κεφάλαιο

5

Επιλογές

Ο στόχος μας σε αυτό το κεφάλαιο:

Να κάνεις το πρώτο βήμα στη χρήση συνθηκών για τον έλεγχο εκτέλεσης ενός προγράμματος: Να μπορείς να επιλέγεις ομάδες εντολών που θα εκτελεσθούν ή δεν θα εκτελεσθούν.

Προσδοκώμενα αποτελέσματα:

Θα μπορείς να γράφεις προγράμματά πιο ευέλικτα, που η πορεία της εκτέλεσής τους θα εξαρτάται από τιμές συνθηκών και των μεταβλητών που περιλαμβάνονται σε αυτές. Θα μπορείς να ελέγχεις αν ισχύει η προϋπόθεση όταν αρχίζει η εκτέλεση του προγράμματος.

Έννοιες κλειδιά:

- *εντολές επιλογής*
- *εντολή* `if`
- *εντολή* `ifelse`
- *σύνθετη εντολή*
- *πολλαπλές επιλογές*
- *λογική εντολών* `if` *και* `ifelse`

Περιεχόμενα:

5.1 Επιλογή - Οι Εντολές if 112
5.2 Η Εντολή if 115
5.3 Φωλιασμένες if - Πολλαπλές Επιλογές 117
5.4 * Η Λογική των Εντολών Επιλογής 120
5.4.1 * Από το Τέλος προς την Αρχή 122
5.5 Εξασφάλιση Προϋποθέσεων 124
5.6 Σειρά Εκτέλεσης των Πράξεων 127
5.7 Τα " ; " και Άλλα Λάθη 128
5.8 Τι (Πρέπει να) Έμαθες 129
Ασκήσεις 129
Α Ομάδα 129
Β Ομάδα 130
Γ Ομάδα 131

Εισαγωγικές Παρατηρήσεις:

Οι εντολές που γνωρίσαμε στα προηγούμενα κεφάλαια, δηλ. οι εντολές εισόδου/εξόδου, οι εντολές δήλωσης και η εντολή εκχώρησης μας δίνουν τη δυνατότητα να γράφουμε απλά προγράμματα υπολογισμών, που όμως δεν έχουν δυνατότητες για επιλογή «διαδρομών», στη διάρκεια της εκτέλεσης. Αυτό σημαίνει ότι, όταν εκτελείται το πρόγραμμα, δεν υπάρχει δυνατότητα «παράκαμψης» κάποιων εντολών του προγράμματος και επομένως *όλες οι*

εντολές εκτελούνται υποχρεωτικά με την ίδια σειρά που είναι γραμμένες στο πρόγραμμα, η μια μετά την άλλη.

Φυσικά, αυτός ο περιορισμός δεν είναι βολικός στην πράξη και στις γλώσσες προγραμματισμού υπάρχουν ειδικές εντολές, που μας δίνουν τη δυνατότητα να ζητούμε την εκτέλεση ή την παράκαμψη εντολών που υπάρχουν στο πρόγραμμά μας.

Στη C++ τέτοιες εντολές είναι οι **εντολές επιλογής** (selection statements). Στο κεφάλαιο αυτό, θα ασχοληθούμε ειδικά με τέτοιες εντολές.

5.1 Επιλογή – Οι Εντολές if

Στο προηγούμενο κεφάλαιο λέγαμε ότι «*μπορούμε ... να ρυθμίζουμε την εκτέλεση του προγράμματός μας με βάση τις τιμές συνθηκών.*» Εδώ θα μάθουμε έναν τρόπο για να κάνουμε κάτι τέτοιο. Ξεκινούμε με ένα παράδειγμα.

Θέλουμε ένα πρόγραμμα που θα διαβάζει δύο πραγματικούς αριθμούς, από το πληκτρολόγιο, και θα γράφει τον μεγαλύτερο από αυτούς μαζί με το μήνυμα `" Μεγαλύτερος είναι ο αριθμός: "`. Ας πούμε ότι δηλώνουμε:

```
   double x, y, maxxy;
```

και διαβάζουμε:

```
   cin >> x >> y;
```

Στο τέλος θα γράψουμε:

```
   cout << " Μεγαλύτερος είναι ο αριθμός: " << maxxy << endl;
```

Το πρόβλημά μας είναι να δώσουμε στη *maxxy* τη σωστή τιμή. Ένας απλός τρόπος, που θα σκέφτονταν πολλοί, είναι ο εξής:

αν $x > y$ **τότε** { **βάλε** *maxxy* ← *x*
αλλιώς { **βάλε** *maxxy* ← *y*

Τα «**βάλε** *maxxy* ← *x*» και «**βάλε** *maxxy* ← *y*» ξέρουμε να τα γράψουμε σε C++. Τα υπόλοιπα; Σχεδόν μεταφράζουμε στα αγγλικά:

```
   if ( x > y )  maxxy = x;
          else  maxxy = y;
```

Να ολόκληρο το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   double x, y, maxxy;

   cin >> x >> y;
   if (x > y)  maxxy = x;
         else  maxxy = y;
   cout << " Μεγαλύτερος είναι ο αριθμός: " << maxxy << endl;
}
```

Η εντολή **ifelse** της C++ έχει γενικώς τη μορφή:

```
if ( S )  Et  else  Ef
```

όπου *S* μια συνθήκη (λογική παράσταση) και E_t, E_f εντολές. Η εντολή εκτελείται ως εξής:

- υπολογίζεται η τιμή της λογικής παράστασης *S*,
- αν η τιμή είναι `true` (αν η συνθήκη ισχύει) τότε α) εκτελείται η εντολή E_t και στη συνέχεια β) αγνοείται η εντολή E_f και η εκτέλεση συνεχίζεται με την εντολή που ακολουθεί την **ifelse**,
- αν η τιμή είναι `false` (αν η συνθήκη δεν ισχύει) τότε α) αγνοείται η εντολή E_t, β) εκτελείται η εντολή E_f και η εκτέλεση συνεχίζεται με την εντολή που ακολουθεί την **ifelse**.

Αυτά φαίνονται και στο λογικό διάγραμμα της **ifelse** στο Σχ. 5-1. Ο ρόμβος δείχνει το σχήμα «απόφασης» –επιλογής διαδρομής– που λαμβάνεται ανάλογα με την τιμή (`true` ή `false`) της παράστασης *S*. Από το σχήμα μπορείς να καταλάβεις ακόμη γιατί οι εντολές **ifelse** (και οι **if** που θα δούμε στη συνέχεια) λέγονται και εντολές **διακλάδωσης** (branching).

Ερχόμαστε στο παράδειγμά μας και ας πούμε ότι όταν εκτελείται η "`cin >> x >> y`" απαντούμε με:

```
1.6   0.6
```

Το "`1.6`" γίνεται τιμή της *x* και το "`0.6`" γίνεται τιμή της *y*. Στη συνέχεια εκτελείται η **ifelse** και κατ' αρχάς υπολογίζεται η συνθήκη "`x > y`" ή "1.6 > 0.6", που είναι `true`. Ύστερα από αυτό εκτελείται η εντολή "`maxxy = x`" και στη συνέχεια η εντολή που ακολουθεί την **ifelse**, δηλαδή η "`cout << ...`" που δίνει:

```
Μεγαλύτερος είναι ο αριθμός: 1.6
```

Η "`maxxy = y`" είναι σαν να μην υπάρχει.

Αν στη "`cin >> ...`" απαντήσουμε με:

```
1.6   2.6
```

η "`x > y`" (1.6 > 2.6) θα πάρει τιμή `false`. Στην περίπτωση αυτή θα εκτελεσθεί η εντολή "`maxxy = y`" και στη συνέχεια η "`cout << ...`" που δίνει:

```
Μεγαλύτερος είναι ο αριθμός: 2.6
```

Το ίδιο θα συμβεί και στην περίπτωση που απαντούμε στη "`cin >> ...`" με:

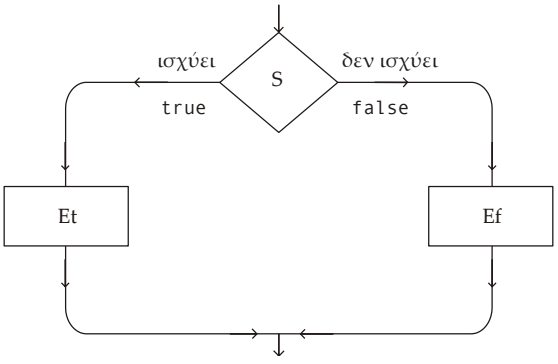
```
2.6   2.6
```

αφού και πάλι η "`x > y`" (2.6 > 2.6) θα πάρει τιμή `false`.

Ας ξαναδούμε τώρα το παράδειγμα 3 της §2.3. Κάναμε ένα πρόγραμμα που έβρισκε τις ρίζες της εξίσωσης $ax^2 + bx + c = 0$. Αλλά όταν η διακρίνουσα ήταν αρνητική είχαμε πρόβλημα. Τώρα μπορούμε να κάνουμε ένα καλύτερο πρόγραμμα: οι εντολές

```
x1 = (-b + sqrt(delta))/(2*a);
x2 = (-b - sqrt(delta))/(2*a);
cout << " Λύση1 = " << x1 << "   Λύση2 = " << x2 << endl;
```

θα πρέπει να εκτελούνται μόνον αν *delta* ≥ 0 αλλιώς θα πρέπει να βγαίνει ένα μήνυμα που



Σχ. 5-1 Λογικό διάγραμμα για την εντολή **ifelse**.

θα λέει ότι η εξίσωση δεν έχει πραγματικές λύσεις. Θα βάλουμε λοιπόν μια **ifelse**, αλλά… Μάθαμε παραπάνω ότι μετά τη συνθήκη μπορούμε να γράψουμε μια εντολή και εδώ έχουμε τρεις· τι κάνουμε τώρα;

Η C++ μας προσφέρει την εξής δυνατότητα: Να δημιουργήσουμε μια **σύνθετη εντολή** (compound statement) που θα περιλαμβάνει τις τρεις εντολές που μας ενδιαφέρουν. Αυτό γίνεται ως εξής: «συσκευάζουμε» τις εντολές μας με ένα άγκιστρο (**{**) στην αρχή και ένα άγκιστρο (**}**) στο τέλος. Στην περίπτωσή μας:

```
   {
      x1 = (-b + sqrt(delta))/(2*a);
      x2 = (-b - sqrt(delta))/(2*a);
      cout << " Λύση1 = " << x1 << "   Λύση2 = " << x2 << endl;
   }
```

Να πώς γίνεται το πρόγραμμά μας:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   double  a, b, c;    // οι συντελεστές της εξίσωσης
   double  x1, x2;     // οι ρίζες της εξίσωσης
   double  delta;      // η διακρίνουσα

   cout << "Δώσε τρεις πραγματικούς αριθμούς" << endl;
   cin >> a >> b >> c;
   cout << " a = " << a << "   b = " << b
        << "   c = " << c << endl;
   delta = b*b - 4*a*c;
   if ( delta >= 0 )
   {
      x1 = (-b + sqrt(delta))/(2*a);
      x2 = (-b - sqrt(delta))/(2*a);
      cout << " Λύση1 = " << x1 << "   Λύση2 = " << x2 << endl;
   }
   else
      cout << " Δεν υπάρχουν Πραγματικές Λύσεις" << endl;
   cout << " ΤΕΛΟΣ" << endl;
}
```

Αν τροφοδοτήσουμε το πρόγραμμα με τους παρακάτω τρείς αριθμούς (συντελεστές της εξίσωσης):

```
            1   3   -10
```

θα πάρουμε το εξής αποτέλεσμα:

```
 a = 1   b = 3   c = -10
 Λύση1 = 2   Λύση2 = -5
 ΤΕΛΟΣ
```

Αν όμως πληκτρολογήσουμε τους αριθμούς:

```
            2   3   4
```

θα πάρουμε τα εξής:

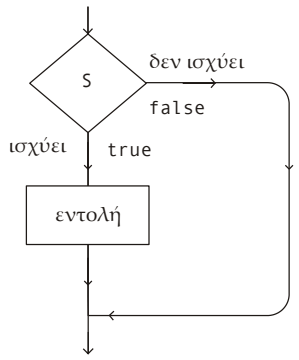
```
 a = 2   b = 3   c = 4
 Δεν υπάρχουν Πραγματικές Λύσεις
 ΤΕΛΟΣ
```

Ζητάμε άλλη μια εκτέλεση, δίνοντας για στοιχεία εισόδου:

```
    0   1   2
```

Αποτέλεσμα: διακοπή εκτέλεσης του προγράμματος και κάποιο «κακό» μήνυμα. Γιατί; Το "**0**" έγινε τιμή της *a* και στον υπολογισμό των *x1* και *x2* διαιρούμε δια 2*a*. Χρειαζόμαστε λοιπόν άλλον έναν έλεγχο, άλλη μια **if**. Θα επανέλθουμε στη συνέχεια.

5.2 Η Εντολή if



Σχ. 5-2 Λογικό διάγραμμα για την εντολή **if**.

Ξαναγυρνάμε στο πρόβλημα του μεγίστου· πολλοί θα προτιμούσαν να δώσουν μια κάπως διαφορετική λύση:

αν *x* > *y* **τότε** { **βάλε** *maxxy* ← *x*

αν *x* ≤ *y* **τότε** { **βάλε** *maxxy* ← *y*

Αυτό πώς το γράφουμε σε C++; Μια πρώτη σκέψη είναι η εξής:

```
if (x > y)  maxxy = x; else;
if (x <= y)  maxxy = y; else;
```

που στην πραγματικότητα μεταφράζει το:

αν *x* > *y* **τότε** { **βάλε** *maxxy* ← *x*

αλλιώς { μη κάνεις τίποτε

αν *x* ≤ *y* **τότε** { **βάλε** *maxxy* ← *y*

αλλιώς { μη κάνεις τίποτε

Μετά το `else` θεωρείται ότι υπάρχει η **κενή** (empty) ή **μηδενική** (null) εντολή που αντιπροσωπεύει στη C++ αυτό το «*μη κάνεις τίποτε*» του ψευδοκώδικα.

Πάντως μας δίνεται και άλλη δυνατότητα, χωρίς εκείνο το "`else;`":

```
if (x > y)  maxxy = x;
if (x <= y)  maxxy = y;
```

Εδώ χρησιμοποιούμε την απλή **if** που έχει γενικώς τη μορφή:

`if ( S )  Et`

όπου *S* μια συνθήκη (λογική παράσταση) και E_t εντολή. Η εντολή εκτελείται ως εξής:

- υπολογίζεται η τιμή της λογικής παράστασης *S*,
- αν η τιμή είναι `true` (αν η συνθήκη ισχύει) τότε α) εκτελείται η εντολή E_t και στη συνέχεια β) η εκτέλεση συνεχίζεται με την εντολή που ακολουθεί την **if**,
- αν η τιμή είναι `false` (αν η συνθήκη δεν ισχύει) τότε αγνοείται η εντολή E_t και η εκτέλεση συνεχίζεται με την εντολή που ακολουθεί την **ifelse**.

Στο Σχ. 5-2 βλέπεις και το λογικό διάγραμμα της **if**.

Να πώς γράφεται το πρόγραμμά μας:

```
#include <iostream>
using namespace std;
int main()
{
   double x, y, maxxy;

   cin >> x >> y;
   if (x > y)  maxxy = x;
   if (x <= y)  maxxy = y;
   cout << " Μεγαλύτερος είναι ο αριθμός: " << maxxy << endl;
}
```

και να πώς εκτελείται. Στη `cin >> x >> y` απαντούμε με:

```
1.6   0.6
```

Το 1.6 γίνεται τιμή της *x* και το 0.6 γίνεται τιμή της *y*. Στη συνέχεια εκτελείται η "`if (x > y) maxxy = x`" και πρώτα απ' όλα υπολογίζεται η συνθήκη "`x > y`" ή "1.6 > 0.6", που είναι `true`. Ύστερα από αυτό εκτελείται η εντολή "`maxxy = x`" και στη συνέχεια η εντολή που ακολουθεί την **if**, δηλαδή η "`if (x <= y) maxxy = y`". Και εδώ υπολογίζεται η συνθήκη "`x <= y`" ή 1.6 <= 0.6 που είναι `false`· έτσι η "`maxxy = y`" *δεν* εκτελείται και πηγαίνουμε στην "`cout << ...`" που δίνει:

```
 Μεγαλύτερος είναι ο αριθμός: 1.6
```

Η "`maxxy = y`" είναι σαν να μην υπάρχει.

Αν στη "`cin >> …`" απαντήσουμε με:

Πλαίσιο 5.1

Η Εντολή if

"`if`", "`(`", *συνθήκη*, "`)`", *εντολή*

Συστατικά:

- το `if` είναι λεξικό σύμβολο,
- η *συνθήκη* είναι μια λογική παράσταση,
- η *εντολή* είναι μια οποιαδήποτε εντολή της C++.

Εκτέλεση της εντολής **if**:

Υπολογίζεται πρώτα η λογική τιμή της συνθήκης.

Αν η τιμή αυτή είναι `true` (αληθής), τότε

εκτελείται η εντολή που ακολουθεί τη συνθήκη

Αν η τιμή είναι `false` η εντολή που ακολουθεί τη συνθήκη αγνοείται.

Η Εντολή ifelse

"`if`", "`(`", *συνθήκη*, "`)`", *εντολή*-1, "`else`", *εντολή*-2

Συστατικά:

- τα `if` και `else` είναι λεξικά σύμβολα,
- η *συνθήκη* είναι μια λογική παράσταση,
- η *εντολή*-1 (περιοχή του if) είναι μια οποιαδήποτε εντολή της C++.
- η *εντολή*-2 (περιοχή του else) είναι μια οποιαδήποτε εντολή της C++.

Εκτέλεση της εντολής **ifelse**:

Υπολογίζεται πρώτα η λογική τιμή της συνθήκης.

Αν η τιμή αυτή είναι `true` (αληθής), τότε

εκτελείται η *εντολή*-1 που βρίσκεται μεταξύ της συνθήκης και του **else**.

Η *εντολή*-2 αγνοείται.

αλλιώς (αν η τιμή είναι `false`)

εκτελείται η *εντολή*-2 που βρίσκεται μετά το **else**.

Η *εντολή*-1 αγνοείται.

1.6 2.6

η "`x > y`" (1.6 > 2.6) θα πάρει τιμή `false` και η "`maxxy = x`" δεν θα εκτελεσθεί. Στην επόμενη **if** η συνθήκη "`x <= y`" ή 1.6 <= 2.6 είναι `true`· στην περίπτωση αυτή θα εκτελεσθεί η εντολή "`maxxy = y`" και στη συνέχεια η "`cout << …`" που δίνει:

```
Μεγαλύτερος είναι ο αριθμός: 2.6
```

Πριν προχωρήσουμε ας δούμε έναν ακόμη τρόπο για να λύσουμε το πρόβλημα του μεγίστου· ο τρόπος αυτός θα μας είναι πολύ χρήσιμος στη συνέχεια.

```
#include <iostream>
using namespace std;
int main()
{
   double x, y, maxxy;

   cin >> x >> y;
   maxxy = y;
   if (x > maxxy)  maxxy = x;
   cout << " Μεγαλύτερος είναι ο αριθμός: " << maxxy << endl;
}
```

Εδώ *θεωρούμε αυθαιρέτως* ότι ο μεγαλύτερος αριθμός είναι ο δεύτερος (δηλ. ο *y*). Στην επόμενη εντολή όμως ελέγχουμε μήπως έχουμε κάνει λάθος (`x > maxxy`)· στην περίπτωση αυτή κάνουμε τη διόρθωση: `maxxy = x`.

Από τους τρεις τρόπους, ο πρώτος είναι ο πιο οικονομικός. Ας δούμε γιατί.

Στο πρώτο πρόγραμμα θα γίνει μια σύγκριση –η `x > y`– και αναλόγως θα εκτελεσθεί μια μόνο από τις εντολές: "`maxxy = x`" ή "`maxxy = y`". Δηλαδή θα έχουμε 1 σύγκριση και 1 εκχώρηση.

Στο δεύτερο θα εκτελεστούν και οι δυο συγκρίσεις (`x > y`, `x <= y`), μια και θα εκτελεσθούν και οι δυο εντολές **if**. Επειδή όμως οι δυο συνθήκες είναι αντίθετες, η μια θα είναι `true` και η άλλη `false`. Έτσι, από τις εντολές "`maxxy = x`" και "`maxxy = y`" θα εκτελεστεί μόνο η μια. Έχουμε λοιπόν: 2 συγκρίσεις και 1 εκχώρηση.

Στην τρίτη περίπτωση η εκχώρηση "`maxxy = y`" θα γίνει οπωσδήποτε. Το ίδιο και η σύγκριση "`x > maxxy`". Η εκχώρηση "`maxxy = x`" θα εκτελεσθεί μόνο στην περίπτωση που $x > y$. Αν δεχτούμε ότι η πιθανότητα για κάτι τέτοιο είναι 50%, το πρόγραμμα αυτό θα εκτελεί 1 σύγκριση και 1.5 εκχώρηση κατά μέσο όρο.

5.3 Φωλιασμένες if – Πολλαπλές Επιλογές

Στις θέσεις των εσωτερικών εντολών της **ifelse** (ή της **if**) μπορεί να δοθεί, όπως ειπώθηκε, οποιαδήποτε εντολή της C++. Επομένως μπορεί να δοθεί μια άλλη εντολή **if**, όπως δείχνει το παρακάτω παράδειγμα:

```
if ( S1 )
   if ( S2 ) εντολή-1
        else εντολή-2
```

Εδώ μπορεί να αναρωτηθείς: που κολλάει το `else`; Στη C++ ισχύει η πρόσθετη σύμβαση που λέει ότι το παραπάνω γενικό σχήμα της **if** (μέσα στην **if**) είναι ισοδύναμο με το σχήμα:

```
if ( S1 )
{
   if ( S2 )  εντολή-1
        else  εντολή-2
}
```

Δηλαδή, η δεύτερη **if** είναι η περιοχή του **if** για την `if (S1)...`. Το `else` είναι της `if (S2)...` . Για να εκτελεσθεί η *εντολή*-1 θα πρέπει να είναι αληθείς και η *S*1 και η *S*2 συγχρόνως. Για να εκτελεσθεί η *εντολή*-2 θα πρέπει να είναι αληθής η *S*1 και να είναι ψευδής η *S*2.

Μπορούμε δηλαδή να πούμε γενικότερα ότι το λεξικό σύμβολο `else` (και η εντολή που το ακολουθεί) αναφέρεται πάντα στο αμέσως προηγούμενο `if`. Αν όμως θέλουμε η *εντολή*-2 να εκτελεστεί σύμφωνα με την τιμή λογικής παράστασης *S*1 μόνον, θα πρέπει να δώσουμε υποχρεωτικώς την παρακάτω μορφή:

```
if ( S1 )  {  if ( S2 )  εντολή-1  }
     else  εντολή-2
```

Στην περίπτωση που μια εντολή **if** εμφανίζεται μέσα σε μιά άλλη **if**, λέγεται **φωλιασμένη** (nested) εντολή **if**.

Με φωλιασμένες **if** λύνουμε προβλήματα στα οποία έχουμε να επιλέξουμε μεταξύ εντολών που είναι περισσότερες από δύο. Στην περίπτωση αυτή βάζουμε δύο ή περίσσότερες **ifelse** τη μία μέσα στην άλλη· συνήθως βάζουμε την επόμενη **ifelse** στην περιοχή **else** της προηγούμενης. Ας ξεκινήσουμε με το

Παράδειγμα 1

Η ΔΕΗ χρεώνει την κατανάλωση ημερήσιου ρεύματος ώς εξής[1]:

- Για τις πρώτες 800 *kWh* (0 .. 800]: 0.088 €/*kWh*
- για τις επόμενες 400 *kWh* (800 .. 1200]: 0.112 €/*kWh*

[1] Η χρέωση γίνεται έτσι, με κλιμακούμενη αύξηση. Για τις τιμές... μην τα συζητάς. Ας πούμε ότι είναι έτσι.

- για τις επόμενες 500 *kWh* (1200 .. 1700]: 0.136 €/*kWh*
- για τις υπόλοιπες *kWh* (1700 .. +∞): 0.28 €/*kWh*

Να γραφεί πρόγραμμα που θα διαβάζει την κατανάλωση σε *kWh* και θα βγάζει το κόστος της ενέργειας που καταναλώθηκε[2].

Για να δούμε τι μας λέει ο παραπάνω πίνακας:

```
αν κατανάλωση <= 800 τότε
  κόστος = κατανάλωση*0.088
αλλιώς (είναι πάνω από 800)
  αν 800 < κατανάλωση <= 1200 τότε
    κόστος = 800*0.088 + (κατανάλωση - 800)*0.112
  αλλιώς (είναι πάνω από 1200)
    αν 1200 < κατανάλωση <= 1700 τότε
      κόστος = 800*0.088 + 400*0.112 + (κατανάλωση - 1200)*0.136
    αλλιώς (είναι πάνω από 1700)
      κόστος = 800*0.088 + 400*0.112 + 500*0.136 +
                    (κατανάλωση - 1700)* 0.28
```

Ας μεταφράσουμε σε **ifelse** την πρώτη σύγκριση:

```
if ( consumption <= 800 )
  cost = consumption*0.088
else
   ...
```

Η περιοχή του **else** θα εκτελεσθεί αν και μόνον αν η "`consumption <= 800`" έχει τιμή `false` ή αλλιώς "`consumption > 800`". Αν λοιπόν γράψουμε την επόμενη **ifelse** ως εξής:

```
if ( 800 < consumption && consumption <= 1200 ) ...
```

η πρώτη σύγκριση (και η λογική πράξη "`&&`") είναι άχρηστη, αφού στη θέση που το βάλαμε έχει σίγουρα τιμή `true`. Στο παρακάτω πρόγραμμα μπορείς να δεις πώς θα γραφούν οι πολλαπλές επιλογές μας.

```
#include <iostream>
using namespace std;
int main()
{
   double  consumption;    // Κατανάλωση σε kWh
   double  cost;

   cout << " Δώσε την κατανάλωση σε kWh: ";  cin >> consumption;
   if (consumption <= 800)
      cost = consumption * 0.088;
   else if (consumption <= (800 + 400))
      cost = 800*0.088 + (consumption - 800)*0.112;
   else if (consumption <= (800 + 400 + 500))
      cost = 800*0.088 + 400*0.112 + (consumption - 1200)*0.136;
   else
      cost = 800*0.088 + 400*0.112 + 500*0.136
             + (consumption - 1700)*0.28;
   cout << cost << endl;
}
```

Γενικώς μπορούμε να γράψουμε:

```
if (S1)  E1
else if (S2)  E2
else if (S3)  E3
   :
else if (Sn)  En
else En+1
```

που θα εκτελεσθεί ως εξής:

[2] Ούτε νυκτερινό, ούτε πάγιο, ούτε ΦΠΑ, ούτε ΕΡΤ, ούτε τίποτε άλλο. Μόνον ημερήσιο!

αν η S_1 έχει τιμή `true` τότε

θα εκτελεσθεί η E_1 και θα αγνοηθούν οι E_2, E_3,... E_n,

αλλιώς (S_1 έχει τιμή `false`) αν η S_2 έχει τιμή `true` τότε

θα εκτελεσθεί η E_2 και θα αγνοηθούν οι E_1, E_3,... E_n,

αλλιώς (S_1 και S_2 έχουν τιμή `false`) αν η S_3 έχει τιμή `true` τότε

θα εκτελεσθεί η E_3 και θα αγνοηθούν οι E_1, E_2,... E_n,

. . .

αλλιώς (S_1, S_2,... S_{n-1} έχουν τιμή `false`) αν η S_n έχει τιμή `true` τότε

θα εκτελεσθεί η E_n και θα αγνοηθούν οι E_1, E_2,... $E_{n\text{-}1}$,

αλλιώς (S_1, S_2,... S_n έχουν τιμή `false`)

θα εκτελεσθεί η E_{n+1} και θα αγνοηθούν οι E_1, E_2,... E_n.

Αλλά προσοχή! Η σειρά που θα βάλεις τις συνθήκες των **if** είναι ουσιαστική. Οι εντολές της k-οστής **if** θα εκτελεστούν αν ισχύει η

$$(\texttt{!}\ S_1)\ \texttt{\&\&}\ \ldots\ \texttt{\&\&}\ (\texttt{!}\ S_{k-1})\ \texttt{\&\&}\ S_k$$

ή αλλιώς:

$$(\texttt{!}\ (S_1\ \texttt{||}\ \ldots\ \texttt{||}\ S_{k-1}))\ \texttt{\&\&}\ S_k$$

Καμιά φορά, αν γράφεις απρόσεκτα, αυτή μπορεί να είναι ισοδύναμη με `false`! Για να δεις ένα (χονδροειδές) παράδειγμα, στο τελευταίο πρόγραμμα βάλε:

```
if ( consumption <= (800 + 400 + 500) )
   cost = 800*0.088 + 400*0.112 + (consumption - 1200)*0.136;
else if ( consumption <= (800 + 400) )
   cost = 800*0.088 + (consumption - 800)*0.112;
else if ( consumption <= 800 )
   cost = consumption * 0.088;
else
  :
```

Για να εκτελεσθεί η

```
   cost = 800*0.088 + (consumption - 800)*0.112;
```

θα πρέπει να έχουμε: "`!(consumption <= 1700) && consumption <= 1200`". Η τιμή αυτής της παράστασης είναι `false`, όποια τιμή και αν έχει η *consumption*. Παρομοίως, `false` είναι πάντοτε και η "`!(consumption <= 1700) && !(consumption <= 1200) && consumption <= 100`" και έτσι αποκλείεται να εκτελεσθεί η: "`cost = consumption*0.088`".

Μερικοί προτιμούν να λύνουν αυτό το πρόβλημα με πολλές ανεξάρτητες **if**. Π.χ. για το παράδειγμά μας θα γράψουν:

```
if ( consumption <= 800 )
   cost = consumption * 0.088;
if ( 800 < consumption && consumption <= 1200 )
   cost = 800*0.088 + (consumption - 800)*0.112;
if ( 1200 < consumption && consumption <= 1700 )
   cost = 800*0.088 + 400*0.112 + (consumption - 1200)*0.136;
if ( consumption < 1700 )
   cost = 800*0.088 + 400*0.112 + 500*0.136
          + (consumption - 1700)*0.28;
```

Φυσικά, το πρόγραμμα είναι μια χαρά. Ποιο είναι το πρόβλημά της; Κάνει πολλές πράξεις χωρίς λόγο.[3] Π.χ. αν του δώσεις κατανάλωση 50 τότε θα υπολογίσει τον λογαριασμό από την πρώτη **if**, αλλά συνέχεια θα υπολογίσει τις συνθήκες όλων των υπολοίπων **if** για να τις βγάλει `false` φυσικά.

[3] Χωρίς λόγο; Πρόσεξε ότι είναι σαφώς πιο σίγουρη από τις άλλες μορφές.

Παράδειγμα 2

Θέλουμε να γράψουμε ένα πρόγραμμα που θα προσομοιώνει τη λειτουργία ενός απλού υπολογιστή τσέπης (pocket calculator). Θα διαβάζει μια γραμμή όπου στην *πρώτη* θέση θα δίνεται το σύμβολο της πράξης (`+`, `-`, `*`, `/`) και στη συνέχεια δυο πραγματικοί αριθμοί με τους οποίους θα γίνει η πράξη.

Θα αποθηκεύουμε το σύμβολο της πράξης σε μια μεταβλητή τύπου `unsigned char` με το όνομα *oper*. Τις δυο πραγματικές τιμές θα τις αποθηκεύουμε σε δυο μεταβλητές *x*1, *x*2, τύπου `double`.

Αφού διαβάσαμε μια γραμμή θα πρέπει να ελέγξουμε την *oper*:

```
αν oper == '+' τότε
  Δώσε το άθροισμα
αλλιώς, η oper μπορεί να είναι '-', '*', '/'
  αν oper == '-' τότε
    Δώσε τη διαφορά
  αλλιώς, η oper μπορεί να είναι '*', '/'
    αν oper == '*' τότε
      Δώσε το γινόμενο
    αλλιώς, η oper μπορεί να είναι '/'
      Δώσε το πηλίκο
```

Μεταφράζοντας τα παραπάνω σε C++ θα έχουμε δυο **ifelse** που η κάθε μια τους έχει σαν περιοχή του else μια άλλη **ifelse**. Αλλά μάλλον δεν τελειώσαμε: Θα πρέπει να προβλέψουμε την πιθανότητα λάθους στην πράξη. Δεν είναι καθόλου σίγουρο ότι αν η πράξη δεν είναι '`+`', '`-`', '`*`' θα είναι '`/`'. Μπορεί ο χρήστης να δώσει κατά λάθος '`@`', ας πούμε. Θα πρέπει λοιπόν να ελέγχουμε την πράξη και για τη διαίρεση. Τέλος, δεν θα πρέπει να προσπαθήσουμε να υπολογίσουμε το πηλίκο αν ο διαιρέτης είναι "`0`" (μηδέν). Να το πρόγραμμά μας τελικώς:

```
#include <iostream>
using namespace std;
int main()
{
   double x1, x2;          // Τα ορίσματα της πράξης
   unsigned char oper;     // Το σύμβολο της πράξης

   cin >> oper >> x1 >> x2;
   if (oper == '+')        cout << (x1 + x2) << endl;
   else if (oper == '-')   cout << (x1 - x2) << endl;
   else if (oper == '*')   cout << (x1 * x2) << endl;
   else if (oper == '/')
      if (x2 != 0)          cout << (x1 / x2) << endl;
      else
         cout << " Δεν γίνεται" << endl;
   else
      cout << " Η πράξη (+,-,*,/) στην πρώτη θέση" << endl;
}
```

5.4 * Η Λογική των Εντολών Επιλογής

Ας δούμε τώρα τα εργαλεία που μας χρειάζονται για να κάνουμε αποδείξεις ορθότητας προγραμμάτων με εντολές **ifelse** και **if**.

Υποθέτουμε ότι έχουμε την εξής περίπτωση:

```
// P
   if (S)  Et else Ef
```

Αν ισχύει η *S* τότε εκτελείται η εντολή *Et* με προϋπόθεση *P* `&&` *S* και έστω ότι καταλήγουμε στην συνθήκη *Qt*. Αν δεν ισχύει η *S* –δηλαδή ισχύει η `!`*S*– τότε εκτελείται η εντολή *Ef* με

προϋπόθεση *P* **&&** (**!***S*) και έστω ότι καταλήγουμε στην *Qf*. Άρα, η συνθήκη που θα καταλήξουμε μετά την εκτέλεση της **ifelse** θα είναι η: *Qt* **||** *Qf*. Βλέπουμε λοιπόν ότι:

- ***η ifelse είναι ένας τρόπος για να πάρουμε συνθήκες που να περιέχουν*** **||**.

Επειδή όμως, όπως ξέρουμε, δεν γράφουμε εντολές στην τύχη για να δούμε πού θα καταλήξουμε, αλλά τις γράφουμε για να καταλήξουμε σε κάποια συγκεκριμένη συνθήκη, ας δούμε το πρόβλημά μας κάπως διαφορετικά:

```
// P
   if (S)  Et else Ef                                    (1)
// Q
```

Δηλαδή: είμαστε στην κατάσταση *P* και γράφουμε την `if (S) Et else Ef` με στόχο να καταλήξουμε στην *Q*. Τι γίνεται τώρα;

Αν ισχύει η *S* τότε εκτελείται η εντολή *Et* με προϋπόθεση *P* **&&** *S*. Αν το πρόγραμμά μας είναι σωστό θα πρέπει να φτάνουμε στην *Q*. Αν πάλι δεν ισχύει η *S* –δηλαδή ισχύει η **!***S*– τότε εκτελείται η εντολή *Ef* με προϋπόθεση *P* **&&** (**!***S*). Αν το πρόγραμμά μας είναι σωστό θα πρέπει και πάλι να φτάνουμε στην *Q*. Δηλαδή:

- ***Για να αποδείξουμε την ορθότητα της* (1) *πρέπει και αρκεί να αποδείξουμε την ορθότητα των:***

```
      P && S { Et } Q
P && (!S) { Ef } Q
```

Αυτός είναι ο συμπερασματικός κανόνας της **ifelse**, που γράφεται συμβολικώς ως εξής:

$$\frac{P \,\&\&\, S\,\{Et\}\,Q,\; P \,\&\&\, (!S)\,\{Ef\}\,Q}{P\,\{\textbf{if}\ (S)\ Et\ \textbf{else}\ Ef\}\,Q}$$

Παρομοίως ισχύουν για την **if**. Αν έχουμε:

```
// P
   if (S)  Et
```

τότε: αν μεν ισχύει η *S* εκτελείται η εντολή *Et* με προϋπόθεση *P* **&&** *S* και έστω ότι καταλήγουμε στην συνθήκη *Qt*. Αν δεν ισχύει η *S* –δηλαδή ισχύει η **!***S*– τότε *δεν εκτελείται* η εντολή *Et* και παραμένει η *P* **&&** (**!***S*). Άρα, η συνθήκη που θα καταλήξουμε μετά την εκτέλεση της **if** θα είναι η: *Qt* **||** (*P* **&&** (**!***S*)).

Ας δούμε και τον συμπερασματικό κανόνα της **if**. Ας πούμε ότι έχουμε:

```
// P
   if (S)  Et                                            (2)
// Q
```

Αν ισχύει η *S* τότε εκτελείται η εντολή *Et* με προϋπόθεση *P* **&&** *S*. Αν το πρόγραμμά μας είναι σωστό θα πρέπει να φτάνουμε στην *Q*. Αν πάλι δεν ισχύει η *S* (ισχύει η **!***S*) τότε θα πρέπει να φτάνουμε στην *Q* χωρίς να εκτελεστεί οποιαδήποτε εντολή. Αφού λοιπόν ισχύει η *P* **&&** (**!***S*) θα πρέπει από αυτήν να συνάγεται η *Q*. Δηλαδή:

- ***Για να αποδείξουμε την ορθότητα της* (2) *πρέπει και αρκεί να αποδείξουμε την ορθότητα των:***

```
P && S { Et } Q
(P && (!S)) ⇒ Q
```

Αυτός είναι ο συμπερασματικός κανόνας της **if**. Συμβολικώς:

$$\frac{P \,\&\&\, S\,\{Et\}\,Q,\; (P \,\&\&\, (!S)) \Rightarrow Q}{P\,\{\textbf{if}\ (S)\ Et\}\,Q}$$

Παράδειγμα

Για παράδειγμα χρήσης του κανόνα της **ifelse** ας αποδείξουμε ότι στο πρώτο πρόγραμμα της §5.2 βρίσκουμε σωστά το μέγιστο. Φυσικά, θα πρέπει να ξεκινήσουμε με τις προδιαγραφές.

Ξεκινούμε με την απαίτηση· θέλουμε να έχουμε τη *maxxy* μεγαλύτερη (ή ίση) και από τη *x* και από τη *y*: (*maxxy* ≥ *x*) **&&** (*maxxy* ≥ *y*). Αλλά η τιμή της *maxxy* θα είναι ίση είτε με *x* είτε με *y*: (*maxxy* **==** *x*) **||** (*maxxy* **==** *y*). Να λοιπόν η απαίτηση:

((*maxxy* ≥ *x*) **&&** (*maxxy* ≥ *y*)) **&&** ((*maxxy* **==** *x*) **||** (*maxxy* **==** *y*))

Τι προϋπόθεση θα έχουμε; Τίποτε ιδιαίτερο. Θα πρέπει να μπορούμε να επεξεργαστούμε *οποιεσδήποτε* τιμές και αν πάρουμε –στις *x*, *y*– από τη "`cin >> x >> y`". Δηλαδή, όπως είπαμε στην §3.7, θα έχουμε για προϋπόθεση:

`true`

Να λοιπόν τι πρέπει να αποδείξουμε:

```
// true
   if (x > y)  maxxy = x;
         else  maxxy = y;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

Σύμφωνα με τον κανόνα της **ifelse**, αρκεί να αποδείξουμε ότι:

```
// true && (x > y)
   maxxy = x;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

και

```
// true && (!(x > y))
   maxxy = y;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

Αυτές όμως αποδεικνύονται πολύ εύκολα.

1: Για να ισχύει η ((*maxxy* ≥ *x*) **&&** (*maxxy* ≥ *y*)) **&&** ((*maxxy* **==** *x*) **||** (*maxxy* **==** *y*)) μετά τη "`maxxy = x`" θα πρέπει πριν από αυτήν να ισχύει η: ((***x*** ≥ *x*) **&&** (***x*** ≥ *y*)) **&&** ((***x*** **==** *x*) **||** (***x*** **==** *y*)), που παίρνουμε από την απαίτηση αν βάλουμε όπου *maxxy* το *x*. Αν πάρουμε υπόψη μας ότι

- ότι οι ***x*** ≥ *x* και ***x*** **==** *x* έχουν τιμή `true` για οποιαδήποτε τιμή της *x*,
- ότι (`true` **&&** A) ≡ A,
- ότι (`true` **||** A) ≡ `true`,

αυτή απλουστεύεται σε: ***x*** ≥ *y*. Αυτή όμως συνάγεται από την προϋπόθεση *x* > *y*.

2: Για να ισχύει η ((*maxxy* ≥ *x*) **&&** (*maxxy* ≥ *y*)) **&&** ((*maxxy* **==** *x*) **||** (*maxxy* **==** *y*)) μετά τη "`maxxy = y`" θα πρέπει πριν από αυτήν να ισχύει η: ((***y*** ≥ *x*) **&&** (***y*** ≥ *y*)) **&&** ((***y*** **==** *x*) **||** (***y*** **==** *y*)). Παρομοίως και εδώ, αν παρουμε υπόψη μας ότι οι ***y*** ≥ *y* και ***y*** **==** *y* έχουν τιμή `true` για οποιαδήποτε τιμή της *y*, καθώς και τις ταυτολογίες που είδαμε παραπάνω, αυτή απλουστεύεται σε: ***y*** ≥ *x*. Αυτή όμως συνάγεται από την προϋπόθεση **!**(*x* > *y*) που σημαίνει: *x* ≤ *y*.

5.4.1 * Από το Τέλος προς την Αρχή

Στις αποδείξεις που κάναμε με την εντολή εκχώρησης, χρησιμοποιούσαμε το αξίωμα της εκχώρησης για βρούμε τη συνθήκη που θα πρέπει να ισχύει *πριν* από μια εντολή ώστε μετά από αυτή να έχουμε μια συγκεκριμένη απαίτηση. Εδώ θα δούμε πώς θα κάνουμε τα ίδια με τις εντολές **if**.

Το πρόβλημα που έχουμε να λύσουμε είναι το εξής: Ποια θα πρέπει να είναι η *P* ώστε να έχουμε:

```
   // P
      if (S)  Et else Ef
   // Q
```

Ας ονομάσουμε *Pt* και *Pf* τις συνθήκες για τις οποίες έχουμε:

```
// Pt                  // Pf
   Et        και          Ef
// Q                   // Q
```

Αν βρούμε τις *Pt* και *Pf* τότε η *P* είναι (δες και την Άσκ. 5-3):

$$(Pt \;\&\&\; S) \;||\; (Pf \;\&\&\; (!S))$$

Παράδειγμα 1

Έστω ότι έχουμε να αποδείξουμε:

```
// (-1 ≤ x < 0) || (1 ≤ x < 6)
   if (x < 0)  y = x;
         else  y = x - 2;
// -1 ≤ y ≤ 4
```

Για να αποδείξουμε αυτό που ζητείται θα πρέπει:

1. να βρούμε, όπως είπαμε παραπάνω, την *P* ώστε:

```
// P
   if (x < 0)  y = x;
         else  y = x - 2;
// -1 ≤ y ≤ 4
```

2. να αποδείξουμε ότι η *P* συνάγεται από την προϋπόθεση (σ.κ. (*E*2), §0.4), δηλ.:

$((-1 \le x \le 0)$ `||` $(1 \le x \le 6)) \Rightarrow P$

Ξεκινούμε από το:

1. Οι αντιστοιχίες με το μοντέλο που δώσαμε πιο πάνω είναι:

- *Et* είναι η: `y = x;`
- *Ef* είναι η: `y = x - 2;`
- *Q* είναι η: $-1 \le y \le 4$

Θέλουμε λοιπόν να βρούμε την *Pt* και την *Pf* ώστε:

```
// Pt                        // Pf
   y = x;                       y = x - 2;
// -1 ≤ y ≤ 4                // -1 ≤ y ≤ 4
```

Για να έχουμε $-1 \le y \le 4$ μετά την `y = x` θα πρέπει να έχουμε πριν από αυτήν $-1 \le x \le 4$. Αυτή είναι η *Pt*.

Παρομοίως, για να έχουμε $-1 \le y \le 4$ μετά την `y = x - 2` θα πρέπει, πριν από αυτήν, να έχουμε $-1 \le x-2 \le 4$ ή αλλιώς $1 \le x \le 6$. Αυτή είναι η *Pf*.

Άρα, πριν από την **ifelse** θα πρέπει να ισχύει η: $(Pt$ `&&` $S)$ `||` $(Pf$ `&&` $(!S))$ ή αλλιώς, αν πάρουμε υπόψη μας ότι *S* είναι η $x < 0$ και `!`*S* η $!(x < 0)$ ή $(x \ge 0)$:

$$((-1 \le x \le 4) \;\&\&\; (x < 0)) \;||\; ((1 \le x \le 6) \;\&\&\; (x \ge 0))$$

ή αλλιώς: $(-1 \le x < 0)$ `||` $(1 \le x \le 6)$

Τώρα ερχόμαστε στο

2. Θα πρέπει να απόδείξουμε ότι η $(-1 \le x < 0)$ `||` $(1 \le x \le 6)$ συνάγεται από την προϋπόθεση $(-1 \le x < 0)$ `||` $(1 \le x < 6)$. Για να ισχύει η προϋπόθεση θα πρέπει να ισχύει είτε η $-1 \le x < 0$ είτε η $1 \le x < 6$ (είτε και οι δύο, αλλά στην περίπτωσή μας αυτό είναι αδύνατο). Αν ισχύει η πρώτη τότε η *P* ισχύει. Αν ισχύει η δεύτερη $(1 \le x < 6)$, τότε ισχύει και η $1 \le x \le 6$, άρα ισχύει και η *P*.

Επομένως το πρόγραμμά μας είναι σωστό.

Στην περίπτωση της **if** έχουμε το εξής πρόβλημα: Ποια θα πρέπει να είναι η *P* ώστε να έχουμε:

```
   // P
      if (S)  Et
   // Q
```

Ας ονομάσουμε, όπως παραπάνω, *Pt* τη συνθήκη για την οποία έχουμε:

```
// Pt
   Et
// Q
```

Αν βρούμε την *Pt*, η *P* είναι (δες και την Άσκ. 5-3):

$$(S \texttt{ \&\& } Pt) \texttt{ || } ((\texttt{! } S) \texttt{ \&\& } Q)$$

Παράδειγμα 2

Θα αποδείξουμε, από το τέλος προς την αρχή, την ορθότητα του τρίτου προγράμματος για τον υπολογισμό του μεγίστου, δηλαδή:

```
// true
   maxxy = y;
   if (x > maxxy)  maxxy = x;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

Θα πρέπει πρώτα να βρούμε τη συνθήκη που θα πρέπει να ισχύει πριν από την **if**, δηλ. την $(S \texttt{ \&\& } Pt) \texttt{ || } ((\texttt{! } S) \texttt{ \&\& } Q)$. Στην περίπτωσή μας:

- S είναι η $x > maxxy$ και $\texttt{!}S$ είναι η $\texttt{!}(x > maxxy)$ ή $x \le maxxy$,
- Q είναι η $((maxxy \ge x) \texttt{ \&\& } (maxxy \ge y)) \texttt{ \&\& } ((maxxy \texttt{ == } x) \texttt{ || } (maxxy \texttt{ == } y))$

Το πρώτο βήμα είναι να υπολογίσουμε την Pt ώστε:

```
// Pt
   maxxy = x;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

Όπως είδαμε και στο παράδ. της §5.5 η Pt είναι: $\boldsymbol{x} \ge y$.

$$((x > maxxy) \texttt{ \&\& } (x \ge y)) \texttt{ ||}$$
$$((x \le maxxy) \texttt{ \&\& } ((maxxy \ge x) \texttt{ \&\& } (maxxy \ge y)) \texttt{ \&\& } ((maxxy \texttt{ == } x) \texttt{ || } (maxxy \texttt{ == } y)))$$

ή, απλούστερα:

$$((x > maxxy) \texttt{ \&\& } (x \ge y)) \texttt{ ||}$$
$$(((maxxy \ge x) \texttt{ \&\& } (maxxy \ge y)) \texttt{ \&\& } ((maxxy \texttt{ == } x) \texttt{ || } (maxxy \texttt{ == } y)))$$

Έχουμε λοιπόν:

```
   maxxy = y;
// ((x>maxxy) && (x≥y)) ||
// ((maxxy≥x) && (maxxy≥y) && ((maxxy==x)||(maxxy==y)))
   if (x > maxxy)  maxxy = x;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

Αντικαθιστώντας στη συνθήκη που βρήκαμε την y στη θέση της $maxxy$ παίρνουμε τη συνθήκη που θα πρέπει να ισχύει πριν από τη `maxxy = y`:

$$((x > y) \texttt{ \&\& } (x \ge y)) \texttt{ || } (((y \ge x) \texttt{ \&\& } (y \ge y)) \texttt{ \&\& } ((y \texttt{ == } x) \texttt{ || } (y \texttt{ == } y)))$$

Αυτή όμως απλουστεύεται:

- η $(x > y) \texttt{ \&\& } (x \ge y)$ ισοδυναμεί με $(x > y)$,
- οι $y \ge y$ και $y \texttt{ == } y$ έχουν τιμή `true`

και η συνθήκη μας γίνεται:

$$(x > y) \texttt{ || } (y \ge x)$$

Αυτή όμως είναι η `true` που είναι και η προϋπόθεσή μας.

5.5 Εξασφάλιση Προϋποθέσεων

Τώρα έχουμε στα χέρια μας και άλλα εργαλεία –εκτός από την *assert*– για να ελέγχουμε τις προϋποθέσεις για την εκτέλεση των προγραμμάτων μας.

Παράδειγμα 1

Οι υπολογισμοί του προγράμματος για το πρόβλημα της ελεύθερης πτώσης απαιτούν να διασφαλίσθεί η συνθήκη $h \ge 0$. Αυτό μπορεί να γίνει πολύ εύκολα:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
```

```
   const double g( 9.81 ); // m/sec², η επιτάχυνση της βαρύτητας
   double h,   // m, αρχικό ύψος
          tP,  // sec, χρόνος πτώσης
          vP;  // m/sec, ταχύτητα τη στιγμή πρόσκρουσης

// Διάβασε το h
   cout << " Δώσε μου το αρχικό ύψος σε m: ";  cin >> h;
   if ( h >= 0 )
   { // (g == 9.81) && (0 ≤ h ≤ DBL_MAX)
   // Υπολόγισε τα tP, vP
      tP = sqrt( (2/g)*h );
      vP = -g*tP;
   // (tP ≈ √(2h/g)) && (vP ≈ -√(2hg))
   // Τύπωσε τα tP, vP
      cout << " Αρχικό ύψος = " << h << " m" << endl;
      cout << " Χρόνος Πτώσης = " << tP << " sec" << endl;
      cout << " Ταχύτητα τη Στιγμή της Πρόσκρουσης = "
           << vP << " m/sec" << endl;
   }
   else
   // false
      cout << " το ύψος δεν μπορεί να είναι αρνητικό" << endl;
}
```

Εκείνο το «**false**» στο σχόλιο, μετά το **else**, τι σημαίνει; Ότι δεν υπάρχουν τιμές των *tP* και *vP* που να πληρούν την απαίτηση όταν $h < 0$.

Σύγκρινέ αυτό το πρόγραμμα με αυτό της §4.4 και διάλεξε αυτό που προτιμάς: *assert* ή **if**. Θα επανέλθουμε...

Παράδειγμα 2

Να ξαναγυρίσουμε στο παράδειγμα του τριωνύμου, για το οποίο, την τελευταία φορά, ανακαλύψαμε ότι ξεχάσαμε τον έλεγχο της *a*.

Πριν κάνουμε οποιονδήποτε άλλον υπολογισμό θα πρέπει να ελέγξουμε αν: $a \neq 0$ και $\Delta \geq 0$. Αυτή είναι η προϋπόθεσή μας:

Προϋπόθεση: $(a \neq 0)$ **&&** $(b^2 - 4ac \geq 0)$

Απαίτηση: $(x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a})$ **&&** $(x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a})$

Το παρακάτω πρόγραμμα, που είναι μια τροποποιημένη μορφή του προγράμματος της §5.2, πριν από όλα ελέγχει αν ισχύει η προϋπόθεση.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   double  a, b, c;    // οι συντελεστές της εξίσωσης
   double  x1, x2;     // οι ρίζες της εξίσωσης
   double  delta;      // η διακρίνουσα

   cout << "Δώσε τρεις πραγματικούς αριθμούς" << endl;
   cin >> a >> b >> c;
   cout << " a = " << a << "   b = " << b
        << "   c = " << c << endl;
   delta = b*b - 4*a*c;
   if ( a != 0  &&  delta >= 0 )
   {
      x1 = (-b + sqrt(delta))/(2*a);
      x2 = (-b - sqrt(delta))/(2*a);
      cout << " Λύση1 = " << x1 << "   Λύση2 = " << x2 << endl;
   }
   else // εκτός προδιαγραφών
   {
      if ( a == 0 )
```

```
         cout << " Η εξίσωση είναι 1ου βαθμού" << endl;
      else
         cout << " Δεν υπάρχουν Πραγματικές Λύσεις" << endl;
   }
   cout << " ΤΕΛΟΣ" << endl;
}
```

Τώρα, όπως βλέπεις, τα *x1* και *x2* θα υπολογιστούν μόνο στην περίπτωση που $a \neq 0$ **και** $delta \geq 0$.

Παρατηρήσεις ▶

1. Να υπενθυμίσουμε ότι αυτή η σύγκριση, "`a != 0`", δεν έχει και τόσο νόημα, αφού η *a* είναι τύπου `double`. Αν η τιμή της *a* δεν έρχεται από το πληκτρολόγιο, αλλά από ενδιάμεσους υπολογισμούς, μπορεί να έχει μια πολύ μικρή (απόλυτη) τιμή, που να μην είναι μηδέν, αλλά να δημιουργεί πρόβλημα.
2. Όπως ξέρεις, μπορούμε να συνεχίσουμε την αναζήτηση λύσης ακόμη και στην περίπτωση που *a* `==` 0. Δες την Άσκ. 5-7.
3. Το να γράφουμε τις εντολές εκχώρησης για τις *x1* και *x2* όπως ξέρουμε από τα μαθηματικά μπορεί να φαίνεται όμορφο αλλά δεν είναι και το καλύτερο για το πρόγραμμά μας. Όπως θα μάθουμε αργότερα, η αφαίρεση είναι μια «κακή» πράξη για τους τύπους κινητής υποδιαστολής διότι μπορεί να προκαλέσει *απώλεια σημαντικών ψηφίων*. Θα δούμε λοιπόν έναν άλλον τρόπο που αποφεύγει την αφαίρεση:

```
      if ( b >= 0 )
         x1 = (-b - sqrt(delta))/(2*a);
      else // b < 0
         x1 = (-b + sqrt(delta))/(2*a);
      x2 = c/(a*x1);
```

Αν, ας πούμε, τα *a*, *b*, *c* έχουν τιμές 1, -1, -30 αντιστοίχως, θα εκτελεσθεί η

```
         x1 = (-b + sqrt(delta))/(2*a);
```

όπου έχουμε τον υπολογισμό $-(-1)+\sqrt{(-1)^2 - 4\cdot1\cdot(-30)} = 1 + \sqrt{121} = 1 + 11 = 12$ (*x1* `==` 6). Η άλλη λύση υπολογίζεται (*x2* `==` -5) από τη γνωστή ιδιότητα: *x1*`*`*x2* `==` *c*`/`*a*.

Αν όμως τα *a*, *b*, *c* έχουν τιμές 1, 11, 30, θα εκτελεσθεί η

```
         x1 = (-b - sqrt(delta))/(2*a);
```

όπου έχουμε τον υπολογισμό $-11 - \sqrt{11^2 - 4\cdot1\cdot30} = -11 - \sqrt{1} = -11 - 1 = -(11+1) = -12$ (*x1* `==` -6). Η άλλη λύση υπολογίζεται ως: *x2* `==` (30`/`1)/(-6) `==` -5.

Και στις δύο περιπτώσεις, οι δύο όροι του αριθμητή έχουν το ίδιο πρόσημο και έτσι «αποφεύγουμε την αφαίρεση».◀

Παράδειγμα 3

Ας έρθουμε στο παράδ. 2 της §5.4 (υπολογιστής τσέπης). Όπως λέγαμε, έχουμε πρόβλημα όταν πάρουμε σύμβολο πράξης που δεν είναι '`+`', '`-`', '`*`', '`/`'. Ακόμη, έχουμε πρόβλημα όταν η πράξη είναι '`/`' και ο διαιρέτης είναι "`0`" (μηδέν). Να λοιπόν οι προδιαγραφές μας:

Προϋπόθεση:

(*oper* `==` '`+`') `||` (*oper* `==` '`-`') `||` (*oper* `==` '`*`') `||`
((*oper* `==` '`/`') `&&` (*x*2 `!=` 0))

Απαίτηση:

((*oper* `==` '`+`') `&&` (*res* `==` *x*1 + *x*2)) `||` ((*oper* `==` '`-`') `&&` (*res* `==` *x*1 - *x*2)) `||`
((*oper* `==` '`*`') `&&` (*res* `==` *x*1 * *x*2)) `||` ((*oper* `==` '`/`') `&&` (*x*2 `!=` 0) `&&` (*res* `==` *x*1 / *x*2))

και να πώς ελέγχουμε την προϋπόθεση στο πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   double x1, x2;           // Τα ορίσματα της πράξης
   unsigned char oper;      // Το σύμβολο της πράξης
```

```
   double res;              // Το αποτέλεσμα της πράξης

   cin >> oper >> x1 >> x2;
   if (oper == '+' || oper == '-' ||
       oper == '*' || (oper == '/' && x2 != 0))
   {
      if (oper == '+')            res = x1 + x2;
      else if (oper == '-')       res = x1 - x2;
      else if (oper == '*')       res = x1 * x2;
      else /* if (oper == '/') */ res = x1 / x2;
      cout << res << endl;
   }
   else
      cout << " ΛΑΘΟΣ" << endl;
}
```

Φυσικά και εδώ, όπως και στο προηγούμενο παράδειγμα, η σύγκριση `x2 != 0` δεν λέει και πολλά πράγματα.

5.6 Σειρά Εκτέλεσης των Πράξεων

Στο προηγούμενο κεφάλαιο, §4.1, όταν εξετάζαμε τη σειρά εκτέλεσης των πράξεων, αφήσαμε για αργότερα ένα «λεπτό» σημείο· ας το δούμε. Η C++ επιταχύνει τον υπολογισμό λογικών παραστάσεων ως εξής

- Στη λογική πράξη "`&&`" αν το πρώτο όρισμα υπολογιστεί "`false`" τότε το αποτέλεσμα της πράξης υπολογίζεται "`false`" χωρίς να υπολογιστεί το δεύτερο όρισμα.
- Στη λογική πράξη "`||`" αν το πρώτο όρισμα υπολογιστεί "`true`" τότε το αποτέλεσμα της πράξης υπολογίζεται "`true`" χωρίς να υπολογιστεί το δεύτερο όρισμα.

Τι επιπτώσεις μπορεί να έχουν τα παραπάνω στον προγραμματισμό; Ας δούμε ένα

Παράδειγμα

Ας πούμε ότι σε κάποιο πρόγραμμα θέλουμε να εκτελεσθούν οι εντολές E με την προϋπόθεση ότι $\sqrt{x} > y + 2$ και, φυσικά, ότι $x \geq 0$.

Με αυτά που ξέραμε μέχρι τώρα θα γράφαμε:

```
   if ( x >= 0 )
      if ( sqrt(x) > y + 2 )  E
```

Έτσι έχουμε σιγουριά ότι δεν θα γίνει απόπειρα υπολογισμού της τετραγωνικής ρίζας αν δεν έχουμε εξασφαλίσει ότι η x έχει τιμή μη αρνητική.

Αν πάρουμε υπόψη τον γρήγορο υπολογισμό της "`&&`" γράφουμε:

```
   if ( x >= 0 && sqrt(x) > y + 2 )  E
```

Αν η x έχει τιμή αρνητική τότε η "`x >= 0`" υπολογίζεται σε "`false`" και την ίδια τιμή παίρνει και ολόκληρη η συνθήκη *χωρίς να γίνει απόπειρα υπολογισμού της* `sqrt(x)`, οπότε οι E δεν εκτελούνται.

Αυτό το χαρακτηριστικό της C++ ονομάζεται **υπολογισμός παράστασης bool με βραχυκύκλωμα** (short-circuit boolean evaluation)[4] και το αναφέρουμε για να καταλαβαίνεις προγράμματα γραμμένα από άλλους και όχι για να τη χρησιμοποιείς κατ' ανάγκη. Ο πρώτος τρόπος, με τις δύο **if**, είναι πιο σίγουρος και σου δίνει πρόγραμμα πιο ευανάγνωστο.[5]

[4] Η «συγγενείς» προς τη C++ γλώσσες Java και η C# δίνουν διαφορετικούς τελεστές για τις βραχυκυκλωμένες πράξεις από αυτούς που δίνουν για τις συνήθεις.

[5] Φυσικά ο δεύτερος είναι πιο γρήγορος, αλλά άφησέ για αργότερα την ανησυχία για την ταχύτητα.

5.7 Τα ";" και Άλλα Λάθη

Καινούριες εντολές, πιο πολλές δυνατότητες για προγραμματισμό, αλλά και πιο πολλές πιθανότητες για λάθη.

Πρώτα απ' όλα πρόσεξε «το λάθος που θα κάνεις πάντα» (§4.1.1):

♦ ***Η σύγκριση για ισότητα γίνεται με το "==" και όχι με το "=".***

Από το μεταγλωττιστή περνάει εύκολα (με μια προειδοποίηση) και μετά δεν είναι εύκολο να το ανακαλύψεις, αν μάλιστα το πρόγραμμα είναι μεγάλο.

Ας έρθουμε τώρα στο ';'. Μερικοί έχουν τη συνήθεια να βάζουν αυτόν το χαρακτήρα κάθε φορά που τελειώνουν μια γραμμή. Γράφει λοιπόν κάποιος:

```
if ( x > y );
   maxxy = x;
else;
   maxxy = y;
```

Ο μεταγλωττιστής της Borland C++ θα δώσει μια προειδοποίηση και ένα λάθος. Συγκεκριμένα:

- Για την `if (x > y)` θα δώσει προειδοποίηση: «code has no effect» (κώδικας που δεν κάνει κάτι). Όπως είναι γραμμένη η εντολή, ο μεταγλωττιστής καταλαβαίνει ότι πρόκειται για μια **if** (όχι **ifelse**) που ζητάει: αν έχουμε `x > y` να εκτελεσθεί η κενή εντολή! Επομένως, η **if** δεν έχει οποιοδήποτε αποτέλεσμα.
- Για το `else` μας δίνει λάθος «misplaced else» (else σε λάθος θέση). Αφού κατάλαβε ότι έχουμε **if** και όχι **ifelse**, μας λέει ότι το **else** κακώς τοποθετήθηκε εκεί.

Μέχρι εδώ τα πράγματα δεν είναι και τόσο άσχημα: αφού μας βγάζει λάθος το βρίσκουμε, το διορθώνουμε και το ξαναδίνουμε:

```
if ( x > y )
   maxxy = x;
else;
   maxxy = y;
```

Ο μεταγλωττιστής σου το βρήκε εντάξει. Το δοκιμάζεις κιόλας: δίνεις στο *x* την τιμή 4, στο *y* την τιμή 5, σου βγάζει μεγαλύτερο το 5, όλα πάνε καλά. Κάνεις άλλη μια δοκιμή: 5 στο *x*, 4 στο *y*, σου βγάζει μεγαλύτερο το 4. Μα τι γίνεται; Τη μια δουλεύει, την άλλη δεν δουλεύει! Για να δούμε: Την πρώτη φορά η συνθήκη "`x > y`" ή "4 > 5" είναι `false`, άρα θα πρέπει να εκτελεσθεί η περιοχή του **else** που είναι... ποιά; Η *κενή εντολή!* Μόνο αυτή χωράει ανάμεσα στο **else** και το ';' που το ακολουθεί. Στη συνέχεια εκτελείται η εντολή που ακολουθεί την **ifelse**, η "`maxxy = y`" και παίρνεις σωστό αποτέλεσμα, αλλά τελείως συμπτωματικά. Τη δεύτερη φορά, η "`x > y`" παίρνει τιμή `true` και εκτελείται η περιοχή του **if**, "`maxxy = x`". Η *maxxy* παίρνει τη σωστή τιμή (5) και στη συνέχεια εκτελείται η εντολή που ακολουθεί την **ifelse**, "`maxxy = y`", που αλλάζει την τιμή της *maxxy* σε 4.

Πάντως, αν ψάξεις τα μηνύματα που σου στέλνει ο μεταγλωττιστής, θα βρεις μια προειδοποίηση για την εντολή: "`maxxy = x`" (περιοχή του **if**): «'maxxy' is assigned a value that is never used» (στη *maxxy* εκχωρείται μια τιμή που δεν χρησιμοποιείται ποτέ).

Παρόμοιο πρόβλημα θα έχεις και στην περίπτωση που θα βάλεις ';' μετά τη συνθήκη κάποιας **if**:

```
maxxy = y;
if (x > maxxy);
   maxxy = x;
```

Εδώ, περιοχή του **if** είναι η κενή εντολή και η "`maxxy = x`" είναι η επόμενη εντολή.

Συνοψίζοντας μπορούμε να πούμε:

♦ ***Δεν βάζουμε ποτέ*** ';'
  - ***μετά τη συνθήκη***
  - ***μετά το else.***

Τώρα που έμαθες τις εντολές **if** μπορείς να μάθεις κάτι που κάνουν οι πεπειραμένοι προγραμματιστές για να εντοπίσουν λάθη στα προγράμματά τους. Ορίζουν μια σταθερά:

```
const bool debug( true );
```

Στη συνέχεια, αντί για απλές "`cout << ...` ", που τυπώνουν τις τιμές των συνθηκών επαλήθευσης, όπως είδαμε στην §3.2, βάζουν εντολές:

```
if ( debug )  cout << ...
```

Όταν ανακαλύψουν τα λάθη τους και τα διορθώσουν, αλλάζουν την αρχική δήλωση σε:

```
const bool debug( false );
```

Έτσι, βλέπουν την εκτέλεση του προγράμματος χωρίς τα ενδιάμεσα αποτελέσματα. Μόλις παρουσιαστούν τα νέα λάθη, που η πείρα τους λέει ότι πρέπει να τα περιμένουν[6], ξαναλλάζουν την τιμή της *debug* σε `true`.[7]

Αυτά βέβαια στην περίπτωση που δεν διαθέτουν **Βοηθητικά Προγράμματα για Ανεύρεση Λαθών** (debugging[8] utilities, debuggers), που σου δίνουν τη δυνατότητα να παίρνεις ενδιάμεσα αποτελέσματα με πιο απλούς τρόπους.

5.8 Τι (Πρέπει να) Έμαθες

Τώρα πρέπει να ξέρεις να γράφεις προγράμματα που να έχουν περισσότερους από έναν κλάδους εκτέλεσης.

Θα πρέπει να μπορείς να χρησιμοποιείς συνθήκες για να επιλέγεις ποιες εντολές θα εκτελεσθούν και ποιες όχι, με βάση τις τιμές που έχουν οι μεταβλητές του προγράμματός σου κατά τη διάρκεια της εκτέλεσης. Θα πρέπει να μπορείς να «φωλιάζεις» εντολές **if** ή/και **ifelse** τη μια μέσα στην άλλη.

Θα πρέπει να ξέρεις να ελέγχεις αν τα δεδομένα που διαβάζεις συμμορφώνονται με τις προδιαγραφές.

Τέλος, αν μελετάς και τις παραγράφους με τα αστεράκια, θα πρέπει να μπορείς να αποδείξεις την ορθότητα προγραμμάτων με εντολές **if** ή/και **ifelse**. Θα ξέρεις επίσης και πώς διαπλέκονται οι συνθήκες επαλήθευσης με αυτές που χρησιμοποιούμε στις εντολές επιλογής.

Ασκήσεις

Α Ομάδα

5-1 Αν πριν από κάθε μια εντολή $a = 0$, $b = 1$, $x = 2$, $y = 3$, τι τιμές θα έχουν οι μεταβλητές μετά την εκτέλεση κάθε μιας από τις παρακάτω εντολές;

α) `if (x > y)  x = y; else y = x;`
β) `if (x == y)  a = b; else b = a;`
γ) `if (x > 0)  a = b;`
δ) `if (x = y)  y = x;`
ε) `if (a = sqrt(b))  y = x;`
στ) `if (x > x)  if (y < y)  a = b;`

---

[6] There is always one more bug! (νόμος του Murphy).

[7] Επειδή αυτοί οι έλεγχοι κάνουν το πρόγραμμα κάπως πιο αργό, συνήθως βάζουμε (παρόμοιες) οδηγίες προς τον μεταγλωττιστή όπως θα μάθουμε αργότερα.

[8] Θα δεις στα ελληνικά τον όρο «εκσφαλμάτωση».

ζ) `if (a + b)  y = x;`

5-2 Ο αλγόριθμος υπολογισμών προβλέπει, ότι η μεταβλητή *y* πρέπει να πάρει τις παρακάτω τιμές, ανάλογα με τις τιμές των μεταβλητών *a* και *b*:

$y = a + 100$, αν $a \geq 0$ και $b = 0$
$y = b + 5$, αν $a \geq 0$ και $b \neq 0$
$y = b - 5$, αν $a < 0$ και $b = 0$
$y = a - 100$, αν $a < 0$ και $b \neq 0$

Δώσε τις αντίστοιχες εντολές στην C++.

5-3 α) Είπαμε παραπάνω ότι στην *P* { **if** (*S*) *Et* **else** *Ef* } *Q* η *Et* εκτελείται με προϋπόθεση *P* **&&** *S* και μας δίνει *Q*, ενώ η *Ef* με προϋπόθεση *P* **&&** (**!***S*) και μας δίνει *Q*.

Στη συνέχεια είπαμε ότι, πηγαίνοντας από το τέλος προς την αρχή, αν *Pt* και *Pf* είναι τέτοιες ώστε: *Pt* { *Et* } *Q* και *Pf* { *Ef* } *Q* τότε πριν από την **ifelse** θα πρέπει να ισχύει η (*Pt* **&&** *S*) **||** (*Pf* **&&** (**!***S*)).

Για να είναι τα παραπάνω συμβιβαστά θα πρέπει:

$$(((Pt \;\&\&\; S) \;||\; (Pf \;\&\&\; (!S))) \;\&\&\; S) \Rightarrow Pt$$

και

$$(((Pt \;\&\&\; S) \;||\; (Pf \;\&\&\; (!S))) \;\&\&\; (!S)) \Rightarrow Pf$$

Να τις αποδείξεις!

β) Είπαμε ακόμη ότι στην *P* { **if** (*S*) *Et* } *Q* η *Et* εκτελείται με προϋπόθεση *P* **&&** *S* και μας δίνει *Q*, ενώ (*P* **&&** (**!***S*)) ⇒ *Q*.

Στη συνέχεια είπαμε ότι, πηγαίνοντας από το τέλος προς την αρχή, αν *Pt* είναι τέτοια ώστε: *Pt* { *Et* } *Q* τότε πριν από την **if** θα πρέπει να ισχύει η (*Pt* **&&** *S*) **||** (*Q* **&&** (**!***S*)).

Για να είναι τα παραπάνω συμβιβαστά θα πρέπει:

$$(((Pt \;\&\&\; S) \;||\; (Q \;\&\&\; (!S))) \;\&\&\; S) \Rightarrow Pt$$

και

$$(((Pt \;\&\&\; S) \;||\; (Q \;\&\&\; (!S))) \;\&\&\; (!S)) \Rightarrow Q$$

Να τις αποδείξεις και αυτές!

Β Ομάδα

5-4 Ποια μαθηματική συνάρτηση υλοποιούν οι παρακάτω εντολές (όπου οι μεταβλητές *a*, *b*, *c* και *d* είναι τύπου `int`):

```
if ( a > b )  y = a; else y = b;
if ( c > y )  y = c;
if ( d > y )  y = d;
```

5-5 Ας υποθέσουμε ότι ο μισθός ενός εργαζόμενου προσαυξάνεται κατά 30 € για κάθε παιδί που έχει, αν έχει μέχρι τρία (3) παιδιά. Αν έχει περισσότερα από 3 παιδιά η προσαύξηση είναι 50 € για κάθε παιδί. Γράψε πρόγραμμα που θα διαβάζει τον βασικό μισθό και τον αριθμό παιδιών ενός υπαλλήλου και θα υπολογίζει το οικογενειακό επίδομα και τον συνολικό μισθό.

5-6 (Σύνθεση των Ασκ. 2-14 και 2-15) Γράψε πρόγραμμα που θα διαβάζει τις τιμές των R_1 και R_2 (θετικούς αριθμούς) και θα υπολογίζει και θα τυπώνει τη συνολική αντίσταση *R*. Πριν πάρει τις τιμές των αντιστάσεων, το πρόγραμμα θα ζητάει να διαβάσει, από το πληκτρολόγιο, τον τρόπο σύνδεσης των αντιστάσεων. Οι δεκτές απαντήσεις θα είναι:

- `'P'` για παράλληλη σύνδεση,
- `'S'` για σύνδεση εν σειρά.

Να διατυπωθεί η προϋπόθεση και το πρόγραμμα να την ελέγχει.

5-7 Ξαναγράψε το πρόγραμμα για το τριώνυμο της §5.6, έτσι ώστε:

1. Στην περίπτωση που $a = 0$ να υπολογίζει την απλή λύση $x1 = -c/b$, αν $b \neq 0$.
2. Αν $a = b = 0$ να εξετάζει τις περιπτώσεις $c = 0$ και $c \neq 0$.

3. Στην περίπτωση που *delta* < 0 να υπολογίζει και να τυπώνει τα

$ReX1 = ReX2 = -b/(2a)$

$ImX1 = ImX2 = \sqrt{-delta}\ /\ (2a)$

Η εκτύπωση των μιγαδικών θα πρέπει να γίνεται στη μορφή:

"`(`", *πραγματικό μέρος*, "`,`", *φανταστικό μέρος*, "`)`"

αφού δοθεί μήνυμα ότι οι ρίζες είναι μιγαδικές.

Γ Ομάδα

5-8 Απόδειξε ότι (`int a, b, c, y`):

```
// true
   if ( a > b )  y = a; else y = b;
   if ( c > y )  y = c;
// (y == a || y == b || y == c) && (y ≥ a && y ≥ b && y ≥ c)
```

5-9 Θέλουμε να αντικαταστήσουμε, στην προϋπόθεση του προβλήματος της δευτεροβάθμιας εξίσωσης, τη σύγκριση $a \neq 0$ με κάτι πιο «ρεαλιστικό». Ας υποθέσουμε ότι δεν έχουμε υπερχείλιση κατά τον υπολογισμό των $-b + \sqrt{b^2 - 4ac}$ ή της $-b - \sqrt{b^2 - 4ac}$. Ποια συνθήκη θα πρέπει να πληρούται ώστε να μην έχουμε υπερχείλιση όταν υπολογίζονται οι ρίζες;

5-10 (Συμπλήρωση της Ασκ. 2-7). Γράψε πρόγραμμα που θα διαβάζει τις καρτεσιανές συντεταγμένες ενός σημείου στο επίπεδο και θα υπολογίζει και θα τυπώνει τις πολικές r και ϕ. Αλλά, προσοχή: η atan επιστρέφει τιμή στο $(-\pi/2, \pi/2)$ ενώ θα πρέπει να έχουμε: $-\pi < \phi \leq \pi$.

Αν το σημείο βρίσκεται στον άξονα y, δηλ.: $x = 0$, τότε η ϕ θα γίνεται:

- $\pi/2$ αν $y > 0$,
- $-\pi/2$ αν $y < 0$,

ενώ θα παραμένει αόριστη αν έχουμε και $y = 0$.

Σύγκρινε τη ϕ που παίρνεις με αυτό που σου δίνει η atan2(y, x).

5-11 Απόδειξε ότι:

```
// (ia ≤ na || (ia = na+1) && ib ≤ nb)
   if ( ib > nb || ia ≤ na )  ia = ia + 1;
                      else  ib = ib + 1;
// ia ≤ na + 1
```

5-12 Απόδειξε ότι:

```
// true
   if ( x > y )  maxxy = x;
   if ( x <= y )  maxxy = y;
// ((maxxy ≥ x) && (maxxy ≥ y)) && ((maxxy == x) || (maxxy == y))
```

5-13 Ένας άλλος τρόπος για να διατυπώσουμε την απαίτηση στο πρόγραμμα για το μέγιστο είναι ο εξής:

$$((x \geq y)\ \&\&\ (maxxy == x))\ ||\ ((y \geq x)\ \&\&\ (maxxy == y))$$

Απόδειξε ότι:

```
// true
   if (x > y)  maxxy = x;
         else  maxxy = y;
// ((x ≥ y) && (maxxy == x)) || ((y ≥ x) && (maxxy == y))
```

5-14 Ψάξε τα εγχειρίδια του ΗΥ σου να δεις ποιό είναι το σύνολο χαρακτήρων που χρησιμοποιεί.

Γράψε ένα πρόγραμμα που θα διαβάζει ένα χαρακτήρα από το πληκτρολόγιο και θα τυπώνει το μήνυμα:

- `"ψηφίο"` αν ο χαρακτήρας ήταν ψηφίο,
- `"γράμμα"` αν ο χαρακτήρας ήταν γράμμα,
- `"κάτι άλλο"` αν δεν είναι ούτε γράμμα ούτε ψηφίο.

Αν έχεις κεφαλαία και μικρά γράμματα θα πρέπει να διαχωρίζονται επίσης.

Δώσε δύο λύσεις:

- Χρησιμοποιώντας τις συναρτήσεις που είδαμε στο προηγούμενο κεφάλαιο.
- Χωρίς να χρησιμοποιήσεις τις συναρτήσεις. Στην περίπτωση αυτή, αν έχεις και Ελληνικά και είναι εύκολος ο διαχωρισμός, να διαχωρίζονται επίσης.