταθερά** (constant)· αλλού θα τη δεις και ως "⊥".

Τα "`&&`", "`||`", "⇒", "⇔", "`!`" είναι τα σύμβολα των **συνδέσμων** (connectives)· οι τέσσερις πρώτοι ονομάζονται **διμελείς** (binary, 2-place) σύνδεσμοι ενώ ο "`!`" είναι **μονομελής** (unary, 1-place) σύνδεσμος. Μπορούμε να θεωρήσουμε τη σταθερά "`F`" και ως **σύνδεσμο χωρίς μέλη** (0-place connective).

Οι προτάσεις της γλώσσας ονομάζονται προτάσεις και σχηματίζονται σύμφωνα με τον εξής κανόνα:

πρόταση = *προτασιακό σύμβολο* | "`F`" | "`!`", *πρόταση* |
"`(`", *πρόταση*, *διμελής σύνδεσμος*, *πρόταση*, "`)`" ;
προτασιακό σύμβολο = "*P*" | "*Q*" | "*R*" |
"P_0" | "Q_0" | "R_0" | "P_1" | "Q_1" | "R_1" | ... ;
διμελής σύνδεσμος = "`&&`" | "`||`" | "⇒" | "⇔" ;

Παράδειγμα

Οι

$$P,\quad Q_2,\quad R_4$$

είναι προτάσεις αφού είναι προτασιακά σύμβολα.

Αφού οι P, Q_2, R_4 είναι προτάσεις και οι

$$!P,\quad !Q_2,\quad !R_4$$

είναι προτάσεις αφού έχουν τη μορφή "`!`", *πρόταση*.

Αφού οι P, Q_2, R_4, $!P$, $!Q_2$, $!R_4$ είναι προτάσεις και οι

$$(!P \;\&\&\; R_4),\quad (Q_2 \;||\; !Q_2),\quad (P \Rightarrow !R_4),\quad (Q_2 \Leftrightarrow R_4)$$

είναι προτάσεις αφού έχουν τη μορφή "`(`", *πρόταση*, *διμελής σύνδεσμος*, *πρόταση*, "`)`".

Αφού οι $(!P \;\&\&\; R_4)$, $(Q_2 \;||\; !Q_2)$, $(P \Rightarrow !R_4)$, $(Q_2 \Leftrightarrow R_4)$ είναι προτάσεις και οι

$$!(!P \;\&\&\; R_4),\quad !(Q_2 \;||\; !Q_2),\quad !(P \Rightarrow !R_4),\quad !(Q_2 \Leftrightarrow R_4)$$

είναι προτάσεις αφού έχουν τη μορφή "`!`", *πρόταση*.

Αφού οι $(!P \;\&\&\; R_4)$, $!R_4$, P, $(Q_2 \Leftrightarrow R_4)$, $!Q_2$, $(P \Rightarrow !R_4)$ είναι προτάσεις και οι

$$((!P \;\&\&\; R_4) \;||\; !R_4),$$
$$(P \;\&\&\; (Q_2 \Leftrightarrow R_4)),$$
$$((P \;\&\&\; (Q_2 \Leftrightarrow R_4)) \;||\; !Q_2),$$
$$((P \Rightarrow !R_4) \;||\; (Q_2 \Leftrightarrow R_4))$$

είναι προτάσεις αφού έχουν τη μορφή "`(`", *πρόταση*, *διμελής σύνδεσμος*, *πρόταση*, "`)`".

Όπως βλέπεις και στο παράδειγμα, οι παρενθέσεις μπορεί να γίνουν ενοχλητικά πολλές. Δίνουμε λοιπόν τον παρακάτω συμπληρωματικό κανόνα:

- ***Από μια πρόταση μπορούμε να αφαιρούμε το εξώτερο ζεύγος παρενθέσεων***

Με βάση αυτόν τον κανόνα, μπορούμε να γράψουμε:

$$!P \;\&\&\; R_4 \quad \text{αντί για} \quad (!P \;\&\&\; R_4),$$
$$Q_2 \;||\; !Q_2 \quad \text{αντί για} \quad (Q_2 \;||\; !Q_2),$$
$$P \Rightarrow !R_4 \quad \text{αντί για} \quad (P \Rightarrow !R_4),$$
$$Q_2 \Leftrightarrow R_4 \quad \text{αντί για} \quad (Q_2 \Leftrightarrow R_4),$$
$$(!P \;\&\&\; R_4) \;||\; !R_4 \quad \text{αντί για} \quad ((!P \;\&\&\; R_4) \;||\; !R_4),$$
$$P \;\&\&\; (Q_2 \Leftrightarrow R_4) \quad \text{αντί για} \quad (P \;\&\&\; (Q_2 \Leftrightarrow R_4)),$$
$$(P \;\&\&\; (Q_2 \Leftrightarrow R_4)) \;||\; !Q_2 \quad \text{αντί για} \quad ((P \;\&\&\; (Q_2 \Leftrightarrow R_4)) \;||\; !Q_2),$$
$$(P \Rightarrow !R_4) \;||\; (Q_2 \Leftrightarrow R_4) \quad \text{αντί για} \quad ((P \Rightarrow !R_4) \;||\; (Q_2 \Leftrightarrow R_4))$$

Α.2 Νοηματικά

Κάθε προτασιακό σύμβολο (P, P_0, P_1,... Q, Q_0, Q_1,... R, R_0, R_1,...) μπορεί να πάρει μια από τις λογικές τιμές `false` (ψέμα, δεν ισχύει) ή `true` (αλήθεια, ισχύει). Η σταθερά `F` παίρνει πάντοτε τιμή `false`.

Από τη στιγμή που καθορίζονται οι τιμές των προτασιακών συμβόλων μπορούμε να υπολογίσουμε τη (λογική) τιμή οποιασδήποτε πρότασης που περιέχει αυτά τα σύμβολα. Για να μπορέσουμε να κάνουμε κάτι τέτοιο, θα πρέπει να ξέρουμε τα αποτελέσματα των λογικών πράξεων που συμβολίζονται με τα "`&&`", "`||`", "`!`". Αυτά συνήθως δίνονται σε πίνακες, που λέγονται **πίνακες αλήθειας** ή **αληθοπίνακες** (truth tables).

A.2.1 Λογική Σύζευξη "&&"

Στον Πίν. Α-1 βλέπεις τον αληθοπίνακα της λογικής **σύζευξης** (conjunction) "`&&`". Η πρώτη γραμμή λέει: Αν η πρόταση $\mathcal{A}$ έχει τιμή `false` και η πρόταση $\mathcal{B}$ έχει τιμή `false` τότε και η $\mathcal{A}$ `&&` $\mathcal{B}$ και $\mathcal{B}$ `&&` $\mathcal{A}$ έχουν τιμή `false`. Με τον ίδιο τρόπο διαβάζεις και τις υπόλοιπες γραμμές.

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$ `&&` $\mathcal{B}$	$\mathcal{B}$ `&&` $\mathcal{A}$
`false`	`false`	`false`	`false`
`false`	`true`	`false`	`false`
`true`	`false`	`false`	`false`
`true`	`true`	`true`	`true`

Πίν. Α-1: Ο αληθοπίνακας της "`&&`".

Πρόσεξε ότι η πράξη "`&&`" είναι αντιμεταθετική: Οι $\mathcal{A}$ `&&` $\mathcal{B}$ και $\mathcal{B}$ `&&` $\mathcal{A}$ έχουν πάντοτε την ίδια τιμή.

Πώς μπορούμε να διατυπώσουμε στην καθομιλουμένη το αποτέλεσμα της πράξης "`&&`";

- ***Αν η πρόταση $\mathcal{A}$ έχει τιμή `true` (είναι αληθής) και ταυτοχρόνως η πρόταση $\mathcal{B}$ έχει τιμή `true` τότε (και μόνον τότε) είναι αληθείς και οι προτάσεις $\mathcal{A}$ `&&` $\mathcal{B}$ και $\mathcal{B}$ `&&` $\mathcal{A}$.***

Παράδειγμα

Έστω ότι P είναι η πρόταση «η μεταβλητή x έχει τιμή μεγαλύτερη από 5» ($x > 5$) και Q η πρόταση «η μεταβλητή x έχει τιμή μικρότερη από 10» ($x < 10$). Η πρόταση «η μεταβλητή x έχει τιμή μεταξύ 5 και 10» ($5 < x < 10$) σημαίνει στην πραγματικότητα

«η μεταβλητή x έχει τιμή μεγαλύτερη από 5»

και

«η μεταβλητή x έχει τιμή μικρότερη από 10»

δηλαδή P `&&` Q. Αν η x έχει τιμή 6, τότε η P και η Q είναι αληθείς, έχουν τιμή `true`. Από την τέταρτη γραμμή του αληθοπίνακα της "`&&`" βλέπουμε ότι και η P `&&` Q έχει τιμή `true`, δηλαδή είναι αληθής.

Αν το x έχει τιμή 17, τότε η P είναι αληθής (τιμή `true`) και η Q είναι ψευδής (τιμή `false`). Από την τρίτη γραμμή του αληθοπίνακα της `&&` βλέπουμε ότι και η P `&&` Q έχει τιμή `false`, δηλαδή είναι ψευδής.

Συχνά θα δεις αυτόν τον σύνδεσμο να γράφεται και ως: $\wedge$, **και**, `and` (στις γλώσσες προγραμματισμού Pascal, Basic και άλλες).

Παρόμοιος είναι και ένας άλλος σύνδεσμος, ο "`cand`"· αυτός έχει «οικονομικό» τρόπο υπολογισμού, δηλαδή: αν κατά τον υπολογισμό της $\mathcal{A}$ `cand` $\mathcal{B}$ η τιμή της $\mathcal{A}$ βρεθεί `false` (ψευδής), η τιμή ολόκληρης της σύνθετης πρότασης υπολογίζεται ως `false` χωρίς να υπολογισθεί η τιμή της $\mathcal{B}$. Να θυμίσουμε ότι έτσι γίνεται ο υπολογισμός της πράξης "`&&`" στη C++ (§5.6).

A.2.2 Λογική Διάζευξη "||"

Στον Πίν. Α-2 βλέπεις τον αληθοπίνακα της λογικής **διάζευξης** (disjunction) "`||`". Στην καθομιλουμένη το αποτέλεσμα της πράξης "`||`" δίνεται με τον εξής κανόνα:

- ***Αν μια τουλάχιστον από τις προτάσεις $\mathcal{A}$, $\mathcal{B}$ έχει τιμή `true` (είναι αληθής) τότε και οι προτάσεις $\mathcal{A}$ `||` $\mathcal{B}$ και $\mathcal{B}$ `||` $\mathcal{A}$ έχουν τιμή `true` (είναι αληθείς).***

Όπως βλέπεις, η μόνη περίπτωση που η $\mathcal{A}$ `||` $\mathcal{B}$ παίρνει τιμή `false` είναι όταν και η $\mathcal{A}$ και η $\mathcal{B}$ έχουν τιμή `false`.

Α	*Β*	*Α* `||` *Β*	*Β* `||` *Α*
`false`	`false`	`false`	`false`
`false`	`true`	`true`	`true`
`true`	`false`	`true`	`true`
`true`	`true`	`true`	`true`

Πίν. Α-2: Ο αληθοπίνακας της "`||`".

Και η "`||`" είναι αντιμεταθετική: Οι *Α* `||` *Β* και *Β* `||` *Α* έχουν πάντοτε την ίδια τιμή.

Παράδειγμα

Έστω ότι *x* = 15 και θέλουμε να αποφανθούμε για την αλήθεια της πρότασης «το *x* είναι πολλαπλάσιο του 2 ή του 3». Ένας άλλος τρόπος να διατυπώσουμε την πρόταση είναι: «το *x* είναι πολλαπλάσιο του 2» ή «το *x* είναι πολλαπλάσιο του 3». Ας ονομάσουμε *P* την «το *x* είναι πολλαπλάσιο του 2» και *Q* την «το *x* είναι πολλαπλάσιο του 3». Η *P* έχει τιμή `false`, ενώ η *Q* έχει τιμή `true`. Σύμφωνα με την τρίτη γραμμή του πίνακά μας η *P* `||` *Q* έχει τιμή `true`.

Άλλες γραφές του συνδέσμου: "∨", "**ή**", "`or`" (στις γλώσσες προγραμματισμού Pascal, Basic και άλλες).

Παρόμοιος είναι και ένας άλλος σύνδεσμος (αντίστοιχος του "`cand`"), ο "`cor`": αν κατά τον υπολογισμό της *Α* `cor` *Β* η τιμή της *Α* βρεθεί `true` (αληθής), η τιμή ολόκληρης της σύνθετης πρότασης υπολογίζεται ως `true` χωρίς να υπολογισθεί η τιμή της *Β*. Αν γυρίσεις στην §5.6 θα θυμηθείς ότι έτσι δουλεύει ο "`||`" της C++.

Α.2.3 Η Λογική Άρνηση "!"

Α	`!`*Α*
`false`	`true`
`true`	`false`

Πίν. Α-3: Ο αληθοπίνακας της "`!`".

Στον Πίν. Α-3 βλέπεις τον αληθοπίνακα της λογικής **άρνησης** (negation) "`!`", που μας λέει απλά:

- ***Αν η Α έχει τιμή*** `true`***, είναι αληθής, τότε η*** `!`***Α έχει τιμή*** `false`***, είναι ψευδής, και αν η Α έχει τιμή*** `false`***, είναι ψευδής, τότε η*** `!`***Α έχει τιμή*** `true`***, είναι αληθής.***

Θα τον δεις γραμμένο ακόμη ως "¬" ή ως "~" ή ως "**όχι**" ή ως "`not`" (Pascal, Basic).

Α.2.4 Συνεπαγωγή "⇒"

Α	*Β*	*Α* ⇒ *Β*	*Β* ⇒ *Α*
`false`	`false`	`true`	`true`
`false`	`true`	`true`	`false`
`true`	`false`	`false`	`true`
`true`	`true`	`true`	`true`

Πίν. Α-4: Ο αληθοπίνακας της "⇒". Οι στήλες *Α* ⇒ *Β* και *Β* ⇒ *Α* είναι διαφορετικές.

Στον Πίν. Α-3 βλέπεις τον αληθοπίνακα της "⇒", που τη λέμε **συνεπαγωγή** (implication).

Στην *Α* ⇒ *Β* η *Α* λέγεται **ηγουμένη** (antecedent) και η *Β* **επομένη** (consequent) ή **απόδοση** (apodosis). Αλλού θα δεις την *Α* ⇒ *Β* να γράφεται ως "***if Α then Β***", δηλαδή ***αν*** (είναι αληθής η) *Α* ***τότε*** (είναι αληθής η) *Β*.[1]

Θα γράψουμε τον αληθοπίνακα της συνεπαγωγής προσπαθώντας να καταλάβουμε ένα παράδειγμα με βάση την καθημερινή κοινή λογική. Πάρε ως *P* την πρόταση «ο Νίκος λέει πάντα ψέματα» και ως *Q* την «αυτό που είπε τώρα ο Νίκος είναι ψέμα». *P* ⇒ *Q* είναι η πρόταση: «***αν*** ο Νίκος λέει πάντα ψέματα ***τότε*** αυτό που είπε τώρα ο Νίκος είναι ψέμα». Θα δοκιμάσουμε όλες τις δυνατές τιμές αλήθειας για τις *P*, *Q* και στην κάθε περίπτωση θα βλέπουμε αν αυτό που προκύπτει συμφωνεί με την κοινή λογική.

Ξεκινούμε από τα εύκολα: η *P* έχει τιμή `true`, το ίδιο και η *Q* (γραμμή 4 του Πίν. Α-3)· έχουμε δηλαδή την πρόταση «αν ο Νίκος λέει πάντα ψέματα τότε αυτό που είπε τώρα ο Νίκος είναι ψέμα». Αυτή είναι αληθής, σύμφωνα με την κοινή λογική. Βάζουμε λοιπόν τιμή `true` στην *P* ⇒ *Q*.

[1] Όπως καταλαβαίνεις, αυτό το ***if ... then*** δεν έχει σχέση με την αντίστοιχη προγραμματιστική δομή.

Πάμε τώρα στην περίπτωση που η P έχει τιμή `true`, αλλά η Q έχει τιμή `false` (γραμμή 3)· τώρα η $P \Rightarrow Q$ σημαίνει: «αν ο Νίκος λέει πάντα ψέματα τότε αυτό που είπε τώρα ο Νίκος *δεν* είναι ψέμα». Αυτή η πρόταση όμως είναι ολοφάνερα ψευδής με βάση την κοινή λογική. Βάζουμε λοιπόν τιμή `false` στην $P \Rightarrow Q$.

Στη γραμμή 2 η ηγουμένη P έχει τιμή `false` και η Q τιμή `true`, λέμε δηλαδή «αν ο Νίκος *δεν* λέει πάντα ψέματα τότε αυτό που είπε τώρα ο Νίκος είναι ψέμα». Αυτή μπορεί να φαίνεται λίγο "ασυνάρτητη" ή "παράλογη", αλλά το ερώτημα είναι: είναι αληθής; Πώς να απαντήσουμε μια τέτοια ερώτηση για μια "ασυναρτησία"; Ας κάνουμε την άλλη ερώτηση: είναι ψευδής; Όχι! Αφού λοιπόν η πρόταση δεν είναι ψευδής (και η λογική μας είναι δίτιμη) δεν μπορεί παρά να είναι αληθής, δηλαδή η $P \Rightarrow Q$ έχει τιμή `true`.

Τέλος, στη γραμμή 1, ηγουμένη και επομένη έχουν τιμή `false`. Έχουμε δηλαδή την "ασυναρτησία" «αν ο Νίκος *δεν* λέει πάντα ψέματα τότε αυτό που είπε τώρα ο Νίκος *δεν* είναι ψέμα». Και εδώ, με το σκεπτικό της προηγούμενης περίπτωσης, βάζουμε στην $P \Rightarrow Q$ τιμή `true`.

Λίγο παράξενα τα δύο τελευταία... Υπάρχουν όμως και χειρότερα. Π.χ. οι παρακάτω προτάσεις είναι αληθείς:

«αν ο ουρανός είναι γαλανός τότε οι λευκές αρκούδες ζουν πολύ βόρεια»

«αν ο Ήλιος γυρίζει γύρω από τη Γη τότε εγώ είμαι αστροναύτης»

Η πρώτη πρόταση είναι αληθής διότι οι δύο ατομικές προτάσεις που την αποτελούν είναι αληθείς, η δεύτερη είναι επίσης αληθής διότι η πρώτη ατομική πρόταση είναι ψευδής. Αλλά, και στις δύο περιπτώσεις, ηγούμενη και επόμενη είναι τελείως άσχετες. Εδώ το πρόβλημα είναι σοβαρό[2]...

Όπως είναι φυσικό, αυτή η λογική πράξη *δεν είναι αντιμεταθετική*. Άλλο το $\mathcal{A} \Rightarrow \mathcal{B}$ και άλλο το $\mathcal{B} \Rightarrow \mathcal{A}$.

Αλλού θα δεις την $\mathcal{A} \Rightarrow \mathcal{B}$ να γράφεται ως $\mathcal{A} \rightarrow \mathcal{B}$ ή $\mathcal{A} \supset \mathcal{B}$ ή, όπως είπαμε, ***if*** $\mathcal{A}$ ***then*** $\mathcal{B}$.

Παρατήρηση:▶

Αν δεις τον Πίν. Α-4 και πάρεις υπόψη σου τη διάταξη "`false < true`" που έχουμε στη C++ καταλαβαίνεις γιατί η "$P \Rightarrow Q$" υλοποιείται στη C++ ως "P `<=` Q".◀

Α.2.5 Διπλή Συνεπαγωγή "⇔"

Στον Πίν. Α-4 βλέπεις τον αληθοπίνακα της "⇔", που τη λέμε **διπλή συνεπαγωγή** (double implication). Τι μας λέει;

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \Leftrightarrow \mathcal{B}$	$\mathcal{B} \Leftrightarrow \mathcal{A}$
`false`	`false`	`true`	`true`
`false`	`true`	`false`	`false`
`true`	`false`	`false`	`false`
`true`	`true`	`true`	`true`

Πίν. Α-5: Ο αληθοπίνακας της "⇔".

- ***Αν οι $\mathcal{A}$ και $\mathcal{B}$ έχουν την ίδια τιμή (και οι δύο*** `true` ***ή και οι δύο*** `false`***) τότε η $\mathcal{A} \Leftrightarrow \mathcal{B}$ είναι αληθής (τιμή*** `true`***), αλλιώς η $\mathcal{A} \Leftrightarrow \mathcal{B}$ είναι ψευδής (τιμή*** `false`***).***

Όπως βλέπεις, οι $\mathcal{A} \Leftrightarrow \mathcal{B}$ και $\mathcal{B} \Leftrightarrow \mathcal{A}$ έχουν πάντοτε την ίδια τιμή· η πράξη ⇔ είναι αντιμεταθετική.

Παράδειγμα

Έστω ότι $x = 17$. Αν P είναι η πρόταση "$x > 10$" και Q η "$x < 35$", τότε η $P \Leftrightarrow Q$ είναι αληθής διότι και η P και η Q έχουν τιμή `true`. Αν P_1 είναι η πρόταση "$x < 10$" και Q_1 η "$x > 35$" τότε η $P_1 \Leftrightarrow Q_1$ είναι αληθής διότι και η P_1 και η Q_1 έχουν τιμή `false`. Η $P \Leftrightarrow Q_1$ είναι ψευδής διότι η P έχει τιμή `true`, ενώ η έχει Q_1 `false`.

---

[2] Αυτή η συνεπαγωγή ονομάζεται και **ουσιώδης συνεπαγωγή** (material implication) σε αντιδιαστολή με μια άλλη πιο **ακριβή συνεπαγωγή** (strict implication) που εξετάζουν οι **τροπικές λογικές** (modal logics) βάζοντας τη συνεπαγωγή σε πιο "σωστή" βάση.

Αλλού θα δεις την $\mathcal{A} \Leftrightarrow \mathcal{B}$ να γράφεται ως $\mathcal{A} \leftrightarrow \mathcal{B}$ ή ***A if and only if B***. Θα τη δεις ακόμη ως $\mathcal{A} \equiv \mathcal{B}$ αλλά εδώ προσοχή: όπως θα δεις στη συνέχεια, εμείς θα χρησιμοποιούμε το "≡" με άλλο νόημα.

Παρατήρηση:▶

Στη C++ η "$P \Leftrightarrow Q$" υλοποιείται ως "`P == Q`".◀

Α.2.6 Άλλοι Σύνδεσμοι

Όπως καταλαβαίνεις, μπορούμε να έχουμε 2^4 διμελείς συνδέσμους και 2^2 μονομελείς. Θα πρέπει να τους περιλάβουμε όλους στο σύστημά μας; Όχι! Αυτοί που έχουμε εδώ φτάνουν και περισσεύουν, όπως μπορεί να αποδειχθεί. Πάντως υπάρχουν τρεις που είναι χρήσιμοι και καλό είναι να τους δούμε.

Θα δούμε στη συνέχεια ότι οι δύο πρώτοι από αυτούς, οι "`nand`" και "`nor`" έχουν μια ενδιαφέρουσα ιδιότητα: όλοι οι άλλοι σύνδεσμοι μπορεί να δοθούν με βάση έναν από αυτούς.

Α.2.6.1 Ο Σύνδεσμος του Sheffer "|"

Στον Πίν. Α-6 βλέπεις τον αληθοπίνακα του "`|`" που λέγεται **σύνδεσμος του Sheffer** (Sheffer's stroke). Αν τον συγκρίνεις με τον Πίν. Α-1 βλέπεις ότι το $\mathcal{A}$ `|` $\mathcal{B}$ είναι μια συντομογραφία του `!(`$\mathcal{A}$ `&&` $\mathcal{B}$`)`.

Θα τον δεις γραμμένο και ως "↑"ή ως "`nand`" (αφού είναι η "άρνηση της and": **n**ot **and**).

Α.2.6.2 Η Άρνηση της "||": "↓"

Όπως υπάρχει "άρνηση της and" υπάρχει και "άρνηση της or". Συμβολίζεται με το "↓" και στον Πίν. Α-7 βλέπεις τον αληθοπίνακά του. Αν τον συγκρίνεις με τον Πίν. Α-2 βλέπεις ότι, πράγματι, το $\mathcal{A} \downarrow \mathcal{B}$ είναι μια συντομογραφία του `!(`$\mathcal{A}$ `&&` $\mathcal{B}$`)`.

Θα τον δεις γραμμένο και ως "`nor`" (**n**ot **or**).

Α.2.6.3 Αποκλειστική Διάζευξη "⊻"

Η **αποκλειστική διάζευξη** (exclusive disjunction) –αληθοπίνακας στον Πίν. Α-8– διαφέρει από την απλή διάζευξη (Πίν. Α-2) στην τελευταία γραμμή του αληθοπίνακα. Η $\mathcal{A} \veebar \mathcal{B}$ έχει τιμή `true` όταν *μία και μόνο μία* από τις $\mathcal{A}$ και $\mathcal{B}$ έχει τιμή `true`. Συγκρίνοντας με τον Πίν. Α-4 βλέπεις ότι είναι το ίδιο πράγμα με το `!(`$\mathcal{A} \Leftrightarrow \mathcal{B}$`)`.

Θα τον δεις γραμμένο και ως "⊕" ή ως "`xor`" (Turbo Pascal).

Α.2.7 Τιμή Παράστασης

Στους αληθοπίνακες ορισμού των λογικών πράξεων χρησιμοποιήσαμε προτασιακά σύμβολα (P, Q). Οι πράξεις γίνονται με τον ίδιο τρόπο μεταξύ οποιωνδήποτε προτάσεων. Ας δούμε ένα

Παράδειγμα 🡲

Με βάση τους πίνακες, θα υπολογίσουμε την τιμή της παράστασης:

$$((P_0 \;\&\&\; P_1) \;||\; (P_2 \;\&\&\; P_1)) \;\&\&\; P_3$$

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$ | $\mathcal{B}$
false	false	true
false	true	true
true	false	true
true	true	false

Πίν. Α-6: Ο αληθοπίνακας της "`|`".

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \downarrow \mathcal{B}$
false	false	True
false	true	False
true	false	False
true	true	False

Πίν. Α-7: Ο αληθοπίνακας της "↓"

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \veebar \mathcal{B}$
False	false	false
False	true	true
True	false	true
True	true	false

Πίν. Α-8: Ο αληθοπίνακας της "⊻".

αν έχουμε δώσει τιμές:

`true` στην P_0, `false` στην P_1, `true` στην P_2, `false` στην P_3

Η P_0 `&&` P_1 παίρνει τιμή `false` (`&&`, γρ. 3),
η P_2 `&&` P_1 παίρνει τιμή `false` (`&&`, γρ. 3),
η (P_0 `&&` P_1) `||` (P_2 `&&` P_1) παίρνει τιμή `false` (`||`, γρ. 1) και
η ((P_0 `&&` P_1) `||` (P_2 `&&` P_1)) `&&` P_3 παίρνει τιμή `false` (`&&`, γρ. 1)[3].
🖘🖘🖘

Α.3 Ταυτολογίες και Ισοδυναμία

Ένα πολύ ενδιαφέρον υποσύνολο προτάσεων είναι οι ταυτολογίες. **Ταυτολογία** (tautology) είναι μια πρόταση που παίρνει τιμή `true` όποιες κι αν είναι οι τιμές των προτασιακών συμβόλων.

Χαρακτηριστικά παραδείγματα ταυτολογιών:

`!F`
A `||` (`!`A)
(A `&&` (B `||` C)) ⇔ ((A `&&` B) `||` (A `&&` C))
(A `||` (B `&&` C)) ⇔ ((A `||` B) `&&` (A `||` C))
(`!`(A `||` B)) ⇔ ((`!`A) `&&` (`!`B))
(`!`(A `&&` B)) ⇔ ((`!`A) `||` (`!`B))

Αν η $A \Leftrightarrow B$ είναι ταυτολογία τότε οι A και B λέγονται **ισοδύναμες** (equivalent). Στην περίπτωση αυτήν γράφουμε $A \equiv B$. Από τα τελευταία παραδείγματα που δώσαμε πιο πάνω έχουμε:

(A `&&` (B `||` C)) ≡ ((A `&&` B) `||` (A `&&` C))
(A `||` (B `&&` C)) ≡ ((A `||` B) `&&` (A `||` C))
(`!`(A `||` B)) ≡ ((`!`A) `&&` (`!`B))
(`!`(A `&&` B)) ≡ ((`!`A) `||` (`!`B))

Αν δεν σε βολεύει να δουλεύεις με την ⇒, μπορείς να χρησιμοποιείς την ισοδυναμία (Άσκ. 1α):

($A \Rightarrow B$) ≡ ((`!`A) `||` B)

Συχνά χρησιμοποιούμε τις ταυτολογίες (Άσκ. 1β):

$A \Rightarrow$ (A `||` B) και $B \Rightarrow$ (A `||` B)

όπως και τις:

(A `&&` B) ⇒ A και (A `&&` B) ⇒ B

Είναι φανερό ότι ισχύουν οι παρακάτω ισοδυναμίες (Άσκ. 2):

($A \veebar B$) ≡ ((A `||` B) `&&` (`!`(A `&&` B)))
($A \veebar B$) ≡ ((A `&&` (`!`B)) `||` ((`!`A) `&&` B))

Α.4 Αποδείξεις

Ας προσπαθήσουμε να αποδείξουμε ότι η A `||` (`!`A) είναι ταυτολογία. Ένας τρόπος είναι να εξετάσουμε όλες τις δυνατές τιμές της A. Αυτό το βλέπεις στον Πίν. Α-9. Όπως βλέπεις, η τελευταία στήλη του πίνακα έχει σ' όλες τις γραμμές την τιμή `true`.

A	`!`A	A `||` (`!`A)
`false`	`true`	`true`
`true`	`false`	`true`

Πίν. Α-9: Αληθοπίνακας για την απόδειξη της ταυτολογίας A`||` (`!`A).

Να δούμε άλλο ένα

[3] Βέβαια, αν είμαστε λίγο προσεκτικοί, αποφαινόμαστε αμέσως ότι πρότασή μας παίρνει τιμή `false`, αφού είναι της μορφής A `&&` P_3 και η P_3 έχει τιμή `false` (`&&`, γρ. 1, 3).

Παράδειγμα

Θα αποδείξουμε ότι η $(P \;\&\&\; Q) \Rightarrow P$ είναι ταυτολογία. Θα χρειαστούμε έναν αληθοπίνακα με στήλες για τις P, Q, $P \;\&\&\; Q$, $(P \;\&\&\; Q) \Rightarrow P$. Θα πρέπει να εξετάσουμε όλους τους συνδυασμούς τιμών των P, Q· άρα ο πίνακας θα έχει 4 (= 2×2) γραμμές. Τον βλέπεις στον Πίν. Α-10.

P	Q	$P \;\&\&\; Q$	$(P \;\&\&\; Q) \Rightarrow P$
`false`	`false`	`false`	`true`
`false`	`true`	`false`	`true`
`true`	`false`	`false`	`true`
`true`	`true`	`true`	`true`

Πίν. Α-10: Πίνακας αλήθειας για την απόδειξη της $(P \;\&\&\; Q) \Rightarrow P$.

Φυσικά, δεν αποδεικνύουμε μόνον ταυτολογίες. Μπορεί να χρειαστεί να αποδείξουμε και κάτι σαν:

Αν ισχύει η A τότε ισχύει και η B.

Παράδειγμα

Απόδειξε ότι: Αν ισχύει η $A \;\&\&\; B$ τότε ισχύει η $A \;||\; B$.

Και στην περίπτωση αυτήν μπορούμε να κάνουμε τον αληθοπίνακα, αλλά τώρα συγκρίνουμε μόνον τις γραμμές στις οποίες η υπόθεση $(A \;\&\&\; B)$ έχει τιμή `true`.[4] Αυτό συμβαίνει μόνο στην 4η γραμμή. Από αυτήν μπορούμε να πούμε ότι: αφού όποτε η υπόθεση παίρνει τιμή `true` και το συμπέρασμα παίρνει τιμή `true`, αποφαινόμαστε ότι: Αν ισχύει η $A \;\&\&\; B$ τότε ισχύει η $A \;||\; B$.

A	B	$A \;\&\&\; B$	$A \;||\; B$
`false`	`false`	`false`	`false`
`false`	`true`	`false`	`true`
`true`	`false`	`false`	`true`
`true`	`true`	`true`	`true`

Πίν. Α-11: Πίνακας αλήθειας για την απόδειξη του θεωρήματος: αν ισχύει η $A \;\&\&\; B$ τότε ισχύει και η $A \;||\; B$.

Όπως βλέπεις, ο πρώτος πίνακας (Α-9) έχει 2 γραμμές, ενώ οι επόμενοι (Α-10, Α-11) έχουν 4 γραμμές. Γενικά, αν προσπαθήσεις να αποδείξεις μια πρόταση με N προτασιακά σύμβολα θα χρειαστείς πίνακα με 2^N γραμμές. Καταλαβαίνεις λοιπόν ότι αυτός ο τρόπος δεν είναι ικανοποιητικός.

Ένας άλλος τρόπος, προτιμότερος, είναι να βάλεις **αξιώματα** και **συμπερασματικούς κανόνες** και να αποδεικνύεις αυτό που θέλεις χωρίς να εξετάζεις όλες τις δυνατές τιμές των προτασιακών συμβόλων.

Ας δούμε, για παράδειγμα, μερικούς συμπερασματικούς κανόνες του προτασιακού λογισμού:

$$\frac{A, B}{A \;\&\&\; B}(1) \qquad \frac{A, B}{B \;\&\&\; A}(2) \qquad \frac{A \;\&\&\; B}{A}(3) \qquad \frac{A \;\&\&\; B}{B}(4)$$

Ο πρώτος κανόνας λέει: "Αν ισχύουν οι A και B τότε μπορείς να βγάλεις το συμπέρασμα ότι ισχύει και η $A \;\&\&\; B$". Ο τρίτος κανόνας λέει: "Αν ισχύει η $A \;\&\&\; B$ τότε μπορείς να βγάλεις το συμπέρασμα ότι ισχύει η A". Όπως καταλαβαίνεις, οι τέσσερις αυτοί κανόνες εκφράζουν αυτό που γράψαμε για την πράξη "`&&`": Αν ισχύει η πρόταση A και ταυτόχρονα ισχύει η πρόταση B τότε και μόνον τότε ισχύουν και οι προτάσεις $A \;\&\&\; B$ και $B \;\&\&\; A$.

Οι συμπερασματικοί κανόνες της "`||`" είναι οι εξής:

$$\frac{A}{A \;||\; B}(5) \qquad \frac{B}{A \;||\; B}(6)$$

Αν διαβάσεις αυτούς τους δύο κανόνες όπως είπαμε παραπάνω θα δεις ότι δεν λένε τίποτε περισσότερο από αυτό που είπαμε πιο πριν: Αν ισχύει η πρόταση A ή η πρόταση B ή και οι δύο τότε ισχύει και η πρόταση $A \;||\; B$.

[4] Φυσικά, το ότι «αν ισχύει η $A \;\&\&\; B$ τότε ισχύει η $A \;||\; B$» *έχει σχέση* με το ότι η $(A \;\&\&\; B) \Rightarrow (A \;||\; B)$ είναι ταυτολογία· δες αυτά που λέμε στο τέλος της παραγράφου.

Ας δούμε τώρα, με ένα παράδειγμα, πώς γίνεται η απόδειξη με αυτούς τους κανόνες, χωρίς να καταφύγουμε σε αληθοπίνακες. Ας πούμε ότι έχουμε να αποδείξουμε ότι: Αν ισχύει η A **&&** B τότε ισχύει η A **||** B.

1. A **&&** B υπόθεση.
2. A από το βήμα (1) και τον κανόνα (3).
3. A **||** B από το βήμα (2) και τον κανόνα (5).

Όπως βλέπεις, η απόδειξη στηρίζεται στη γραφή της πρότασης και όχι στις τιμές των A και B.

Αυτό που αποδείξαμε ήταν ένα *θεώρημα*. Το «Αν ισχύει η A **&&** B τότε ισχύει η A **||** B» γράφεται συμβολικώς ως εξής:

$$(A \;\&\&\; B) \vdash (A \;||\; B)$$

Αν η απόδειξη γίνει με αληθοπίνακες γράφουμε: $(A \;\&\&\; B) \models (A \;||\; B)$.

Πάντως αποδεικνύεται ότι: $S \vdash T$ αν και μόνο αν $S \models T$.

Και κάτι άλλο, ενδιαφέρον· αποδεικνύεται ότι: $S \vdash T$ αν και μόνον αν η $S \Rightarrow T$ είναι ταυτολογία.

Α.5 Κατηγορηματικός Λογισμός

Ο Προτασιακός Λογισμός έχει πολύ περιορισμένες δυνατότητες. Π.χ. δεν μπορούμε να ασχοληθούμε με προτάσεις όπως: «για κάθε πραγματικό x έχουμε $x^2 \geq 0$».

Ο **Κατηγορηματικός Λογισμός** (Κ.Λ. predicate calculus) επιτρέπει τη διαχείριση τύπων που περιέχουν σταθερές και μεταβλητές (που παίρνουν τιμές) από κάποιο σύνολο τιμών (π.χ. το $\mathbb{Z}$, το $\mathbb{R}$ κλπ). Έτσι, το παραπάνω παράδειγμα μπορεί να διατυπωθεί συμβολικά ως εξής:

$$\forall x \bullet x^2 \geq 0$$

Το σύμβολο "$\forall$" σημαίνει "για κάθε" και ονομάζεται **καθολικός ποσοδείκτης** (universal quantifier). Το x είναι μια μεταβλητή και το $x^2 \geq 0$ είναι ένα **κατηγόρημα** (predicate) μιας θέσης (μεταβλητής).

Μια άλλη κατηγορία προτάσεων είναι οι **υπαρξιακές** (existential), π.χ.: «υπάρχει μια τουλάχιστον τιμή της οποίας το τετράγωνο είναι μηδέν». Αυτή διατυπώνεται συμβολικώς ως εξής:

$$\exists x \bullet x^2 == 0$$

Το σύμβολο "$\exists$" σημαίνει "υπάρχει" και ονομάζεται **υπαρξιακός ποσοδείκτης** (existential quantifier). Το "**==**" σημαίνει "είναι ίσο":[5]

Στους δύο τύπους που γράψαμε παραπάνω μπορούμε να αλλάξουμε χωρίς κανένα πρόβλημα όνομα της μεταβλητής, π.χ.:

$$\forall y \bullet y^2 \geq 0 \quad \text{ή} \quad \exists t \bullet t^2 == 0$$

Λέμε ότι οι μεταβλητές αυτές είναι **δεσμευμένες** (bound) από τον ποσοδείκτη. Κοίταξε όμως τον τύπο:

$$\exists x \bullet x + f(x, y) > 0$$

Εδώ η x είναι δεσμευμένη, δηλαδή μπορούμε να γράψουμε: $\exists u \bullet u + f(u,y) > 0$ αλλά δεν συμβαίνει το ίδιο με τη y: η y είναι μια **ελεύθερη** (free) μεταβλητή.

Μετά το "$\bullet$" μπορεί να ακολουθεί, όχι μόνον κατηγόρημα, αλλά τύπος του Κ.Λ., π.χ.:

$$\forall y \bullet (\exists x \bullet x + y == 0)$$

που σημαίνει: για κάθε y υπάρχει x τέτοιο ώστε $x + y == 0$. Για τον τύπο $\exists x \bullet x + y == 0$ η x είναι δεσμευμένη μεταβλητή –και θα μπορούσαμε να της αλλάξουμε όνομα, π.χ.: $\exists z \bullet z + y == 0$– ενώ η y είναι ελεύθερη. Για τον αρχικό τύπο η y είναι δεσμευμένη και μπορούμε να

[5] Γιατί "**==**" και όχι "**=**"; Διότι το "**=**" στη C++ έχει διαφορετικό νόημα. Το "**==**" είναι ο τελεστής σύγκρισης για ισότητα της C++.

της αλλάξουμε όνομα. Θα μπορούσαμε να την ονομάσουμε x; Βεβαίως, αλλά δεν θα ήταν και πολύ έξυπνο διότι τότε ο τύπος σου θα γίνει: $\forall x \bullet (\exists x \bullet x + x == 0)$ και άντε να βγάλεις άκρη! Παρ' όλο που οι συντακτικοί κανόνες του Κ.Λ. επιτρέπουν τέτοιους τύπους, αυτοί δεν είναι δεκτοί.

Πολύ συχνά έχουμε τύπους όπως: $\forall x \bullet ((x > 0) \Rightarrow (-x < 0))$. Αυτός γράφεται και ως εξής: $\forall x\ (x > 0) \bullet (-x < 0))$ ή ακόμη: $\forall x > 0 \bullet (-x < 0))$ και λέμε ότι στην περίπτωση αυτή έχουμε **φραγμένο ποσοδείκτη**.

Γενικώς, στον Κ.Λ. δεν είναι δυνατόν να κάνουμε αποδείξεις με πίνακες, αφού θα πρέπει να εξετάσουμε όλες τις δυνατές τιμές των μεταβλητών μας (εκτός από την περίπτωση που το σύνολο τιμών των μεταβλητών μας είναι πολύ μικρό). Θα πρέπει λοιπόν οι αποδείξεις να γίνονται με βάση αξιώματα και συμπερασματικούς κανόνες. Για τον καθολικό ποσοδείκτη υπάρχουν οι εξής κανόνες:

$$\frac{\forall x \bullet P(x)}{P(a)} \text{ (για τυχαίο } a) \quad \text{και} \quad \frac{P(a)}{\forall x \bullet P(x)} \text{ (για τυχαίο } a)$$

Ο πρώτος μας λέει: αν (έχω αποδείξει ότι) για κάθε x ισχύει η $P(x)$ τότε για οποιοδήποτε (τυχαίο) a ισχύει η $P(a)$. Και ο δεύτερος: αν (έχω αποδείξει ότι) για κάποιο τυχαίο (χωρίς ιδιαίτερα χαρακτηριστικά) a ισχύει η $P(a)$ τότε μπορώ να συμπεράνω ότι για κάθε x ισχύει η $P(x)$.

Για τον υπαρξιακό ποσοδείκτη:

$$\frac{\exists x \bullet P(x)}{P(q)} \quad \text{και} \quad \frac{P(q)}{\exists x \bullet P(x)}$$

Ο πρώτος μας λέει: αν (έχω αποδείξει ότι) υπάρχει ένα τουλάχιστον x ώστε να ισχύει η $P(x)$ τότε ισχύει $P(q)$, όπου q κάποια από τις συγκεκριμένες τιμές. Και ο δεύτερος: αν (έχω αποδείξει ότι) για κάποιο q ισχύει η $P(q)$ τότε μπορώ να συμπεράνω ότι υπάρχει τουλάχιστον ένα x ώστε να έχω $P(x)$.

Α.6 Ισότητα

Έστω ότι σε ένα σύνολο τιμών έχουμε ορίσει την **ισότητα** (==). Στο σύνολο αυτό:

1. ορίζεται και η **ανισότητα** (≠) ως εξής: με την $a \neq b$ εννοούμε $!(a == b)$
2. θα ισχύει το αξίωμα της **ανακλαστικότητας**: $\forall x \bullet x == x$
3. και ακόμη οι συμπερασματικοί κανόνες της **αντικατάστασης**: Αν $S(n)$ είναι ένας τύπος που περιέχει το n –αλλά όχι ως δεσμευμένη μεταβλητή– και $m == n$, τότε, μπορείς να αντικαταστήσεις το n με το m και να πάρεις έναν τύπο $S[m/n]$:

$$\frac{m = n, S(n)}{S[m/n]} \text{ (A1)} \quad \text{και} \quad \frac{n = m, S(n)}{S[m/n]} \text{ (A2)}$$

Για παράδειγμα, αν

- $S(y)$ είναι η $\exists x \bullet x^2 + y == 0$ και έχουμε αποδείξει ότι ισχύει και
- $z == y$

τότε ισχύει και η:

- $\exists x \bullet x^2 + z == 0$

Όταν λέμε «μπορείς να αντικαταστήσεις το n με το m» δεν εννούμε όλες υποχρεωτικά τις περιπτώσεις που εμφανίζεται το n, αλλά μόνον όσες μας ενδιαφέρουν.

Παράδειγμα 1

Ας δούμε πώς μπορούμε να αποδείξουμε τη **συμμετρικότητα** της ισότητας, δηλαδή:

αν ισχύει η: $x == y$ τότε ισχύει και η: $y == x$

1. $x == y$ υπόθεση
2. $x == x$ αξίωμα της ανακλαστικότητας

Έχουμε ότι x == y (από το 1.) και θεωρούμε ως $S(x)$ την x == x του 2. Σύμφωνα με τον κανόνα (Α2) αν αντικαταστήσουμε *μόνον το πρώτο* x με y θα πάρουμε μια πρόταση που ισχύει:

3. y == x.

ό.έ.δ.

Παράδειγμα 2

Απόδειξε ότι:

αν ισχύει η: (!($k \neq N$))&& ((y == 2k+1) && (x == (k+1)²))

τότε ισχύει και η: (k == N)) && ((y == 2N+1) && (x == (N+1)²))

1. (!($k \neq N$))&&((y == 2k+1)&&(x == (k+1)²))	υπόθεση
2. !($k \neq N$)	1. και σ.κ. (3) της §Α.4
3. !(!(k == N))	2. και ορισμός της ≠
4. k == N	3. και $A \equiv$ (!(!A)) §Α.3
5. (y == 2k+1) && (x == (k+1)²)	1. και σ.κ. (4) της §Α.4
6. (y == 2N+1) && (x == (N+1)²)	4.,5. και σ.κ. (Α2)
7. (k == N) && ((y == 2N+1) && (x == (N+1)²)	4.,6. και σ.κ. (1) της §Α.4

ό.έ.δ.

Α.7 Οι Δικές μας Αποδείξεις

Στις αποδείξεις ορθότητας των προγραμμάτων δεν θα είμαστε και τόσο αυστηροί (αν και στο Παρ. 2, της προηγούμενης παραγράφου, στο βήμα 4 τα πράγματα δεν είναι και τόσο αυστηρά).

Θα ασχολούμαστε κατά κανόνα με τύπους του Κ.Λ. και οι μεταβλητές μας θα παίρνουν τιμές στα σύνολα `int` (υποσύνολο του $\mathbb{Z}$) και `double` (υποσύνολο του $\mathbb{R}$).

Συχνά θα μας χρειάζονται οι ισοδυναμίες:

$$(a == b) \equiv (!(a \neq b))$$
$$(a \neq b) \equiv (!(a == b)) \equiv ((a < b) \;||\; (a > b))$$
$$(!(a < b)) \equiv (a \geq b) \equiv ((a > b) \;||\; (a == b))$$
$$(!(a > b)) \equiv (a \leq b) \equiv ((a < b) \;||\; (a == b))$$
$$((a \geq b) \;\&\&\; (a \leq b)) \equiv (a == b)$$
$$((a \geq b) \;||\; (a < b)) \equiv \texttt{true} \text{ και } ((a > b) \;||\; (a \leq b)) \equiv \texttt{true} \quad [(A \;||\; (!A)) \equiv \texttt{true}]$$
$$((a \geq b) \;\&\&\; (a < b)) \equiv \texttt{false} \text{ και } ((a > b) \;\&\&\; (a \leq b)) \equiv \texttt{false} \quad [(A \;\&\&\; (!A)) \equiv \texttt{false}]$$
$$(A \;\&\&\; (B \;||\; C)) \equiv ((A \;\&\&\; B) \;||\; (A \;\&\&\; C))$$
$$(A \;||\; (B \;\&\&\; C)) \equiv ((A \;||\; B) \;\&\&\; (A \;||\; C))$$
$$(!(A \;||\; B)) \equiv ((!A) \;\&\&\; (!B))$$
$$(!(A \;\&\&\; B)) \equiv ((!A) \;||\; (!B))$$

και οι συμπερασματικοί κανόνες (1)-(6) της §Α.4.

Ασκήσεις

Α-1 Απόδειξε ότι:

α) $(A \Rightarrow B) \equiv ((!\,A) \;||\; B)$ $\quad$ β) $A \Rightarrow (A \;||\; B)$ και $B \Rightarrow (A \;||\; B)$

Α-2 Απόδειξε ότι:

α) (A **xor** B) ≡ ((A || B) && (!(A && B)))

β) (A **xor** B) ≡ ((A && (!B)) || ((!A) && B))

γ) (A **xor** B) ≡ (!($A \Leftrightarrow B$))

παράρτημα

Β

Μαθηματικές Συναρτήσεις

Βάζοντας στο πρόγραμμά σου την οδηγία "`#include <cmath>`" μπορείς να χρησιμοποιήσεις τις μαθηματικές συναρτήσεις που σου δίνει η C++ και τις έχει κληρονομήσει από τη C.

Στον πίνακα, στις επόμενες δύο σελίδες, αντιγράψαμε τα χαρακτηριστικά αυτών των συναρτήσεων.

Όπως θα δεις, όλες σχεδόν οι συναρτήσεις υπάρχουν σε δύο μορφές: μια για τον τύπο `double` και μια για τον τύπο `long double`. Η *abs()* –για την απόλυτη τιμή– υπάρχει σε τέσσερις διαφορετικούς τύπους: *abs()* (`int`), *labs()* (`long`), *fabs()* (`double`), *fabsl()* (`long double`).

Να πώς θα διαβάζεις τον πίνακα: Στη στήλη 1 (Τι δίνει) βλέπεις την τιμή της συνάρτησης, π.χ. «τόξο εφαπτομένης». Στη στήλη 2 βλέπεις το ίδιο πράγμα συμβολικά, π.χ. «$τοξεφ\frac{a_1}{a_2}$». Στη στήλη 3 βλέπεις το πλήθος ορισμάτων, π.χ. «2». Αν το πλήθος είναι 1, στη στήλη 2 το όρισμα παριστάνεται ως *a*. Αν το πλήθος είναι 2, τότε το πρώτο όρισμα παριστάνεται ως a_1 και το δεύτερο ως a_2. Στις επόμενες τρεις στήλες, για το παράδειγμά μας βλέπεις τα εξής:

- Υπάρχει μια συνάρτηση με όνομα *atan2* (στ. 4), που παίρνει δύο ορίσματα τύπου `double` (στ. 5) και επιστρέφει τιμή τύπου `double` (στ. 6). Ακόμη:
- Υπάρχει μια συνάρτηση με όνομα *atan2l* (στ. 4), που παίρνει δύο ορίσματα τύπου `long double` (στ. 5) και επιστρέφει τιμή τύπου `long double` (στ. 6).

Αριθμητικές Συναρτήσεις της C++
Επικεφαλίδες στο cmath

Τι δίνει	Ορισμός	Πλ. Ορισ	Όνομα	Τύπος Ορίσματος	Τύπος Συνάρτησης

απόλυτη τιμή	$\lvert a \rvert$	1	*abs*
			labs
			fabs
			fabsl
				`int`
				`long`
				`double`
				`long double`
					`int`
					`long`
					`double`
					`long double`

στρογγ. στον πλησ. μικρότ. ακέραιο	$\lfloor a \rfloor$	1	*floor* *floorl*	`double` `long double`	`double` `long double`
στρογγ. στον πλησ. μεγαλ. ακέραιο.	$\lceil a \rceil$	1	*ceil* *ceill*	`double` `long double`	`double` `long double`
υπόλοιπο	a_1 **mod** a_2	2	*fmod* *fmodl*	`double` `long double`	`double` `long double`
τετραγ. ρίζα	$\sqrt{a}$	1	*sqrt* *sqrtl*	`double` `long double`	`double` `long double`
υποτείνουσα	$\sqrt{a_1^2 + a_2^2}$	2	*hypotl* *hypotl*	`double` `long double`	`double` `long double`
ημίτονο	ημ*a*	1	*sin* *sinl*	`double` `long double`	`double` `long double`
συνημίτονο	συν*a*	1	*cos* *cosl*	`double` `long double`	`double` `long double`
εφαπτομένη	εφ*a*	1	*tan* *tanl*	`double` `long double`	`double` `long double`
τόξο ημιτόνου	τοξημ*a*	1	*asin* *asinl*	`double` `long double`	`double` `long double`
τόξο συνημιτόνου	τοξσυν*a*	1	*acos* *acosl*	`double` `long double`	`double` `long double`
τόξο εφαπτομένης	τοξεφ*a*	1	*atan* *atanl*	`double` `long double`	`double` `long double`
τόξο εφαπτομένης	$\tau o \xi \varepsilon \varphi \frac{a_1}{a_2}$	2	*atan2* *atan2l*	`double` `long double`	`double` `long double`
εκθετική	e^a	1	*exp* *expl*	`double` `long double`	`double` `long double`
υπερβολικό ημίτονο	sinh*a*	1	*sinh* *sinhl*	`double` `long double`	`double` `long double`
υπερβολικό συνημίτονο	cosh*a*	1	*cosh* *coshl*	`double` `long double`	`double` `long double`
υπερβολική εφαπτομένη	tanh*a*	1	*tanh* *tanhl*	`double` `long double`	`double` `long double`
φυσικός λογάριθμος	`ln`*a*	1	*log* *logl*	`double` `long double`	`double` `long double`
δεκαδικός λογάριθμος	λογ*a*	1	*log10* *log10l*	`double` `long double`	`double` `long double`
δύναμη	$a_1^{a_2}$	2	*pow* *powl*	`double` `long double`	`double` `long double`
δύναμη του 10	10^a	1	*pow10* *pow10l*	`int` `int`	`double` `long double`

παράρτημα

C

Λέξεις-Κλειδιά της C++

Στον Πίν. C-1, παρακάτω, βλέπεις τις λέξεις-κλειδιά της C++ (που δεν επιτρέπεται να χρησιμοποιηθούν ως ονόματα) όπως δίνονται στο (ISO 2011). Οι λέξεις που σημειώνονται με αστερίσκο δεν υπήρχαν στο (ISO 2003).

Πίν. C-1: Λέξεις-κλειδιά της C++

alignas *	continue	friend	register	true
alignof *	decltype *	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert *	using
char	enum	new	static_cast	virtual
char16_t *	explicit	noexcept *	struct	void
char32_t *	export	nullptr *	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr *	float	protected	thread_local *	
const_cast	for	public	throw	

Στον επόμενο Πίν. C-2 δίνουμε λέξεις-κλειδιά της C++, που δεν υπήρχαν στη C και έχουν σχέση με λογικές πράξεις και ψηφιοπράξεις. Είναι πολύ πιθανόν ο μεταγλωττιστής που χρησιμοποιείς να μην αναγνωρίζει αυτές τις λέξεις. Η δυνατότητα παράστασης των τελεστών με λέξεις δόθηκε διότι σε ορισμένα εθνικά σύνολα χαρακτήρων, μερικά σύμβολα τελεστών έχουν αντικατασταθεί από γράμματα.

Πίν. C-2: Εναλλακτικά Λεξικά Σύμβολα της C++

εναλλακτικό	αρχικό	εναλλακτικό	αρχικό	εναλλακτικό	αρχικό
<%	{	and	&&	and_eq	&=
%>	}	bitor	|	or_eq	|=
<:	[	or	||	xor_eq	^=
:>	]	xor	^	not	!
%:	#	compl	~	not_eq	!=
%:%:	##	bitand	&		

Όπως βλέπεις όμως, στην πρώτη στήλη η εναλλακτική παράσταση δεν γίνεται με λέξεις αλλά με ζεύγη χαρακτήρων. Υπάρχουν και εναλλακτικές παραστάσεις με τριάδες χαρακτήρων. Τις βλέπεις στον πίνακα που ακολουθεί.

Πίν. C-3: Παράσταση Λεξικών Συμβόλων με 2 ή 3 Χαρακτήρες

2 χαρακτ	3 χαρακτ	αρχικό	2 χαρακτ	3 χαρακτ	αρχικό	2 χαρακτ	3 χαρακτ	αρχικό
%:	??=	#	<:	??(	[	<%	??<	{
	??/	\	:>	??)	]	%>	??>	}
	??'	^		??!	|		??-	~

παράρτημα

Δ

Πρότυπα Σύνολα Χαρακτήρων

Περιεχόμενα:

D.1 ISO 646 (ASCII) **1097**
D.2 Πρότυπο ΕΛΟΤ 928 ..... **1098**
D.3 Πρότυπο ΕΛΟΤ 927 ..... **1099**

D.1 ISO 646 (ASCII)

Το γνωστό ASCII (American Standard Code for Information Interchange) είναι μια εθνική εκδοχή του διεθνούς προτύπου ISO 646. Αποτελείται από 128 χαρακτήρες που αριθμούνται από το 0 μέχρι το 127. Οι πρώτοι 32 χαρακτήρες (0 .. 31) και ο τελευταίος (127) δεν είναι εκτυπώσιμοι (Πίν. Δ-1).

Στις θέσεις 48 μέχρι 57 βρίσκονται τα ψηφία του δεκαδικού συστήματος, κατ' αύξουσα τάξη:

`static_cast<char>`(48)=`'0'` και `static_cast<char>`(57)=`'9'`

Στις θέσεις 65 μέχρι 90 βρίσκονται τα κεφαλαία γράμματα του λατινικού αλφαβήτου σε αλφαβητική διάταξη. Στις θέσεις 97 μέχρι 122 βρίσκονται τα πεζά γράμματα του λατινικού αλφαβήτου. Η θέση ενός κεφαλαίου γράμματος από το αντίστοιχο πεζό διαφέρει κατά 32:

`static_cast<int>(`κεφαλαίο`) == static_cast<int>(`πεζό`)-`32

Οι χαρακτήρες των θέσεων 0 μέχρι 31 δίνονται από το πληκτρολόγιο ως εξής: με πιεσμένο το πλήκτρο <Ctrl> πιέζουμε τον χαρακτήρα που βρίσκεται στο σύνολο μετά από 64 θέσεις. Π.χ. αν θέλουμε να δώσουμε τον χαρακτήρα <ETX> = `static_cast<char>`(3), με πιεσμένο το πλήκτρο <Ctrl> πιέζουμε το πλήκτρο <C> ('C' = `static_cast<char>`(67)). Αυτή η πληκτρολόγηση συμβολίζεται και ως <Ctrl-C> ή "^C". Το τι θα πετύχεις με μια τέτοια πληκτρολόγηση δεν καθορίζεται από κανένα πρότυπο, αλλά εξαρτάται από το πρόγραμμα που θα δεχτεί την πληκτρολόγηση.

Αν προσπαθήσεις να «γράψεις» μερικούς από αυτούς τους χαρακτήρες, δεν θα δεις κάποιο σύμβολο να εμφανίζεται στην οθόνη σου αλλά θα κάποια άλλη λειτουργία. Π.χ. συνηθέστατα γράψιμο των παρακάτω χαρακτήρων έχουν τα εξής αποτελέσματα:

BEL (`static_cast<char>`(7)):

«Γράφοντας» αυτόν τον χαρακτήρα –`cout<<static_cast<char>(7)` ή `cout << '\a'`– στην οθόνη του τερματικού μας, μάλλον δεν θα δούμε κάτι αλλά θα το ακούσουμε: θα ηχήσει το μεγάφωνό του (alert).

BS (`static_cast<char>`(8)):

«Γράψιμο» αυτού του χαρακτήρα –`cout<<static_cast<char>(8)` ή `cout << '\b'`– έχει ως αποτέλεσμα να σβηστεί ο προηγούμενος χαρακτήρας που είχε γραφεί (Back Space).

```
  0 NUL  16 DLE 32 SP  48 0   64 @   80 P   96 `  112 p
  1 SOH  17 DC1 33 !   49 1   65 A   81 Q   97 a  113 q
  2 STX  18 DC2 34 "   50 2   66 B   82 R   98 b  114 r
  3 ETX  19 DC3 35 #   51 3   67 C   83 S   99 c  115 s
  4 EOT  20 DC4 36 $   52 4   68 D   84 T  100 d  116 t
  5 ENQ  21 NAK 37 %   53 5   69 E   85 U  101 e  117 u
  6 ACK  22 SYN 38 &   54 6   70 F   86 V  102 f  118 v
  7 BEL  23 ETB 39 '   55 7   71 G   87 W  103 g  119 w
  8 BS   24 CAN 40 (   56 8   72 H   88 X  104 h  120 x
  9 HT   25 EM  41 )   57 9   73 I   89 Y  105 i  121 y
 10 LF   26 SUB 42 *   58 :   74 J   90 Z  106 j  122 z
 11 VT   27 ESC 43 +   59 ;   75 K   91 [  107 k  123 {
 12 FF   28 FS  44 ,   60 <   76 L   92 \  108 ℓ  124 |
 13 CR   29 GS  45 -   61 =   77 M   93 ]  109 m  125 }
 14 SO   30 RS  46 .   62 >   78 N   94 ^  110 n  126 ~
 15 SI   31 US  47 /   63 ?   79 O   95 _  111 o  127 DEL
```

Πίν. Δ-1 Πρότυπο Σύνολο Χαρακτήρων ISO 646 (ASCII).

HT (`static_cast<char>`(9)):
Με `cout<<static_cast<char>(9)` ή `cout << '\t'` συνεχίζεις το γράψιμο από τον επόμενο οριζόντιο στηλοθέτη (Horizontal Tab).

LF (`static_cast<char>`(10)):
Με `cout<<static_cast<char>(10)` ή `cout << '\n'` κατεβάζεις τον οθονοδείκτη στην επόμενη γραμμή (Line Feed).

VT (`static_cast<char>`(11)):
Με `cout<<static_cast<char>(11)` ή `cout << '\v'`– συνεχίζεις το γράψιμο από τον επόμενο κατακόρυφο στηλοθέτη (Vertical Tab).

FF (`static_cast<char>`(12)):
Με `cout<<static_cast<char>(12)` ή `cout << '\f'` καθαρίζεις την οθόνη (στον εκτυπωτή αλλάζεις σελίδα) (Form Feed).

CR (`static_cast<char>`(13)):
Με `cout << static_cast<char>(13)` ή `cout << '\r'` φέρνεις τον οθονοδείκτη στην αρχή της γραμμής (Carriage Return).

DEL (`static_cast<char>`(127)):
Σβήνει τη γραμμή που βρίσκεται (DElete Line).

Ο `static_cast<char>`(26) (<Ctrl>Z), στο MS DOS, σημειώνει το τέλος ενός αρχείου text. Σε άλλα συστήματα το ίδιο πράγμα γίνεται με τον <ETX> (`static_cast<char>`(3) = ^C - End of TeXt).

Τα συμβολικά ονόματα των χαρακτήρων ελέγχου ξεκινούν από τη λειτουργία τους στις επικοινωνίες.

D.2 Πρότυπο ΕΛΟΤ 928

Το Πρότυπο ΕΛΟΤ 928 (Πίν. Δ-3), βγήκε από τη Τεχνική Επιτροπή 48 που λειτουργεί με ευθύνη των ΕΛΟΤ και ΕΛΚΕΠΑ.

Στις θέσεις 0..31 και 127..159 δεν υπάρχουν εκτυπώσιμοι χαρακτήρες, σύμφωνα με τους κανόνες του ISO και με τα πρότυπα της σειράς ISO 8859. Θα τονίσουμε την παρουσία δυο χαρακτήρων:

NBSP (`static_cast<char>`(160)): No-Break SPace = Διάστημα Χωρίς Διακοπή. "Γράφεται" όπως το *διάστημα*. Η παρουσία του δείχνει σε προγράμματα **μορφοποίησης κειμένων** (text formatters) ότι στο σημείο αυτό δεν θα πρέπει να γίνει διακοπή του κειμένου με αλλαγή γραμμής.

SHY (`static_cast<char>`(173)): Soft HYphen = Μη-Σταθερό Ενωτικό. Γράφεται όπως η παύλα (`static_cast<char>`(45)). Η παρουσία του δείχνει σε προγράμματα μορφοποίησης

0	NUL	32	SP	64	@	96	`	128	□	160	NBSP	192	Ἱ	224	ὗ
1	SOH	33	!	65	A	97	a	129	□	161	‘	193	A	225	α
2	STX	34	"	66	B	98	b	130	□	162	’	194	B	226	β
3	ETX	35	#	67	C	99	c	131	□	163	£	195	Γ	227	γ
4	EOT	36	$	68	D	100	d	132	□	164	¤	196	Δ	228	δ
5	ENQ	37	%	69	E	101	e	133	□	165	¥	197	E	229	ε
6	ACK	38	&	70	F	102	f	134	□	166	¦	198	Z	230	ζ
7	BEL	39	'	71	G	103	g	135	□	167	§	199	H	231	η
8	BS	40	(	72	H	104	h	136	□	168	¨	200	Θ	232	θ
9	HT	41	)	73	I	105	i	137	□	169	©	201	I	233	ι
10	LF	42	*	74	J	106	j	138	□	170	□	202	K	234	κ
11	VT	43	+	75	K	107	k	139	□	171	«	203	Λ	235	λ
12	FF	44	,	76	L	108	ℓ	140	□	172	¬	204	M	236	μ
13	CR	45	-	77	M	109	m	141	□	173	SHY	205	N	237	ν
14	SO	46	.	78	N	110	n	142	□	174	®	206	Ξ	238	ξ
15	SI	47	/	79	O	111	o	143	□	175	―	207	O	239	ο
16	DLE	48	0	80	P	112	p	144	□	176	°	208	Π	240	π
17	DC1	49	1	81	Q	113	q	145	□	177	±	209	P	241	ρ
18	DC2	50	2	82	R	114	r	146	□	178	²	210	□	242	ς
19	DC3	51	3	83	S	115	s	147	□	179	³	211	Σ	243	σ
20	DC4	52	4	84	T	116	t	148	□	180	΄	212	T	244	τ
21	NAK	53	5	85	U	117	u	149	□	181	΅	213	Y	245	υ
22	SYN	54	6	86	V	118	v	150	□	182	Ά	214	Φ	246	φ
23	ETB	55	7	87	W	119	w	151	□	183	·	215	X	247	χ
24	CAN	56	8	88	X	120	x	152	□	184	Έ	216	Ψ	248	ψ
25	EM	57	9	89	Y	121	y	153	□	185	Ή	217	Ω	249	ω
26	SUB	58	:	90	Z	122	z	154	□	186	Ί	218	Ϊ	250	ϊ
27	ESC	59	;	91	[	123	{	155	□	187	»	219	Ÿ	251	ü
28	FS	60	<	92	\	124	|	156	□	188	Ό	220	ὰ	252	ὸ
29	GS	61	=	93	]	125	}	157	□	189	½	221	έ	253	ὺ
30	RS	62	>	94	^	126	~	158	□	190	Ύ	222	ή	254	ώ
31	US	63	?	95	_	127	DEL	159	□	191	Ώ	223	ί	255	□

Πίν. Δ-2 Πρότυπο Σύνολο Χαρακτήρων ΕΛΟΤ 928.

κειμένων ότι στο σημείο αυτό μπορεί να εισαχθεί ή να αφαιρεθεί μια παύλα χωρισμού λέξης (τέλος συλλαβής).

Στο περιβάλλον MS Windows χρησιμοποιείται σύνολο χαρακτήρων που είναι παραλλαγή του ΕΛΟΤ 928 (Πίν. Δ-4). Οι διαφορές βρίσκονται κυρίως στη θέση του 'Ά' και στο ότι στις «απαγορευμένες θέσεις» 128 .. 159 υπάρχουν εκτυπώσιμοι χαρακτήρες. Στο περιβάλλον Windows χρησιμοποιούνται πολυγλωσσικοί πίνακες χαρακτήρων (με βάση τα πρότυπα ISO 8859 και τις ταυτότητες των συμβόλων) και η προσαρμογή τους σε συγκεκριμένη γλώσσα γίνεται με κατάλληλα προγράμματα.

D.3 Πρότυπο ΕΛΟΤ 927

Το Πρότυπο ΕΛΟΤ 927 έγινε για να καλύψει τις ανάγκες παλιών ΗΥ και εκτυπωτών που έχουν σύνολο 128 χαρακτήρων και πρέπει να έχουν δυνατότητα εκτύπωσης (τουλάχιστον κεφαλαίων) ελληνικών και λατινικών χαρακτήρων. Πρακτικά έχει πια αχρηστευθεί.

παράρτημα

Ε

Η Προτεραιότητα των Πράξεων

Στον πίνακα, που δίνεται στις επόμενες σελίδες, βλέπεις τα χαρακτηριστικά των πράξεων που έχουμε μάθει μέχρι τώρα. Οι πράξεις δίνονται κατά τη σειρά προτεραιότητας.

Δεν βλέπεις την προσεταιριστικότητα των πράξεων, αλλά γι' αυτήν ο κανόνας είναι απλός:

- η προσεταιριστικότητα των δυϊκών πράξεων, εκτός από την εκχώρηση, είναι από τα αριστερά προς τα δεξιά,
- η προσεταιριστικότητα των ενικών πράξεων και της εκχώρησης είναι από τα δεξιά προς τα αριστερά.

Για παράδειγμα: η `a + b + c` υπολογίζεται ως `(a + b) + c`, ενώ η `a = b = c` υπολογίζεται ως `a = (b = c)`.

Αλλά προσοχή! Αν έχουμε `a*b + c/d + f(e)` ο παραπάνω κανόνας μας λέει *μόνον* ότι ο υπολογισμός θα γίνει ως εξής: `(a*b + c/d) + f(e)`.

- *Δεν μας εγγυάται* ότι πρώτα θα υπολογιστεί η `(a*b + c/d)` και μετά η `f(e)`.
- *Δεν μας εγγυάται* ότι πρώτα θα υπολογιστεί η `a*b` και μετά η `c/d`.

Κάθε υλοποίηση της C++ θα κάνει τους υπολογισμούς με τη σειρά που "θέλει". Αν θέλεις να επιβάλεις συγκεκριμένη σειρά εκτέλεσης των πράξεων θα πρέπει να σπάσεις την παράσταση σε μικρότερες και να τις γράψεις με την κατάλληλη σειρά.

Όταν υπολογίζεται μια αριθμητική παράσταση γίνονται και ορισμένες μετατροπές τύπων, που θα τις ονομάζουμε *Συνήθεις Αριθμητικές Μετατροπές* (ΣΑΜ). Ας πούμε ότι έχουμε κάποια πράξη a *θ* b. Τότε:

1. Αν κάποια από τις τιμές a, b είναι τύπου `long double`, τότε μετατρέπεται στον τύπο `long double` και η άλλη τιμή.
2. Αλλιώς, αν κάποια από τις τιμές a, b είναι τύπου `double`, τότε μετατρέπεται στον τύπο `double` και η άλλη τιμή.
3. Αλλιώς, αν κάποια από τις τιμές a, b είναι τύπου `float`, τότε μετατρέπεται στον τύπο `float` και η άλλη τιμή.
4. Αλλιώς, τα a και b (είναι ακέραιου τύπου) υφίστανται **ακέραιη προαγωγή** (integral promotion), δηλαδή: αν κάποια (ή και οι δύο) από τις τιμές a, b είναι "μικρού ακέραιου τύπου" (`char`, `short int`, `signed` ή `unsigned`), αυτή μετατρέπεται σε τύπο `int` ή `unsigned int` (αν δεν μπορεί να παρασταθεί στον `int`) και στη συνέχεια

   αν κάποια από τις τιμές a, b είναι τύπου `unsigned long`, τότε

   μετατρέπεται στον τύπο `unsigned long` και η άλλη τιμή.

Πράξεις και Προτεραιότητες		
Προτερ.	**Τελεστής**	**Σύμβολα**
0	Παράστ. σε παρενθέσεις	`(`*παράσταση*`)`
1	Επίλυση Εμβέλειας Καθολικό	*όνομα κλάσης*`::`*μέλος* *όνομα ονοματοχώρου*`::`*μέλος* `::`*αναγνωριστικό*

2	Επιλογή μέλους
	Στοιχείο Πίνακα (δείκτης)
	Κλήση Συνάρτησης
	Κατασκευή τιμής
	Μεταθεματικές Παραστάσεις
	Χαρακτηριστικά Τύπου
	Ελεγχ. μετατρ. τύπου (εκτ)
	Μετατροπή τύπου (μετγλ.)
	Μη ελεγχ. μετατρ. τύπου(εκτ)
	Μετατροπή const
		αντικείμενο`.`*μέλος*, *βέλος*`->`*μέλος*
		βέλος`[]`
		όνομα`(`*λίστα παραστάσεων*`)`
		τύπος`(`*λίστα παραστάσεων*`)`
		τιμή-`l++`
		τιμή-`l--`
		`typeid(`*παράσταση*`)`, `typeid(`*τύπος*`)`
		`dynamic_cast<` *τύπος*`>(`*παράσταση*`)`
		`static_cast<` *τύπος*`>(`*παράσταση*`)`
		`reinterpret_cast<`*τύπος*`>(`*παράσταση*`)`
		`const_cast<`*τύπος*`>(`*παράσταση*`)`

3	Μέγεθος
	Προθεματικές Παραστ.
	Συμπλήρωμα (ψηφιοπρ.)
	Λογική Άρνηση
	Πρόσημα
	Διεύθυνση
	Αποπαραπομπή
	Δημιουργία Δυν. Μετ.
	Δημ Δυν. Μετ. με αρχ. τιμή
	Δημιουργία σε θέση
	Δημ. σε θέση με αρχ. τιμή
	Καταστροφή
	Καταστροφή πίνακα
	Τυποθεώρηση (C)
		`sizeof` *παράσταση*, `sizeof(`*τύπος*`)`
		`++` *τιμή*-`l`, `--` *τιμή*-`l`
		`~` *παράσταση*
		`!` *παράσταση*
		`-` *παράσταση*, `+` *παράσταση*
		`&` *όνομα*
		`*` *παράσταση*
		`new` *τύπος*
		`new` *τύπος* `(`*παράσταση*`)`
		`new (`*παράσταση*`)` *τύπος*
		`new (`*παράσταση*`)` *τύπος* `(`*παράσταση*`)`
		`delete` *βέλος*
		`delete []` *βέλος*
		`(`*όνομα*`)` *παράσταση*

4	Επιλογή μέλους	*αντικείμενο*`.*`*βέλος προς μέλος* *βέλος*`->*`*βέλος προς μέλος*
5	Πολλαπλασιασμός Διαίρεση Υπόλοιπο	*παράσταση* `*` *παράσταση* *παράσταση* `/` *παράσταση* *παράσταση* `%` *παράσταση*
6	Πρόσθεση Αφαίρεση	*παράσταση* `+` *παράσταση* *παράσταση* `-` *παράσταση*

5. Αλλιώς, αν κάποια από τις τιμές a, b είναι τύπου `long int` και η άλλη `unsigned int`, τότε

   αν κάθε τιμή `unsigned int` μπορεί να παρασταθεί σε `long int`

   η τιμή `unsigned int` μετατρέπεται στον τύπο `long int`

   αλλιώς

   και οι δύο μετατρέπονται σε `unsigned long int`.

6. Αλλιώς, αν κάποια από τις τιμές a, b είναι τύπου `long`, τότε μετατρέπεται σε `long` και η άλλη τιμή.

7. Αλλιώς, αν κάποια από τις τιμές a, b είναι τύπου `unsigned`, τότε μετατρέπεται στον τύπο `unsigned` και η άλλη τιμή.

8. Αλλιώς και οι δυο τιμές είναι τύπου `int`.

Πράξεις και Προτεραιότητες (συνέχεια)		
Προτερ.	**Τελεστής**	**Σύμβολα**
5	Πολλαπλασιασμός Διαίρεση Υπόλοιπο	*παράσταση* ***** *παράσταση* *παράσταση* **/** *παράσταση* *παράσταση* **%** *παράσταση*
6	Πρόσθεση Αφαίρεση	*παράσταση* **+** *παράσταση* *παράσταση* **-** *παράσταση*
7	Ολίσθηση αριστερά Ολίσθηση δεξιά	*παράσταση* **<<** *παράσταση* *παράσταση* **>>** *παράσταση*
8	Μικρότερο Μικρότερο ή ίσο Μεγαλύτερο Μεγαλύτερο ή ίσο	*παράσταση* **<** *παράσταση* *παράσταση* **<=** *παράσταση* *παράσταση* **>** *παράσταση* *παράσταση* **>=** *παράσταση*
9	Ίσο Διάφορο	*παράσταση* **==** *παράσταση* *παράσταση* **!=** *παράσταση*
10	Ψηφιοπράξη **and**	*παράσταση* **&** *παράσταση*
11	Ψηφιοπράξη **xor**	*παράσταση* **^** *παράσταση*
12	Ψηφιοπράξη **or**	*παράσταση* **|** *παράσταση*
13	Λογικό **and**	*παράσταση* **&&** *παράσταση*
14	Λογικό **or**	*παράσταση* **||** *παράσταση*
15	Απλή εκχώρηση Πολλαπλασιαστική Εκχ. Διαιρετική Εκχώρηση Διαίρεση Υπολοίπου Προσθετική Εκχώρηση Αφαιρετική Εκχώρηση Εκχώρ. με αριστ. ολίσθηση Εκχώρ. με δεξιά ολίσθηση Εκχώρ. με ψηφιοπρ. **and** Εκχώρ. με ψηφιοπρ. **or** Εκχώρ. με ψηφιοπρ. **xor**	*τιμή-l* **=** *παράσταση* *τιμή-l* ***=** *παράσταση* *τιμή-l* **/=** *παράσταση* *τιμή-l* **%=** *παράσταση* *τιμή-l* **+=** *παράσταση* *τιμή-l* **-=** *παράσταση* *τιμή-l* **<<=** *παράσταση* *τιμή-l* **>>=** *παράσταση* *τιμή-l* **&=** *παράσταση* *τιμή-l* **|=** *παράσταση* *τιμή-l* **^=** *παράσταση*
16	Υπό Συνθήκη	*παράσταση* **?** *παράσταση* **:** *παράσταση*
17	Εξαίρεση	**throw** *παράσταση*
18	Ακολουθία Παραστάσεων	*παράσταση* **,** *παράσταση*

Οι ΣΑΜ εφαρμόζονται όταν γίνονται οι πράξεις **+**, **-**, *****, **/** μεταξύ αριθμητικών τιμών και καθορίζουν τον τύπο του αποτελέσματος. Ακέραιη προαγωγή γίνεται και στις ενικές πράξεις **+**, **-** (πρόσημα). Ο τύπος του αποτελέσματος είναι αυτός που προκύπτει από την προώθηση.