κεφάλαιο

10

Προγράμματα με Κείμενα

Ο στόχος μας σε αυτό το κεφάλαιο:

Να μάθεις να γράφεις προγράμματα που διαχειρίζονται μη-αριθμητικά δεδομένα. Να χρησιμοποιείς τις τεχνικές αυτές και σε αριθμητικά δεδομένα.

Προσδοκώμενα αποτελέσματα:

Είναι μάλλον απίθανο να γράφεις «πραγματικό» πρόγραμμα χωρίς να χρειαστεί να διαχειριστείς κείμενο, έστω και αν πρόκειται για κείμενο με αριθμούς. Θα μάθεις να γράφεις πιο «επαγγελματικά» προγράμματα.

Έννοιες κλειδιά:

- *ορμαθός χαρακτήρων*
- *τύπος (κλάση) string*
- *σύνδεση ορμαθών*
- *αναζήτηση υποορμαθού*
- *πίνακες χαρακτήρων*
- *μετατροπή ορμαθού σε αριθμό*
- *μετατροπή αριθμού σε ορμαθό*

Περιεχόμενα:

Εισαγωγικές Παρατηρήσεις:
Αν γράψεις στο πρόγραμμά σου:

```
cout << "πάν μέτρον εκατό πόντοι" << endl;
```

θα περιμένεις ως αποτέλεσμα, σύμφωνα με όσα έχουμε μάθει μέχρι τώρα, να δεις στην οθόνη

```
πάν μέτρον εκατό πόντοι
```

Ξέρουμε λοιπόν πώς να στείλουμε ένα κείμενο στην οθόνη μας μέσω του ρεύματος *cout*. Δεν ξέρουμε όμως πώς να αποθηκεύσουμε ένα κείμενο στη μνήμη ούτε πώς να το επεξεργαστούμε.

Η πιο απλή ιδέα, σύμφωνα με αυτά που ξέρουμε ήδη, είναι να δούμε το κείμενο ως πίνακα χαρακτήρων:

```
char a[] = { 'π','ά','ν',' ','μ','έ','τ','ρ','ο','ν',' ',
             'ε','κ','α','τ','ό',' ','π','ό','ν','τ','ο','ι' };
```

Σωστό! Και αν δώσεις:

```
cout << a << endl;
```

θα δεις:

```
πάν μέτρον εκατό πόντοι
```

(είναι όμως πιθανό να δεις το κείμενό σου να ακολουθείται από μερικά «σκουπίδια».)

Η C++ σου επιτρέπει να κάνεις τη δήλωση πολύ πιο απλά:

```
char b[] = "πάν μέτρον εκατό πόντοι";
```

και αν δώσεις:

```
cout <<  b << endl;
```

θα δεις σίγουρα:

```
πάν μέτρον εκατό πόντοι
```

Έχουν διαφορά οι δύο δηλώσεις; Ναι. Αν ζητήσεις:

```
cout << (sizeof a) << "  " << (sizeof b) << endl;
```

θα πάρεις:

```
            23  24
```

Η διαφορά οφείλεται σε ένα `char(0)` (ή αλλιώς: `'\0'`), που μπαίνει ως τελευταίος χαρακτήρας στον b. Αυτό είναι κληρονομιά από τη C, που χρησιμοποιεί αυτόν τον χαρακτήρα ως *φρουρό* στους αλγόριθμους επεξεργασίας κειμένων της βιβλιοθήκης της. Το ότι στην πρώτη περίπτωση δεν έχουμε αυτόν τον `char(0)` έχει ως αποτέλεσμα να παίρνουμε τα «σκουπίδια» σε ορισμένες περιπτώσεις.

Η βασική ιδέα λοιπόν είναι αυτή: *Το κείμενο παριστάνεται ως πίνακας με στοιχεία τύπου* `char`. Η C έμεινε σε αυτό και απλώς περιλαμβάνει στις βιβλιοθήκες της πολλές συναρτήσεις που βοηθούν στην επεξεργασία τέτοιων πινάκων. Αυτές οι συναρτήσεις έχουν κληρονομηθεί και στη C++. Αλλά, στη C++ σχεδιάστηκε ένας ειδικός τύπος (κλάση) με το όνομα `string` (ορμαθός) που έχει ενσωματωμένες πολλές μεθόδους που μας χρειάζονται για να διαχειριστούμε ένα κείμενο.

Για να δουλέψεις με τον τύπο `string` θα πρέπει να περιλάβεις στο πρόγραμμά σου το αρχείο string ("`#include <string>`").

10.1 Δήλωση Μεταβλητών Τύπου *string*

Η πιο απλή περίπτωση δήλωσης είναι η εξής[1]:

[1] Αν δεν βάλεις το "`using namespace std`" θα πρέπει να δηλώνεις "`std::string`".

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string s0;
```

Η *s0* έχει ως τιμή έναν ορμαθό *μηδενικού μήκους*, χωρίς περιεχόμενο. Στη συνέχεια μπορείς να του δώσεις την τιμή που θέλεις, όπως θα δούμε παρακάτω.

Πάντως, μπορείς να του δώσεις τιμή από την αρχή, με τη δήλωση:

```
   string s1( "ένας ορμαθός της C++" );
```

Να σου υπενθυμίσουμε εδώ, ότι αν ο ορμαθός που θα εκχωρήσεις ως τιμή έχει κάποιους ειδικούς χαρακτήρες θα πρέπει να χρησιμοποιήσεις τον χαρακτήρα "\" (δες τον Πίν. 4-3). Π.χ.:

```
   string path( "f:\\cwork\\prog7_1.cpp" );
   string quest( "what\'s this\?" );
```

Αν μετά τα παραπάνω δώσεις τις εντολές:

```
   cout << "    s0: " << s0 << endl;
   cout << "    s1: " << s1 << endl;
   cout << "  path: " << path << endl;
   cout << " quest: " << quest << endl;
```

θα πάρεις:

```
    s0:
    s1: ένας ορμαθός της C++
  path: f:\cwork\prog7_1.cpp
 quest: what's this?
```

Δεν μπορείς να δώσεις ως αρχική τιμή μια σταθερά τύπου **char**. Έτσι, η δήλωση:

```
   string a( 'a' );
```

είναι λάθος!

Μπορείς όμως να δώσεις:

```
   string a( 10,'a' );
```

οπότε η:

```
   cout << "     a: " << a << endl;
```

θα δώσει:

```
     a: aaaaaaaaaa
```

Δηλαδή η a πήρε ως τιμή έναν ορμαθό με 10 **'a'**. Αν λοιπόν θέλεις να δώσεις ως τιμή ένα **'a'** μπορείς να δώσεις ένα από τα παρακάτω:

```
   string a( 1,'a' );
   string a( "a" );
```

Μπορείς να δώσεις ως αρχική τιμή έναν πίνακα τύπου **char**:

```
   char b[] = "πάν μέτρον εκατό πόντοι";
   string s2( b );
```

Μπορείς όμως να δώσεις ως αρχική τιμή και την τιμή μιας άλλης μεταβλητής τύπου **string**:

```
   string s2( a );
```

ή έναν *υποορμαθό* μιας μεταβλητής τύπου **string**:

```
   string s3( s1, 5, 7 );
```

Με αυτήν τη δήλωση δίνουμε ως αρχική τιμή στην *s3* έναν υποορμαθό της *s1* που ξεκινάει από τον χαρακτήρα 5 (αρχίζουμε το μέτρημα από 0) και έχει μήκος 7 χαρακτήρες. Αν *s1* είναι αυτή που δηλώσαμε παραπάνω, τότε, αν ζητήσουμε να τυπωθεί η τιμή της *s3*, θα πάρουμε:

```
ορμαθός
```

10.2 Μετατροπή σε Απλό Πίνακα

Πριν προχωρήσουμε, ας δούμε πώς μπορείς να πάρεις το «περιεχόμενο» ενός αντικειμένου τύπου *string*, σε έναν πίνακα με στοιχεία τύπου `char`. Υπάρχουν δύο *μέθοδοι* γι' αυτό: η *c_str* και η *data* που κάνουν την ίδια –περίπου– δουλειά.

Ας πούμε ότι έχουμε:

```
const int sz = 50;
string s1( "ένας ορμαθός της C++" );
char ac[sz];
```

και παίρνουμε το `s1.c_str()`. Αυτό έχει `char(0)` στο τέλος. Έτσι, μπορείς να το χειριστείς με τις συναρτήσεις της C, π.χ. να τα αντιγράψεις σε έναν πίνακα σαν τον ac:

```
strcpy( ac, s1.c_str() );
```

Για τη *strcpy()* θα τα πούμε στη συνέχεια.

Ποια η διαφορά του `s1.data()`; *Δεν υπάρχει εγγύηση ότι θα έχει* `char(0)` *στο τέλος*. Έτσι, η

```
strcpy( ac, s1.data() );
```

δεν είναι σίγουρο ότι θα δουλέψει. Πάντως σε πολλές περιπτώσεις θα δεις να συμπεριφέρεται ακριβώς σαν το `s1.c_str()`.

Και για τις δύο συναρτήσεις υπάρχει ένας περιορισμός: *Μην προσπαθήσεις να τροποποιήσεις τους πίνακες που βγάζουν*. Για παράδειγμα μην προσπαθήσεις να κάνεις κάτι σαν:

```
(c1.c_str())[3] = 'Σ';
```

10.3 Εκχώρηση Τιμής και Αντιμετάθεση

Χρησιμοποιώντας τον τελεστή "`=`" της εκχώρησης, μπορείς να εκχωρήσεις σε μια μεταβλητή τύπου *string*:

- μια τιμή τύπου *string*,
- μια τιμή τύπου `char`,
- έναν ορμαθό χαρακτήρων

Έτσι, αν έχεις δηλώσει:

```
string s0;
string s1( "ένας ορμαθός της C++" );
char b[] = "πάν μέτρον εκατό πόντοι";
```

μπορείς να δώσεις:

```
s0 = s1;                cout << "    s0: " << s0 << endl;
s0 = 'a';               cout << "    s0: " << s0 << endl;
s0 = b[4];              cout << "    s0: " << s0 << endl;
s0 = "what\'s this\?";  cout << "    s0: " << s0 << endl;
```

και θα πάρεις από τις εντολές εξόδου:

```
 s0: ένας ορμαθός της C++
 s0: a
 s0: μ
 s0: what's this?
```

Εκτός όμως από το "`=`", μπορείς να εκχωρήσεις τιμή και με κάποια από τις μορφές της μεθόδου *assign()*: Ας πούμε ότι έχουμε δηλώσει:

```
string s1( "ένας ορμαθός της C++" );
char b[] = "0123456789";
string s2;
```

1. Μπορείς να δώσεις: "`s2.assign(s1)`" που είναι ίδιο με το "`s2 = s1`". Οι

```
s2.assign( s1 );              cout << s2 << endl;
```

θα δώσουν:

```
ένας ορμαθός της C++
```

2. Μπορείς να δώσεις: "`s2.assign(s1, p, n)`" που σημαίνει: Δώσε ως τιμή στην *s2* το κομμάτι της *s1* που έχει *n* (πλήθος) χαρακτήρες και ξεκινάει από τον χαρακτήρα *p*. Π.χ. οι

```
   s2.assign( s1, 5, 7 );          cout << s2 << endl;
```

θα δώσουν:

```
ορμαθός
```

3. Μπορείς να δώσεις: "`s2.assign(cs)`" όπου *cs* ορμαθός ή πίνακας με στοιχεία τύπου `char`. Π.χ. οι:

```
   s2.assign( b );                 cout << s2 << endl;
   s2.assign( "0123456789" );      cout << s2 << endl;
```

θα δώσουν:

```
0123456789
0123456789
```

3. Μπορείς να δώσεις: "`s2.assign(cs, n)`" όπου *cs* ορμαθός ή πίνακας με στοιχεία τύπου `char` και *n* ακέραιος, από 0 μέχρι το μήκος του *cs*. Στην *s2* εκχωρούνται οι *n* πρώτοι χαρακτήρες του *cs*. Π.χ. οι:

```
   s2.assign( b, 5 );              cout << s2 << endl;
   s2.assign( "0123456789", 5 );   cout << s2 << endl;
```

θα δώσουν:

```
01234
01234
```

4. Τέλος, μπορείς να δώσεις "`s2.assign(n, c)`" όπου *n* φυσικός και *c* τιμή τύπου `char`. Τιμή της *s2* είναι ένας ορμαθός με *n* φορές τον *c*. Π.χ. η:

```
   s2.assign( 5, 'a' );            cout << s2 << endl;
```

θα δώσει:

```
aaaaa
```

Αντιμετάθεση τιμών ξέρεις να κάνεις! Αν πρόκειται για τιμές τύπου *string* τα πράγματα είναι πιο εύκολα: υπάρχει σχετική μέθοδος που λέγεται *swap()*. Δες τις παρακάτω εντολές:

```
   cout << " s1: " << s1 << "  s2: " << s2 << endl;
   s2.swap( s1 );
   cout << " s1: " << s1 << "  s2: " << s2 << endl;
```

που δίνουν:

```
 s1: ένας ορμαθός της C++  s2: aaaaa
 s1: aaaaa  s2: ένας ορμαθός της C++
```

Δηλαδή: η "`s2.swap(s1)`" έχει ως αποτέλεσμα την αντιμετάθεση τιμών των μεταβλητών *s1* και *s2*. Αλλά, όπως θα δούμε αργότερα, η αντιμετάθεση με τη *swap()* είναι ασφαλέστερη από την αντιμετάθεση που ξέρουμε.

10.4 Ανάγνωση και Γραφή

Ας ξεκινήσουμε με το γράψιμο, για το οποίο έχουμε ήδη μιλήσει. Απλώς θα συμπληρώσουμε ότι ο τελεστής "`<<`" στέλνει τιμές μεταβλητών τύπου `string`, όχι μόνον μέσω του *cout* στην οθόνη, αλλά και μέσω οποιουδήποτε ρεύματος *ofstream* σε αρχείο text.

Πώς μπορούμε να διαβάσουμε την τιμή μιας μεταβλητής *a* τύπου `string` από το πληκτρολόγιο ή από κάποιο αρχείο; Δηλαδή, δεν μπορούμε να διαβάσουμε την τιμή της *a* από το *cin* με τον "`>>`"; Μπορούμε, αλλά... Ας κάνουμε μια δοκιμή. Δίνουμε:

```
   cout << "Τι εχεις να πεις;" << endl;
   cin >> a;
   cout << a << endl;
```

και έχουμε την εξής εκτέλεση:

```
Τι εχεις να πεις;
των φρονίμων τα παιδιά τα κάνει κρεμαστάρια<enter>
των
```

Δηλαδή, μόλις βρήκε κενό στάματησε το διάβασμα· το ίδιο θα γινόταν αν έβρισκε τέλος γραμμής (`'\n'`) ή στηλοθέτη (`'\t'`).

Η ανάγνωση γίνεται με την[2]:

```
getline( cin, a, '\n' );
```

που λέει τα εξής:

- διάβασε από το ρεύμα *cin*,
- αποθηκεύοντας αυτά που διαβάζεις στην *a*,
- μέχρι να βρεις τέλος γραμμής (`'\n'`).

Ως πρώτο όρισμα μπορείς να βάλεις και ρεύμα τύπου *ifstream*, δηλαδή μπορείς να διαβάζεις και από αρχείο.

Αν, ως απάντηση στην `getline(cin, a, '\n')`, πιέσεις απλώς το πλήκτρο <enter>, χωρίς να γράψεις κείμενο, η *a* γίνεται *κενή*.

Η *getline()* είναι ένα πολύ καλό εργαλείο και για την ανάγνωση αρχείων text. Ας πούμε ότι στο αρχείο `inpData.txt` έχουμε γραμμές της μορφής:

```
873\t537\tΑνδρικόπουλος\t2510 997 799\n
```

(με τα '`\t`' και '`\n`' συμβολίζουμε τους χαρακτήρες "tab" και "newline" αντιστοίχως.) Αν έχουμε δηλώσει:

```
ifstream tin( "inpData.txt" );
string s1, s2, s3, s4;
```

μπορούμε να διαβάσουμε μια γραμμή ως εξής:

```
getline( tin, s1, '\t' );
getline( tin, s2, '\t' );
getline( tin, s3, '\t' );
getline( tin, s4, '\n' );
```

Βέβαια, οι δύο ακέραιοι, στην αρχή, έχουν διαβαστεί στις *s1* και *s2* ως *string*. Στη συνέχεια θα δεις πώς μπορούμε να τις μετατρέψουμε σε `int`. Παρομοίως, αν έχουμε μια γραμμή με μια ημερομηνία

```
19.11.2008
```

μπορούμε να τη διαβάσουμε ως εξής:

```
getline( tin, s1, '.' );
getline( tin, s2, '.' );
getline( tin, s3, '\n' );
```

Αντί για την (*std*`::`)*getline()* μπορείς να αποθηκεύσεις αυτό που διαβάζεις σε ορμαθό της C –δηλαδή πίνακα χαρακτήρων– με τη *getline* της *cin*. Αν, ας πούμε, έχεις δηλώσει:

```
char q[100];
```

μπορείς να διαβάσεις κείμενο από το πληκτρολόγιο στον q και από εκεί να το αντιγράψεις στην *a*, ως εξής:

```
cin.getline( q, 100 );
a.assign( q );
```

Εδώ ζητούμε να διαβαστεί μια γραμμή από το πληκτρολόγιο και να αποθηκευτεί στον *q*· αλλά να διαβαστούν το πολύ 99 (= 100 – 1) χαρακτήρες. Στο μήκος αυτό (100) περιλαμβάνεται και μια θέση για το `char(0)` που θα σημειώνει το τέλος του κειμένου στον *q*. Στη συνέχεια, με την `a.assign(q)`, αντιγράφουμε το περιεχόμενο του *q* (χωρίς το `char(0)`) στην *a*.

[2] `std::getline` για την ακρίβεια...

Η απάντηση <enter> χωρίς κείμενο στη `cin.getline()` έχει παρόμοιο αποτέλεσμα με αυτό που είδαμε στην `std::getline`: ο *q* παραμένει «κενός» (με `char(0)` στη θέση q[0]).

Ας δούμε και ένα

Παράδειγμα

Θα κάνουμε πιο ευέλικτο το πρόγραμμα αντιγραφής αρχείων που είδαμε στο Κεφ. 8. Θα βάλουμε σε μεταβλητές τα ονόματα των αρχείων:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
   ifstream s;
   ofstream t;
   string sFlNm, tFlNm;
   char ch;

   cout << "όνομα αρχείου: ";  getline( cin, sFlNm, '\n' );
   s.open( sFlNm.c_str() );
   if ( s.fail() )
      cerr << "δεν μπορώ να ανοίξω το αρχείο " << sFlNm << endl;
   else
   {
      cout << "όνομα αντιγράφου: ";  getline( cin, tFlNm, '\n' );
      t.open( tFlNm.c_str() );
      if ( t.fail() )
      {
         s.close();
         cerr << "δεν μπορώ να δημιουργήσω το " << tFlNm << endl;
      }
      else // ok και τα δύο αρχεία ανοικτά
      {
         s.get( ch );
         while ( !s.fail() && !t.fail() )
         {  t.put( ch );  s.get( ch );  } // while
         if ( !s.eof() )
            cerr << "πρόβλημα κατά την αντιγραφή" << endl;
         t.close();
         s.close();
      } // if ( t.fail
   } // if ( s.fail
} // main
```

Όπως βλέπεις, βάλαμε μεν τα ονόματα των αρχείων σε μεταβλητές τύπου `string`, αλλά δεν μπορούμε να περάσουμε τα ονόματα των μεταβλητών στις *open* που περιμένουν ορμαθό χαρακτήρων ή πίνακα με στοιχεία τύπου `char`. Έτσι, περάσαμε τα `sFlNm.c_str()` και `tFlNm.c_str()`.

10.5 Συγκρίσεις

Μπορείς να συγκρίνεις δύο ορμαθούς μεταξύ τους είτε με τη μέθοδο *compare()* είτε με τους γνωστούς τελεστές (`<`, `>`, `==` κλπ).

Η "`s1.compare(s2)`" δίνει ακέραιη

- *αρνητική τιμή* αν η τιμή της *s1* προηγείται λεξικογραφικώς της τιμής της *s2*,
- *θετική* τιμή αν *η τιμή της s1 έπεται λεξικογραφικώς της τιμής της s2*,
- *μηδέν* (0) αν *οι τιμές των s1 και s2 είναι ίσες*.

Αν λοιπόν δηλώσουμε:

```
string s1( "abc" );
string s2( "bce" );
string s3( "abcd" );
string s4( "abc" );
```

οι:

```
cout << s1.compare(s2) << "  " << s2.compare(s1) << endl;
cout << s1.compare(s3) << "  " << s3.compare(s1) << endl;
cout << s1.compare(s4) << "  " << s4.compare(s1) << endl;
```

θα δώσουν:

```
-1  1
-1  1
0  0
```

που σημαίνουν:

- Η τιμή της *s1* ("**abc**") λεξικογραφικώς προηγείται της τιμής της *s2* ("**bce**"), ενώ
- η τιμή της *s2* λεξικογραφικώς έπεται της τιμής της *s1*. Παρομοίως,
- η τιμή της *s1* λεξικογραφικώς προηγείται της τιμής της *s3* ("**abcd**"), ενώ
- η τιμή της *s3* λεξικογραφικώς έπεται της τιμής της *s1*. Τέλος,
- οι τιμές των *s1* και *s4* είναι ίσες.

Μπορείς να συγκρίνεις και την τιμή μιας μεταβλητής τύπου **string** με έναν ορμαθό χαρακτήρων ή έναν πίνακα με στοιχεία τύπου **char**. Αν, για παράδειγμα

```
char a[] = "abd";
```

οι:

```
cout << s1.compare( "cdef" ) << endl;
cout << s2.compare( a ) << endl;
```

θα δώσουν:

```
-2
1
```

Μπορείς λοιπόν να γράφεις στο πρόγραμμά σου:

```
if ( s1.compare(s2) < 0 )
   cout << "το " << s1 << " προηγείται λεξικογραφικώς του "
        << s2 << endl;
else if ( s1.compare(s2) > 0 )
   cout << "το " << s1 << " έπεται λεξικογραφικώς του "
        << s2 << endl;
else
   cout << "τα " << s1 << " και " << s2
        << " είναι ίσα" << endl;
```

Πάντως μπορείς να χρησιμοποιήσεις και τους γνωστούς μας τελεστές σύγκρισης:

αντί για **s1.compare(s2)** θ **0** μπορείς να γράφεις **s1** θ **s2**

όπου θ κάποιος από τους "**>**", "**>=**", "**<**", "**<=**", "**==**", "**!=**".

Έτσι, μπορείς να γράψεις την παραπάνω διπλή **if** και ως εξής:

```
if ( s1 < s2 )
   cout << "το " << s1 << " προηγείται λεξικογραφικώς του "
        << s2 << endl;
else if ( s1 > s2 )
   cout << "το " << s1 << " έπεται λεξικογραφικώς του "
        << s2 << endl;
else
   cout << "τα " << s1 << " και " << s2
        << " είναι ίσα" << endl;
```

όπως και:

```
if ( s1 == "abc" )  cout << "eq" << endl;
```

Ας δούμε ένα παράδειγμα χρήσης της δυνατότητας σύγκρισης.

Παράδειγμα

Έχουμε έναν πίνακα με στοιχεία τύπου `string`:

```
   string v[N] = { "PASCAL", "BASIC", "FORTRAN", "COBOL", "RPG",
                   "ALGOL", "LISP", "PROLOG", "LOGO", "C++",
                   "PL/I", "ADA", "BCPL", "SNOBOL", "APL", "C" };
```

και θέλουμε να ταξινομήσουμε τα στοιχεία κατ' αλφαβητική σειρά.

Μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο που αντιγράφουμε από την §9.5.2:

```
// ταξινόμηση
   for ( k = N; k >= 2; k = k-1 )
   {
      meg = maxNdx( v, 1, k );
      sv = v[meg];  v[meg] = v[k];  v[k] = sv;
   } // for
```

μόνο που εδώ τα στοιχεία μας είναι στις θέσεις[3] v[0] μέχρι v[*N*-1]. Ακόμη η αντιμετάθεση τιμών των v[*mxp*] και v[*k*] μπορεί να γίνει με τη *swap*:

```
// ταξινόμηση
   for ( k = N-1; k >= 1; k = k-1 )
   {
      mxp = maxNdx( v, N, 0, k );
      v[mxp].swap( v[k] );
   } // for
```

Και με τη *maxNdx* τι γίνεται; Την αντιγράφουμε όπως είναι αλλάζοντας μόνον τον τύπο των στοιχείων του πίνακα από `int` σε *string*:

```
   int maxNdx( const string x[], int n, int from, int upto )
```

Να ολόκληρο το πρόγραμμα:

```
#include <iostream>
#include <string>
using namespace std;

   int maxNdx( const string x[], int n, int from, int upto );

int main()
{
   const int N = 16;

   string v[N] = { "PASCAL", "BASIC", "FORTRAN", "COBOL", "RPG",
                   "ALGOL", "LISP", "PROLOG", "LOGO", "C++",
                   "PL/I", "ADA", "BCPL", "SNOBOL", "APL", "C" };
   int  k, mxp;
// ταξινόμηση
   for ( k = N-1; k >= 1; k = k-1 )
   {
      mxp = maxNdx( v, N, 0, k );
      v[mxp].swap( v[k] );
   } // for
// αποτέλεσμα
   for ( k = 0; k <= N-1; k=k+1 )  cout << v[k] << endl;
} // main
```

Αποτέλεσμα:

```
ADA
ALGOL
APL
```

---

[3] Φυσικά δεν υπάρχει πρόβλημα αν θέλεις να βάλεις φρουρούς στην αρχή και στο τέλος. Άλλαξε τη δήλωση σε:

```
   string v[N+2] = { " ", "PASCAL",... "C", " " };
```

και μετά δώσε:

```
   v[0].assign(5, char(0));  v[N+1].assign(5,char(255));
```

```
BASIC
BCPL
C
C++
COBOL
FORTRAN
LISP
LOGO
PASCAL
PL/I
PROLOG
RPG
SNOBOL
```

10.6 Μήκος Ορμαθού – Κενός Ορμαθός

Αφού τιμή μιας μεταβλητής τύπου **string** είναι ένας ορμαθός χαρακτήρων, έχει νόημα το μήκος της, που είναι ακριβώς το μήκος της τιμής της. Το μήκος μας δίνεται από τη μέθοδο **length** (μήκος).

Ας πούμε ότι έχουμε δηλώσει:

```
string s0, s1( "ένας ορμαθός της C++" );
string path( "f:\\cwork\\prog7_1.cpp" ),
       quest( "what\'s this\?" );
string a( 10,'a' ), c( "a" );
```

και δίνουμε:

```
cout << s0.length() << "  " << s1.length() << "  "
     << path.length() << "  " << quest.length() << "  "
     << a.length() << "  " << c.length() << endl;
```

Αποτέλεσμα:

```
0  20  20  12  10  1
```

Πρόσεξε ότι στην *path* το "\\" μετράει ως *ένας* χαρακτήρας. Ως ένας χαρακτήρας μετράνε και τα "\'" και "\?" στην *quest*.

Εκτός από τη *length()* υπάρχει και η *size()* που κάνει ακριβώς τα ίδια. Θα πάρεις τα παραπάνω αποτελέσματα και με τη:

```
cout << s0.size() << "  " << s1.size() << "  "
     << path.size() << "  " << quest.size() << "  "
     << a.size() << "  " << c.size() << endl;
```

Πρόσεξε ακόμη ότι το μήκος της *s0* είναι μηδέν. Η τιμή της *s0* είναι ο κενός ορμαθός, χωρίς κάποιον χαρακτήρα. Αυτό ελέγχεται με τη μέθοδο *empty()*, που μας δίνει τιμή **true** αν η τιμή της μεταβλητής είναι ο κενός ορμαθός (μηδενικό μήκος) και **false** αν έχει έστω και έναν χαρακτήρα. Π.χ. οι:

```
if ( s0.empty() )  cout << " το s0 είναι άδειο!" << endl;
             else  cout << " το s0 κάτι έχει μέσα" << endl;
if ( s1.empty() )  cout << " το s1 είναι άδειο!" << endl;
             else  cout << " το s1 κάτι έχει μέσα" << endl;
```

θα δώσουν:

```
το s0 είναι άδειο!
το s1 κάτι έχει μέσα
```

10.6.1 Το Μέγιστο Μήκος Ορμαθού

Και ποιο είναι το μέγιστο μήκος ορμαθού που μπορώ να αποθηκεύσω σε μεταβλητή *string*; Το βρίσκεις με τη *max_size()*:

```
string s1;
```

```
   cout << "max_size = " << s1.max_size() << endl;
```

Αποτέλεσμα (BC++ v.5.5):

```
max_size = 4294967281
```

10.6.2 Ο Τύπος "size_type"

Ο τύπος του αποτελέσματος της *length()*, της *size()* και της *max_size()* είναι *string*::*size_type*, που είναι ο τύπος της C++ για τα μεγέθη των αντικειμένων που παριστάνονται στον τύπο *string*.

Για παρόμοιες δουλειές η C μας κληρονομεί τον **size_t** που είναι ο τύπος του αποτελέσματος που επιστρέφει ο τελεστής "**sizeof**". Αν ψάξεις το **cstddef** (ή το **stddef.h**) θα βρεις τον ορισμό:

```
typedef unsigned int size_t;
```

ή κάτι παρόμοιο.

Μπορείς να θεωρείς ότι οι δυο τύποι είναι ταυτόσημοι.[4]

10.7 Σύνδεση – Επισύναψη

Πολύ συχνά έχουμε ανάγκη να **συνδέσουμε** (concatenate) δυο ή περισσότερους ορμαθούς. Αυτό μπορούμε να το κάνουμε «προσθέτοντας» ορμαθούς με το "**+**". Αν, για παράδειγμα, έχουμε δηλώσει:

```
   string s1( "πέντε" ), s2( "στο χέρι παρά δέκα" ), s3;
```

τότε οι εντολές:

```
   cout << ("κάλλιο " + s1 + " και " + s2 + " και καρτέρει")
        << endl;
   s3 = "κάλλιο " + s1 + " και " + s2 + " και καρτέρει";
   cout << s3 << endl;
```

θα μας δώσουν:

```
κάλλιο πέντε και στο χέρι παρά δέκα και καρτέρει
κάλλιο πέντε και στο χέρι παρά δέκα και καρτέρει
```

Με τον τελεστή "**+**" μπορούμε να συνδέσουμε:

- δύο μεταβλητές τύπου **string**,
- μεταβλητή τύπου **string** και ορμαθό χαρακτήρων,
- μεταβλητή τύπου **string** και πίνακα με στοιχεία τύπου **char** (το περιεχόμενό του θα πρέπει να τελειώνει με '**\0**'.)

Το αποτέλεσμα της πράξης μπορεί να εκχωρηθεί σε μια μεταβλητή τύπου **string**.

Πολύ συχνά υπάρχει ανάγκη να **επισυνάψουμε** (append) έναν ορμαθό στην τιμή μιας μεταβλητής τύπου **string**. Π.χ. αυτά που κάναμε στο παράδειγμα θα μπορούσαν να γίνουν και ως εξής:

```
   s3 = "κάλλιο ";
   s3 = s3 + s1;  s3 = s3 + " και ";  s3 = s3 + s2;
   s3 = s3 + " και καρτέρει";
```

Για την επισύναψη ο **string** έχει ειδική μέθοδο, την *append()*. Τα ίδια πράγματα θα μπορούσαν να γραφούν και ως εξής[5]:

---

[4] Πάντως, για να μην χρησιμοποιεί η C++ τον "**size_t**" διατηρεί το δικαίωμα για αλλαγές.

[5] Αλλά και ως εξής:

```
   s3 = "κάλλιο ";   s3 += s1;  s3 += " και ";
                     s3 += s2;  s3 += " και καρτέρει";
```

```
s3 = "κάλλιο ";
s3.append( s1 );  s3.append( " και " );  s3.append( s2 );
s3.append( " και καρτέρει" );
```

10.8 Αναζήτηση

Ένα πολύ συνηθισμένο πρόβλημα επεξεργασίας κειμένου είναι η αναζήτηση ενός χαρακτήρα ή μιας λέξης ή ενός κομματιού κειμένου μέσα σε κάποιο άλλο κείμενο. Αυτό το ξέρεις καλά, αφού σίγουρα έχεις χρησιμοποιήσει πολλές φορές την εντολή **find** (ή **αναζήτησε**) του κειμενογράφου σου. Η κλάση `string` είναι εφοδιασμένη με αρκετές μεθόδους για τη δουλειά αυτή.

Ας ξεκινήσουμε με τη `find` (βρες). Αν έχεις δηλώσει "`string s`" και αν *t* είναι

- μια μεταβλητή τύπου `string` ή
- ένας ορμαθός χαρακτήρων ή
- πίνακας χαρακτήρων ή
- τιμή τύπου `char`

τότε

- Η `s.find(`*t*`)` ψάχνει την τιμή της *s*, *από αριστερά προς τα δεξιά*, για να εντοπίσει τον «υποορμαθό» *t*. Αν τον βρει επιστρέφει τον δείκτη του χαρακτήρα στην *s*, που αρχίζει ο *t*.
- Η `s.find(`*t*, *i*`)`, όπου *i* τιμή τύπου *string*`::`*size_type*, ψάχνει την τιμή της *s*, *από αριστερά προς τα δεξιά*, ξεκινώντας από τιμή δείκτη *i*, για να εντοπίσει τον «υποορμαθό» *t*. Αν τον βρει επιστρέφει τον δείκτη, στην *s*, που αρχίζει ο *t*.

Αν δεν βρεθεί η *t* μέσα στην *s* η *find* επιστρέφει μια απίθανη τιμή. Αυτήν την τιμή μπορείς να την ελέγχεις ως *string*`::`*npos*. Ο τύπος του αποτελέσματος είναι *string*`::`*size_type*.

Παράδειγμα

Γράφει ο Ποιητής:

ΑΞΙΟΝ ΕΣΤΙ στὸ πέτρινο πεζούλι
ἀντικρὺ τοῦ πελάγους ἡ Μυρτὼ νὰ στέκει
σὰν ὡραῖο ὀκτὼ ἢ σὰν κανάτι
μὲ τὴν ψάθα τοῦ ἥλιου στὸ ἕνα χέρι

Μπορούμε να αποθηκεύσουμε αυτό το κείμενο σε μια μεταβλητή

```
string text1;
```

ως εξής: [6]

```
text1 = "ΑΞΙΟΝ ΕΣΤΙ στο πέτρινο πεζούλι\n"
        "αντικρύ του πελάγους η Μυρτώ να στέκει\n"
        "σαν ωραίο οκτώ ή σαν κανάτι\n"
        "με την ψάθα του ήλιου στο ένα χέρι";
```

Με τα `'\n'` αποθηκεύουμε και τις αλλαγές γραμμής.

Δηλώνουμε ακόμη:

```
char a[] = "Μυρτώ";
```

Οι εντολές:

```
if ( text1.find("Μυρτώ") == string::npos )
```

Λέγαμε στο Κεφ. 2 ότι, για το αριθμητικό νόημα του "`+`", "`v += a`" σημαίνει: "`v = v + a`". Το ίδιο ισχύει και όταν ο τελεστής "`+`" σημειώνει τη σύνδεση.

[6] Μπορείς να το γράψεις και έτσι:

```
text1 = "ΑΞΙΟΝ ΕΣΤΙ στο πέτρινο πεζούλι\n";
text1.append( "αντικρύ του πελάγους η Μυρτώ να στέκει\n" );
text1.append( "σαν ωραίο οκτώ ή σαν κανάτι\n" );
text1.append( "με την ψάθα του ήλιου στο ένα χέρι" );
```

```
      cout << "Μυρτώ δεν βρέθηκε" << endl;
   else
      cout << text1.find("Μυρτώ") << endl;
   if ( text1.find("Μυρσίνη") == string::npos )
      cout << "Μυρσίνη δεν βρέθηκε" << endl;
   else
      cout << text1.find( "Μυρσίνη" ) << endl;
   cout << text1.find(a) << endl;
   cout << text1.find('M') << endl;
   cout << text1.find(a[0]) << endl;
```

θα δώσουν:

```
54
Μυρσίνη δεν βρέθηκε
54
54
54
```

Δηλαδή:

- Η λέξη `"Μυρτώ"` υπάρχει στη θέση με δείκτη 54 (γραμμές 1 και 3).
- Το γράμμα `'M'` υπάρχει στη 54 (γραμμές 4 και 5).
- Η λέξη `"Μυρσίνη"` *δεν* υπάρχει στην *text1* (γραμμή 2).

Παρατήρηση:▶

Για όσους δεν το πρόσεξαν (και δεν θύμωσαν) ήδη: Ο σωστός τρόπος να γράψουμε το πρόγραμμα είναι:

```
   size_t pos;
// . . .
   pos = text1.find("Μυρτώ");
   if ( pos == string::npos ) cout << "Μυρτώ δεν βρέθηκε" << endl;
                         else cout << pos << endl;
   pos = text1.find("Μυρσίνη");
   if ( pos == string::npos ) cout << "Μυρσίνη δεν βρέθηκε" << endl;
                         else cout << pos << endl;
   cout << text1.find(a) << endl;
   cout << text1.find('M') << endl;
   cout << text1.find(a[0]) << endl;
```

Η αναζήτηση κειμένου είναι χρονοβόρα διαδικασία και δεν αφήνουμε τέτοια πράγματα στην «καλή θέληση» (και ευφυΐα) του μεταγλωττιστή. Στο παράδειγμά μας το κείμενο δεν έχει ούτε 100 χαρακτήρες· άρα μικρό το κακό. Σε «πραγματικές συνθήκες» όμως…◀

Οι εντολές:

```
   cout << text1.find( "του" ) << endl;
   cout << text1.find( "του", 40 ) << endl;
```

θα δώσουν:

```
39
110
```

Δηλαδή: η λέξη "`του`" υπάρχει για πρώτη φορά στη θέση με δείκτη 39. *Ξεκινώντας την αναζήτηση από τη θέση* με δείκτη 40, η λέξη "`του`" *βρίσκεται για πρώτη φορά* με δείκτη 110.

Ας πάρουμε το κείμενο:

Νῦν ἡ ταπείνωση τῶν Θεῶν Νῦν ἡ σποδὸς τοῦ Ἀνθρώπου
Νῦν Νῦν τὸ μηδὲν

καὶ Αἰὲν ὁ κόσμος ὁ μικρὸς, ὁ Μέγας!

που το αποθηκεύουμε στη μεταβλητή:

```
   string text2;
// . . .
   text2 = "Νυν η ταπείνωση των θεών Νυν η σποδός του Ανθρώπου\n"
           "Νυν Νυν το μηδέν\n\n"
           "και Αιέν ο κόσμος ο μικρός, ο Μέγας!";
```

Αν έχουμε δηλώσει:

```
int ndx;
```

οι παρακάτω εντολές μας δίνουν όλους τους δείκτες όλων των θέσεων που αρχίζει η λέξη "Νυν":

```
ndx = text2.find( "Νυν" );
while ( 0 <= ndx && ndx < text2.length() )
{
   cout << ndx << "  ";
   ndx = text2.find( "Νυν", ndx+1 );
}
cout << endl;
```

Αποτέλεσμα:

```
0  25  51  55
```

Μια άλλη μέθοδος, η *rfind()*, είναι όμοια με τη *find()*, με μόνη διαφορά ότι ψάχνει *από δεξιά προς τα αριστερά* (από το τέλος προς την αρχή). Έτσι, για τα στοιχεία του παραπάνω παραδείγματος, οι εντολές:

```
cout << text1.rfind("του") << endl;
cout << text1.rfind("του", 108) << endl;
```

θα δώσουν:

```
110
39
```

Δες τώρα ένα άλλο πρόβλημα αναζήτησης: Ας πούμε ότι έχουμε την *text2* με την τιμή που έχει στο παράδειγμα και θέλουμε να βρούμε το πρώτο τονούμενο πεζό φωνήεν στο κείμενο. Μπορούμε να δώσουμε το εξής:

```
cout << text2.find_first_of("άέήίΐόύΰώ") << endl;
```

που θα μας δώσει:

```
10
```

Πράγματι, στη θέση με δείκτη 10 του κειμένου μας υπάρχει το 'ΐ' (ταπείνωση). Μπορούμε να συνεχίσουμε την αναζήτηση:

```
cout << text2.find_first_of("άέήίΐόύΰώ", 11) << endl;
```

Αποτέλεσμα:

```
22
```

Στη θέση με δείκτη 22 του κειμένου μας υπάρχει το 'ώ' (Θεών).

Δηλαδή, η μέθοδος *find_first_of()* μας δίνει *τον δείκτη της πρώτης θέσης όπου υπάρχει χαρακτήρας από κάποιο σύνολο*, που το δίνουμε *ως πρώτο όρισμα σε μορφήν ορμαθού*. Αν θέλουμε η αναζήτηση να μην ξεκινήσει από την αρχή, δίνουμε και δεύτερο όρισμα με τον δείκτη της επιθυμητή θέσης εκκίνησης.

Ό,τι είναι η *rfind()* για τη *find()* είναι η *find_last_of()* για τη *find_first_of()*: ψάχνει για τον τελευταίο χαρακτήρα που να ανήκει σε κάποιο σύνολο ή, αλλιώς, ψάχνει από δεξιά προς τα αριστερά.

Παρόμοια δουλειά κάνει και η *find_first_not_of()* η οποία ψάχνει για τον πρώτο χαρακτήρα που *δεν ανήκει* στο σύνολο-όρισμα. Η αντίστοιχη μέθοδος που ψάχνει από το τέλος είναι η *find_last_not_of()*.

Παρατήρηση: ▶

Θα πεις: «Καλά αυτά, αλλά ο Ποιητής γράφει πολυτονικό και εδώ κατακρεουργήθηκε!» και θα έχεις δίκιο! Αυτό όμως έγινε για να δείξουμε αυτά που θέλουμε με απλό και γρήγορο τρόπο. Υπάρχει ένας άλλος τύπος, ο *wstring*, ακριβώς σαν τον *string* αλλά με βάση τον `wchar_t`. Με αυτόν τον τύπο μπορείς να χειριστείς όλα τα σύμβολα του Unicode, επομένως και το πολυτονικό μας.◀

10.9 Αντικατάσταση – Διαγραφή – Εισαγωγή

Μια άλλη δουλειά, πολύ χρήσιμη, που την ξέρεις και αυτήν από τον κειμενογράφο σου (**replace** ή **αντικατάστησε**), είναι η αντικατάσταση ενός ορμαθού από έναν άλλον. Για μεταβλητές τύπου `string` μπορούσε να την επιτύχουμε με τη μέθοδο *replace()*. Δες ένα παράδειγμα.

Ας πούμε ότι έχουμε:

```
string target( "ενός" ), text1;
text1 ="Ένα πολύ συνηθισμένο πρόβλημα επεξεργασίας κειμένου\n"
       "είναι η αναζήτηση ενός χαρακτήρα ή μιας λέξης ή\n")
       "ενός κομματιού κειμένου μέσα σε κάποιο άλλο κείμενο.";
```

και δίνουμε τις εντολές:

```
cout << text1 << endl << endl;
text1.replace( text1.find(target), target.length(), "ΚΑΠΟΙΟΥ");
cout << text1 << endl;
```

Αποτέλεσμα:

```
Ένα πολύ συνηθισμένο πρόβλημα επεξεργασίας κειμένου
είναι η αναζήτηση ενός χαρακτήρα ή μιας λέξης ή
ενός κομματιού κειμένου μέσα σε κάποιο άλλο κείμενο.

Ένα πολύ συνηθισμένο πρόβλημα επεξεργασίας κειμένου
είναι η αναζήτηση ΚΑΠΟΙΟΥ χαρακτήρα ή μιας λέξης ή
ενός κομματιού κειμένου μέσα σε κάποιο άλλο κείμενο.
```

Οι παράμετροι της `replace` στο παράδειγμά μας είναι οι εξής:

- Η πρώτη είναι ένας φυσικός που μας δίνει τον δείκτη της θέσης από την οποίαν ξεκινάει το προς αντικατάσταση κείμενο. Στην περίπτωσή μας βάλαμε τη θέση της τιμής του *text*1 που αρχίζει η τιμή της μεταβλητής *target* (δηλαδή τη λέξη "`ενός`") για πρώτη φορά (`text1.find(target)`).
- Η δεύτερη είναι και πάλι ένας φυσικός που δίνει το *μήκος* του κειμένου που θα αντικατασταθεί. Στην περίπτωσή μας βάλαμε το μήκος της τιμής της μεταβλητής *target* (δηλαδή της λέξης "`ενός`" που είναι 4).
- Η τρίτη παράμετρος είναι ο ορμαθός με τον οποίον θα γίνει η αντικατάσταση. Μπορεί να είναι ακόμη: τιμή τύπου `char` ή πίνακας χαρακτήρων ή μεταβλητή τύπου *string*.

Όπως βλέπεις τα μήκη των δύο ορμαθών ("`ενός`" και "`ΚΑΠΟΙΟΥ`") δεν είναι απαραίτητο να είναι ίσα. Αυτό φυσικά έχει ως συνέπεια να αλλάξει και το μήκος του κειμένου μας.

Αν θέλουμε να αντικαταστήσουμε όλες τις λέξεις "`ενός`" που θα βρούμε, δουλεύουμε όπως στην πολλαπλή *find*:[7]

```
ndx = text1.find( target );
while ( 0 <= ndx && ndx < text1.length() )
{
   text1.replace( ndx, target.length(), "ΚΑΠΟΙΟΥ" );
   ndx = text1.find( target, ndx+1 );
}
```

και παίρνουμε:

```
Ένα πολύ συνηθισμένο πρόβλημα επεξεργασίας κειμένου
είναι η αναζήτηση ΚΑΠΟΙΟΥ χαρακτήρα ή μιας λέξης ή
ΚΑΠΟΙΟΥ κομματιού κειμένου μέσα σε κάποιο άλλο κείμενο.
```

Με τη *replace()* μπορείς να κάνεις και *διαγραφή*. Αν δώσεις, π.χ.:

```
text1.replace( text1.find(target), target.length(), "" );
```

ζητάς να αντικατασταθεί η τιμή της *target*, την πρώτη φορά που θα βρεθεί στην τιμή της *text*1, με τον κενό ορμαθό, δηλ. να διαγραφεί.

---

[7] Το "`ndx+1`" θα δουλέψει συνήθως. Γενικώς, είναι λάθος. Το σωστό είναι "`ndx+μήκος("ΚΑΠΟΙΟΥ")`". Σκέψου το λιγάκι!

Υπάρχει όμως και μέθοδος *erase()* που κάνει διαγραφές. Στη συνήθη της μορφή παίρνει δύο ορίσματα: το πρώτο είναι ο δείκτης της θέσης από την οποίαν αρχίζει η διαγραφή και το δεύτερο είναι το πλήθος των χαρακτήρων που θα διαγραφούν. Μπορούμε λοιπόν να γράψουμε:

```
text1.erase( text1.find(target), target.length() );
```

Φυσικά, εκτός από αντικατάσταση και διαγραφή μπορείς να κάνεις και εισαγωγή με τη μέθοδο *insert()*. Είναι σαν την *append()* με τη διαφορά ότι:

- με την *append()* η επισύναψη γίνεται πάντοτε στο τέλος, ενώ
- με την *insert()* η εισαγωγή γίνεται όπου θέλουμε· το καθορίζουμε με την τιμή που δίνουμε στην πρώτη παράμετρο.

Ας ξαναδούμε το παράδειγμα που δώσαμε για την *append()*, αλλά κάπως πιο «ανακατεμένο». Είχαμε:

```
string s1( "πέντε" ), s2( "στο χέρι παρά δέκα" ), s3;
```

και δίνουμε:

```
s3 = " και καρτέρει";      cout << s3 << endl;
s3.insert( 0, s1 );        cout << s3 << endl;
s3.insert( 5, s2 );        cout << s3 << endl;
s3.insert( 0, "κάλλιο " ); cout << s3 << endl;
s3.insert( 12, " και " );  cout << s3 << endl;
```

Αποτέλεσμα:

```
 και καρτέρει
πέντε και καρτέρει
πέντεστο χέρι παρά δέκα και καρτέρει
κάλλιο πέντεστο χέρι παρά δέκα και καρτέρει
κάλλιο πέντε και στο χέρι παρά δέκα και καρτέρει
```

10.10 Διαχείριση Χαρακτήρων

Όπως στους πίνακες, έτσι και στις μεταβλητές τύπου *string*, μπορείς να διαχειρίζεσαι κάθε χαρακτήρα βάζοντας μετά από το όνομα της μεταβλητής τον κατάλληλο δείκτη. Αν έχεις δηλώσει:

```
string s3;
```

μπορείς να γράψεις για παράδειγμα:

```
if ( s3[0] == 'κ' )  s3[0] = 'K';
```

Βέβαια, χρειάζεται προσοχή: είναι δική σου ευθύνη να διασφαλίσεις ότι ο δείκτης θα παίρνει τιμές από 0 μέχρι `s3.length()-1`.

Αυτά τα πράγματα σου θυμίζουν πολύ τους πίνακες· αλλά η ομοιότητα σταματάει εδώ: Μια τιμή τύπου *string* κρύβει σίγουρα κάποιον πίνακα με τιμές τύπου `char` αλλά δεν είναι μόνον αυτό.

Εκτός από τη χρήση δείκτη, υπάρχει και άλλος τρόπος διαχείρισης των χαρακτήρων μιας μεταβλητής τύπου *string*: η μέθοδος `at`. Αυτό που γράψαμε παραπάνω μπορεί να γραφεί και ως εξής:

```
if ( s3.at(0) == 'κ' )  s3.at( 0 ) = 'K';
```

Αυτός ο τρόπος είναι ασφαλέστερος από αυτόν με τον δείκτη, διότι αν κάνεις λάθος και βγεις έξω από τα όρια της μεταβλητής σου θα ειδοποιηθείς, όπως θα μάθουμε αργότερα.

Παράδειγμα

Το παρακάτω πρόγραμμα, αντιστρέφει ένα κείμενο, που δίνεται ως στοιχείο εισόδου. Αν η αντίστροφη γραφή συμπίπτει με την ορθή, το πρόγραμμα εκφράζει τον ενθουσιασμό του! Στο πρόγραμμα χρησιμοποιούνται οι μέθοδοι:

- *at()*, για τη διαχείριση τιμής μιας μεταβλητής *string*, χαρακτήρα προς χαρακτήρα (`strIn.at(k)`)· για την ίδια δουλειά χρησιμοποιείται και η
- *length()*, διότι θέλουμε να επεξεργαστούμε όλους τους χαρακτήρες, από το τέλος προς την αρχή (`for (k = strIn.length()-1; k>=0; k=k-1)`),
- *compare()*, για τη σύγκριση τιμών (`strIn.compare(strOut) == 0`) δύο μεταβλητών τύπου `string`· σύγκριση όμως γίνεται και με τον τελεστή `!=` (`strIn != "T" && strIn != "t" && ...`),
- *append()*, για να κτίσουμε την τιμή μιας μεταβλητής τύπου *string*, χαρακτήρα προς χαρακτήρα. Για αυτήν συζητούμε και παρακάτω.

Πώς δουλεύει το πρόγραμμα; Αφού πάρουμε στη *strIn* το κείμενο που θα χειριστούμε αντιγράφουμε στη *strOut* έναν προς ένα τους χαρακτήρες της *strIn* αλλά από το τέλος προς την αρχή. Αυτό γίνεται με τη

```
      for ( k = strIn.length()-1; k >= 0; k = k-1 )
```

Έτσι αποκλείεται να βγούμε έξω από όρια και μπορούμε να χρησιμοποιήσουμε είτε τον δείκτη (`strIn[k]`) είτε την *at* (`strIn.at(k)`). Ποιο είναι το πρόβλημα με την *append*; Και οι δύο τρόποι μας δίνουν έναν χαρακτήρα και η *append* δεν δέχεται τέτοια παράμετρο. Και ποια λύση δίνουμε; Δηλώνουμε μια μεταβλητή

```
   string str1( "x" );
```

Το 'x' »κρατάει τη θέση» (place holder) για να έχουμε ορμαθό με μήκος 1. Στη συνέχεια το αντικαθιστούμε με κάθε χαρακτήρα που παίρνουμε από τη *strIn*:

```
         str1[0] = strIn.at(k);
```

Φυσικά αυτή δεν είναι η μοναδική λύση. Στην επόμενη παράγραφο θα δούμε μια καλύτερη.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   const string msg( "ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): " );

   int k;
   string strIn, strOut, str1( "x" );

   cout << msg;  getline( cin, strIn, '\n' );
   while ( strIn != "T" && strIn != "t" &&
           strIn != "Τ" && strIn != "τ" )
   {
      strOut = "";    // κενό
      for ( k = strIn.length()-1; k >= 0; k=k-1 )
      {
         str1[0] = strIn.at( k );
         strOut.append( str1 );
      }
      cout << " Η ανάποδη γραφή της: " << strIn
           << " είναι: " << endl;
      cout << strOut << endl;
      if ( strIn == strOut )
         cout << " Καλό αυτό!" << endl;
      cout << endl;
      cout << msg;  getline( cin, strIn, '\n' );
   } // while
   cout << " ΤΕΛΟΣ" << endl;
} // main
```

Να ένα παράδειγμα διαλόγου με το πρόγραμμα αυτό. Με κεκλιμένα οι απαντήσεις του χρήστη:

```
ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): ABCDEFGHJIK
 Η ανάποδη γραφή της: ABCDEFGHJIK είναι:
```

```
KIJHGFEDCBA

ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΜΗ ΜΟΝΑΝ ΟΨΙΝ
 Η ανάποδη γραφή της: ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΝΗ ΜΟΝΑΝ ΟΨΙΝ είναι:
ΝΙΨΟ ΝΑΝΟΜ ΗΝ ΑΤΑΜΗΜΟΝΑ ΝΟΨΙΝ

ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): ΝΙΨΟΝΑΝΟΜΗΜΑΤΑΜΗΜΟΝΑΝΟΨΙΝ
 Η ανάποδη γραφή της: ΝΙΨΟΝΑΝΟΜΗΜΑΤΑΜΗΜΟΝΑΝΟΨΙΝ είναι:
ΝΙΨΟΝΑΝΟΜΗΜΑΤΑΜΗΜΟΝΑΝΟΨΙΝ
 Καλό αυτό!

ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): MADAM IN EDEN I'M ADAM
 Η ανάποδη γραφή της: MADAM IN EDEN I'M ADAM είναι:
MADA M'I NEDE NI MADAM

ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): MADAMINEDENIMADAM
 Η ανάποδη γραφή της: MADAMINEDENIMADAM είναι:
MADAMINEDENIMADAM
 Καλό αυτό!

ΔΩΣΕ ΜΙΑ ΦΡΑΣΗ (Τ ΓΙΑ ΤΕΛΟΣ): Τ
 ΤΕΛΟΣ
```

10.11 Υποορμαθοί

Μερικές φορές θέλουμε να πάρουμε έναν **υποορμαθό** (substring), ένα κομμάτι ενός ορμαθού, για να το χειριστούμε χωριστά. Η μέθοδος *substr()* μας δίνει αυτήν τη δυνατότητα: μας δίνει μια νέα τιμή τύπου *string* με περιεχόμενο έναν υποορμαθό κάποιας άλλης τιμής του ίδιου τύπου.

Αν έχουμε μια μεταβλητή *s1* τύπου *string*, μπορούμε να πούμε ότι ένας υποορμαθός της ορίζεται από δύο φυσικούς αριθμούς:

- τον *δείκτη της θέσης* που αρχίζει και
- το *μήκος* που έχει.

Αυτές ακριβώς τις παραμέτρους περιμένει και η *substr()*. Αν:

```
string s1 = "ένας ορμαθός της C++", s2;

s2 = s1.substr( 5, 7 );
cout << s2 << endl;
```

θα πάρουμε:

```
ορμαθός
```

Μπορούμε να βάλουμε μόνο μία παράμετρο, τη θέση, οπότε το τέλος του υποορμαθού είναι το τέλος του αρχικού:

```
s2 = s1.substr( 5 );
cout << s2 << endl;
```

Αποτέλεσμα:

```
ορμαθός της C++
```

Παράδειγμα 1

Στο παράδειγμα της προηγούμενης παραγράφου, η αντιγραφή από τη *strIn* στη *strOut* μπορεί να γίνει ως εξής:

```
     strOut = "";    // κενό
     for ( k = strIn.length()-1; k >= 0; k = k-1 )
        strOut.append( strIn.substr(k,1) );
```

Παράδειγμα 2 ✍

Έχοντας δηλώσει:

```
string s1, s2, s3;
```

μετά την εκτέλεση των παρακάτω εντολών:

```
s1 = "ΦΑΣΟΥΛΙ ΦΑΣΟΥΛΙ ΓΕΜΙΖΕΙ ΤΟ ΣΑΚΟΥΛΙ";
s2 = s1.substr( 8, 8 );
s3 = s1.substr( s1.find('Τ'), 10 );
cout << s2 << s3 << endl;
s2 = s1.substr( 20, 20 );
cout << s2 << endl;
```

θα τυπωθούν τα ακόλουθα:

```
ΦΑΣΟΥΛΙ ΤΟ ΣΑΚΟΥΛΙ
ΖΕΙ ΤΟ ΣΑΚΟΥΛΙ
```

Από τη δεύτερη γραμμή βλέπεις ότι εσύ μπορεί να παραβείς τους περιορισμούς μήκους, αλλά η `substr` δεν θα υπερβεί τα όρια της αρχικής τιμής.
✍✍✍

10.12 Ρεύματα Από και Προς *string*

Μέσα στις πολλές δυνατότητες διαχείρισης εισόδου/εξόδου στοιχείων η C++ δίνει και τύπους **ρευμάτων από και προς *string*** (string streams). Έτσι, έχουμε τη δυνατότητα:

- Να παίρνουμε τα στοιχεία εισόδου –όπως και αν είναι αυτά– να κάνουμε ελέγχους εγκυρότητας και να γνωστοποιήσουμε στον χρήστη τυχόν προβλήματα ώστε να τα διορθώσει.
- Να ετοιμάζουμε (μορφοποιούμε) τα στοιχεία εξόδου όπως θέλουμε πριν τα βγάλουμε στην οθόνη ή σε κάποιο αρχείο text.

Αν βάλεις στο πρόγραμμά σου την οδηγία "`#include <sstream>`" σου δίνεται η δυνατότητα να δηλώσεις:

```
istringstream ssin( aString );
ostringstream ssout;
```

Το *ssin* είναι ένα ρεύμα σαν αυτά που ξέρουμε μόνο που δεν διαβάζει ούτε το πληκτρολόγιο ούτε κάποιο αρχείο αλλά το

```
string aString( "17 -375 145.78 31e4" );
```

Να ένα πρόγραμμα από τα παλιά (§2.3) κάπως αλλαγμένο:

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main()
{
   int    k, j;
   double x, y, t;
   string aString( "17 -375 145.78 31e4" ), outString;
   istringstream ssin( aString );
   ostringstream ssout;

   ssin >> k >> j >> x >> y;
   ssout << k << ' ' << j << ' ' << x << ' ' << y;
   outString = ssout.str();
   cout << outString << endl;
}
```

Με την:

```
ssin >> k >> j >> x >> y;
```

διαβάζονται οι 17, -375, 145.78, 31*e*4 ως τιμές των *k*, *j*, *x*, *y* αντιστοίχως. Αν μετά από αυτήν προσπαθήσεις να διαβάσεις (`ssin >> t`) δεν θα τα καταφέρεις. Αν ελέγξεις την *ssin*`.eof()` θα τη βρεις `true`.

Με την

```
ssout << k << ' ' << j << ' ' << x << ' ' << y;
```

οι τιμές των μεταβλητών γράφονται σε έναν ενταμιευτή τύπου *string* που έχει το *ssout*. Αυτόν τον ενταμιευτή τον παίρνουμε με την `ssout.str()` και τον αντιγράφουμε σε μια άλλη μεταβλητή τύπου *string*. Φυσικά θα μπορούσαμε να γράψουμε κατ' ευθείαν:

```
cout << ssout.str() << endl;
```

Τα ρεύματα προς/από *string* έχουν τις ίδιες ιδιότητες (και δυνατότητες) με τα ρεύματα προς/από αρχεία. Επιπλέον όμως έχουν τη μέθοδο *str* για τον χειρισμό του ενταμιευτή:

- Με τις `ssin.str(aString)` και `ssout.str(aString)` βάζουμε ως ενταμιευτή την (τιμή που έχει εκείνη τη στιγμή η) *aString*. Για το *ssout* αυτό έχει μικρή σημασία αφού, με το πρώτο γράψιμο, ο ενταμιευτής θα «καθαριστεί».
- Με τις `as = ssin.str()` και `as = ssout.str()` αντιγράφουμε στην *as* (τύπου *string*) την τιμή του ενταμιευτή (εκείνη τη στιγμή). Στην περίπτωση αυτήν η πρώτη έχει μικρή χρησιμότητα αφού την έχουμε βάλει εμείς.

  Στη συνέχεια θα ασχοληθούμε με τον χειρισμό «αριθμητικών» ορμαθών.

10.12.1 Ορμαθοί και Αριθμοί

Οι πεπειραμένοι προγραμματιστές δεν γράφουν εντολές που να διαβάζουν αριθμητικά δεδομένα παρά μόνον στην περίπτωση που διαβάζουν αρχείο text που έχει ήδη ελεγχθεί. Συνήθως διαβάζουν ορμαθούς χαρακτήρων και στη συνέχεια τους μετατρέπουν σε αριθμητικές τιμές ελέγχοντας τυχόν λάθη. Ποιο είναι το κέρδος;

- Διαβάζοντας αριθμητικές τιμές μπορεί να συναντήσεις λάθη, π.χ. "`,`" αντί για "`.`" στην υποδιαστολή, ελληνικό έψιλον αντί για "`e`" στον εκθέτη και άλλα παρόμοια. Όπως ήδη ξέρεις, αυτά τα προβλήματα «διαταράζουν» (μπορεί και να διακόψουν) την ομαλή εκτέλεση του προγράμματός σου.
- 'Οταν διαβάζεις χαρακτήρες τα πάντα είναι δεκτά! Ο έλεγχος εγκυρότητας γίνεται από το πρόγραμμα και οι χειρισμοί λαθών είναι ευθύνη του προγραμματιστή.

Τα ρεύματα από τιμές *string* είναι ένα καλό εργαλείο για τη δουλειά αυτή.

Ας πούμε ότι διαβάζεις έναν ορμαθό χαρακτήρων στη μεταβλητή

```
   string aString;
// . . .
   getline( cin, aString, '\n' );
```

Έχεις ζητήσει από τον χρήστη να πληκτρολογήσει έναν ακέραιο –που θέλεις να αποθηκεύσεις στη μεταβλητή (`int`) *k*– αλλά αντί για "`7`" σου πληκτρολόγησε "`&`". Αυτή είναι και η τιμή της *aString*. Για να περάσουμε από την *aString* στην *k* θα χρειαστούμε ένα ρεύμα:

```
istringstream ssin;
```

στο οποίο θα βάλουμε ως ενταμιευτή την *aString*:

```
ssin.str( aString );
```

και από αυτό θα αποπειραθούμε να διαβάσουμε:

```
ssin >> k;
if ( ssin.fail() )  // αποτυχία
```

Αμέσως μετά ελέγχουμε, με τη γνωστή μας *fail*, αν τα καταφέρουμε.

Στην περίπτωσή μας θα μας πει ότι αποτύχαμε και θα πρέπει να δώσουμε το κατάλληλο μήνυμα προς τον χρήστη.

Ας πούμε τώρα ότι ο χρήστης πληκτρολόγησε "**1&**" και αυτή είναι η τιμή της *aString*. Στην περίπτωση αυτήν η **ssin.fail()** θα δώσει **false**, δηλαδή: όλα πήγαν καλά! Αλλά η *k* θα πάρει τιμή 1. Εδώ πώς πιάνουμε το λάθος; Προσπαθούμε να διαβάσουμε 1 χαρακτήρα. Αν τα καταφέρουμε θα πει ότι η τιμή της *k* δεν έχει προκύψει από ολόκληρη την τιμή της *aString*. Άρα έχουμε πρόβλημα. Αν δεν τα καταφέρουμε και πάρουμε **ssin.eof()** θα πει ότι όλα πήγαν καλά.

Ας τα δώσουμε όλα μαζί: Πρώτα οι δηλώσεις

```
   istringstream ssin;
   int    k;
   string aString;
   char   c;
```

Και μετά η ανάγνωση και οι έλεγχοι:

```
   getline( cin, aString, '\n' );
   ssin.str( aString );
   ssin >> k;
   if ( ssin.fail() )
   // λάθος
   else // κάτι διάβασε
   {
      ssin >> c;
      if ( ssin.eof() ) // OK, δεν έχει άλλα, τα διάβασε όλα
         cout << k << endl;
      else
      // λάθος
   }
```

Με παρόμοιο τρόπο διαβάζουμε και πραγματικές τιμές.

Στη συνέχεια δίνουμε ένα παράδειγμα για να δούμε τη χρήση αυτών που είπαμε στα προγράμματά μας.

Παράδειγμα

Ας πάρουμε το πρόγραμμα της ελεύθερης πτώσης, που είδαμε στην §2.12, και ας κάνουμε την ανάγνωση του ύψους μέσω ορμαθού χαρακτήρων:

```
#include <iostream>
#include <cmath>
#include <string>
#include <sstream>
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
   string s;
   istringstream ssin;
   bool done( false );
   char c;

// Διάβασε το h
   while ( !done || h < 0 )
   {
      cout << " Δώσε μου το αρχικό ύψος (h >= 0) σε m: ";
      getline( cin, s, '\n' );
      ssin.clear();
      ssin.str( s );   ssin >> h;
      if ( ssin.fail() )
         cout << "απαράδεκτοι χαρακτήρες (" << s << ")" << endl;
      else
      {
         ssin >> c;
         if ( ssin.eof() )
```

```
            done = true;
         else
            cout << "απαράδεκτος χαρακτήρας (\'" << c << "\')"
                 << endl;
      } // if ( !ssin.fail
   } // while

// (g == 9.81) && (0 <= h <= DBL_MAX)
// Υπολόγισε τα tP, vP
// ΤΑ ΥΠΟΛΟΙΠΑ ΙΔΙΑ
```

Όπως βλέπεις, αντί για την `cin >> h` έχουμε βάλει:

```
getline( cin, s, '\n' );
ssin.clear();
ssin.str( s );   ssin >> h;
```

και μετά έρχονται οι έλεγχοι. Πρόσεξε την "`ssin.clear()`": αν, μετά το λάθος που βρίσκουμε όταν προσπαθήσει να διαβάσει το "`*)`", δεν δώσουμε `ssin.clear()`, στην επόμενη προσπάθεια ανάγνωσης, θα βγάλει το ίδιο λάθος ακόμη και αν τα κάνουμε όλα σωστά. Όλα αυτά τα βάλαμε σε μια **while** από την οποίαν βγαίνει μόνον αν πάρει σωστή τιμή (και μη αρνητική). Αυτό είναι κάπως ανελαστικό. Καλύτερα θα ήταν αν δίναμε και κάποια διαφυγή αν ο χρήστης μετανιώσει και θέλει να τα παρατήσει (π.χ. να πιέσει το <esc>).

Στη συνέχεια βλέπεις ένα παράδειγμα εκτέλεσης:

```
 Δώσε μου το αρχικό ύψος (h >= 0) σε m: *)
απαράδεκτοι χαρακτήρες (*))
 Δώσε μου το αρχικό ύψος (h >= 0) σε m: 8o
απαράδεκτος χαρακτήρας ('o')
 Δώσε μου το αρχικό ύψος (h >= 0) σε m:  -80
 Δώσε μου το αρχικό ύψος (h >= 0) σε m:  80
 Αρχικό ύψος = 80 m
 Χρόνος Πτώσης = 4.03855 sec
 Ταχύτητα τη Στιγμή της Πρόσκρουσης = -39.6182 m/sec
```

Το αντίστροφο, η μετατροπή μιας αριθμητικής τιμής σε *string*, είναι πολύ απλή. Στο παρακάτω πρόγραμμα δίνουμε μερικά παραδείγματα από το Κεφ. 1 κάπως αλλαγμένα:

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{
   ostringstream sout;
   string aString;

   sout.precision( 8 );
   sout << 1234.5 << "  " << 123456.78 << "  "
        << 123456789.0123 << endl;

   sout.setf( ios_base::scientific, ios_base::floatfield );
   sout.precision(8);
   sout << 1234.5 << "  " << 123456.78 << "  "
        << 123456789.0123 << endl;

   sout.setf( ios_base::fixed, ios_base::floatfield );
   sout.precision(8);
   sout << 1234.5 << "  " << 123456.78 << "  "
        << 123456789.0123 << endl;
   aString = sout.str();
   cout << aString << endl;
}
```

Εδώ, αντί για το *cout*, γράφουμε στο ρεύμα *sout*. Τελικώς, αντιγράφουμε τον ενταμιευτή του *sout* στη μεταβλητή *aString*. Να η τιμή της:

```
1234.5  123456.78  1.2345679e+008
1.23450000e+003  1.23456780e+005  1.23456789e+008
1234.50000000  123456.78000000  123456789.01230000
```

Από εδώ και πέρα μπορείς να χειριστείς την τιμή αυτή με όλα τα εργαλεία που έχεις για τον τύπο *string*.

10.13 Η Κληρονομιά της C: Πίνακες με Χαρακτήρες

Θα δούμε εν συντομία τη διαχείριση κειμένου από τη C διότι:

1. θα τη δεις σε πολλά προγράμματα C++ (οι προγραμματιστές δύσκολα εγκαταλείπουν αυτά που ξέρουν) και
2. είναι χρήσιμη ακόμη και όταν δουλεύεις με τον τύπο *string*.

Όπως είδαμε στην εισαγωγή του κεφαλαίου:

- αποθηκεύουμε κείμενα σε πίνακες με στοιχεία τύπου `char`,
- στο τέλος του κειμένου βάζουμε *απαραιτήτως* έναν `char(0)`· αυτό πρέπει να το παίρνουμε υπόψη μας όταν δηλώνουμε τον πίνακα.

Αν δηλώσεις:

```
char a[100] = "πάν μέτρον εκατό πόντοι";
```

ή

```
char a[] = "πάν μέτρον εκατό πόντοι";
```

ο `char(0)` εισάγεται αυτομάτως. Αν όμως δηλώσεις:

```
char a[] = { 'π','ά','ν',' ','μ','έ','τ','ρ','ο','ν',' ',
             'ε','κ','α','τ','ό',' ',
             'π','ό','ν','τ','ο','ι','\0' };
```

θα πρέπει να βάλεις το `char(0)` όπως βλέπεις εδώ.[8]

Είδαμε ακόμη ότι μπορείς να γράφεις στην οθόνη όπως ξέρουμε:

```
cout << a << endl;
```

και ότι μπορούμε να διαβάσουμε από το πληκτρολόγιο με τη *cin.getline()*. Αν

```
char a[100];
```

μπορείς να διαβάσεις κείμενο από το πληκτρολόγιο, ως τιμή του a, με την:

```
cin.getline( a, 100 );
```

Πάντως, η κλάση *istream* του *cin*, έχει και μια μορφή της *get()* με την οποία μπορείς να διαβάσεις κείμενο:

```
cin.get( a, 100 );
```

Και εδώ, το δεύτερο όρισμα, δείχνει το μέγιστο πλήθος χαρακτήρων που είναι δεκτοί. Και εδώ, στο μήκος περιλαμβάνεται και μια θέση για το `char(0)`. Αλλά, όταν δουλεύεις με τη *get* μην ξεχνάς να διαβάζεις και το «τέλος γραμμής» (<enter>, `'\n'`):

```
cin.get( a, 100 );  cin.get( ceol );
```

όπου η *ceol* είναι μεταβλητή τύπου `char`.

Στη συνέχεια θα δούμε μερικές από τις συναρτήσεις που έχει η C++ (από τη C) για να διαχειρίζεται πίνακες με κείμενα. Για να τις χρησιμοποιήσεις θα πρέπει να βάλεις στο πρόγραμμά σου:[9]

```
#include <string>
```

Μπορούμε να αλλάξουμε την τιμή της a με εντολή εκχώρησης σαν την:

---

[8] Το `'\0'` είναι που γράφεται συνήθως. Αν βάλεις `char(0)` ή απλώς `0` και πάλι είναι μια χαρά.

[9] Αν βάλεις "`#include <cstring>`" έχεις μόνον τις συναρτήσεις της C. Αν δεν βάλεις το "`c`" τα έχεις όλα.

```
a = "όποιος φυλάει τα ρούχα του είναι demodé";
```

Όχι! Για να αλλάξεις την τιμή της a χρειάζεται μια συνάρτηση της C++, η *strcpy()* (**str**ing **cpy**):

```
strcpy( a, "όποιος φυλάει τα ρούχα του είναι demodé" );
```

ή ακόμη:

```
strcpy( a, b );
```

αν ο b είναι ένας παρόμοιος πίνακας, π.χ.:

```
char b[] = "πάν μέτρον εκατό πόντοι";
```

Η *strncpy()* είναι σαν την *strcpy()*, αλλά έχει και τρίτη παράμετρο όπου περιμένει έναν φυσικό αριθμό που καθορίζει πόσοι χαρακτήρες θα αντιγραφούν από το δεύτερο όρισμα στο πρώτο. Θέλει λίγη προσοχή:

```
char a[50] = "abcdefghijklmnopq";

strncpy( a, "123456789", 15 );
cout << a << endl;
strcpy( a, "abcdefghijklmnopq" );
strncpy( a, "123456789", 5 );
cout << a << endl;
```

Αποτέλεσμα:

```
123456789
12345fghijklmnopq
```

Την πρώτη φορά ζητήσαμε να αντιγραφούν 15 χαρακτήρες ενώ το μήκος του δεύτερου ορίσματος είναι 9. Αντιγράφηκαν λοιπόν όλοι οι χαρακτήρες και το `char(0)` που σημειώνει το τέλος του ορμαθού· έτσι τιμή του a γίνεται το `123456789`. Τη δεύτερη φορά ζητήσαμε να αντιγραφούν 5 χαρακτήρες. Αυτοί αντικατάστησαν τους 5 πρώτους χαρακτήρες της τιμής του a. Αν θέλεις τιμή του a να γίνει το `12345` θα πρέπει να δώσεις:

```
strncpy ( a, "123456789", 5 );  a[5] = char( 0 );
```

Πολύ συχνά έχεις την εξής περίπτωση:

```
string cpps;
char   cs[ L ];
```

όπου *L* θετικός ακέραιος και θέλεις να αντιγράψεις στον cs την τιμή της *cpps* ή, εν πάσει περιπτώσει, «όσο χωράει». Αυτό γίνεται ως εξής:

```
strncpy ( cs, cpps.c_str(), L-1 );  cs[L-1] = char( 0 );
```

Δηλαδή: αντιγράφουμε *L* - 1 χαρακτήρες το πολύ και στην τελευταία θέση βάζουμε τον φρουρό ώστε ο cs να είναι διαχειρίσιμος. Αν το μήκος της τιμής της *cpps* είναι μικρότερο από *L* - 1 θα υπάρχει και άλλο `char(0)` πιο πριν και θα είναι ο πραγματικός φρουρός.

Μπορείς να συγκρίνεις δύο πίνακες με κείμενα, με τη *strcmp()*, που είναι σαν την `compare`. Η `strcmp(a, b)` επιστρέφει ακέραιη

- αρνητική τιμή *αν* η τιμή του a προηγείται λεξικογραφικώς της τιμής του b,
- *θετική* τιμή *αν η τιμή του* a *έπεται λεξικογραφικώς της τιμής του* b,
- *μηδέν* (0) *αν οι τιμές των* a *και* b *είναι ίσες*.

Η *strncmp* είναι σαν την *strcmp* αλλά παίρνει και τρίτο όρισμα, έναν φυσικό αριθμό, που μας λέει πόσους χαρακτήρες να συγκρίνει. Οι:

```
char a[50] = "abcdefghijklmnopq", b[] = "abcdrstuvwxyz";
cout << strcmp(a, b) << "  " << strncmp(a, b, 3) << endl;
```

δίνουν:

```
-13  0
```

Δηλαδή: η τιμή του a προηγείται λεξικογραφικώς της τιμής του b (strcmp(a, b) `==` -13 < 0), αλλά οι τρεις πρώτοι χαρακτήρες τους είναι ίδιοι (*strncmp*`(`a, b, 3`) ==` 0).

Πολύτιμη είναι και η *stricmp()* είναι σαν την *strcmp()* αλλά βλέπει κεφαλαία και πεζά (λατινικά) γράμματα ίσα. Οι:

```
char a[] = "firstName", b[] = "FirstName", c[] = "firstname";
cout << strcmp(a, b) << "  " << strcmp(a, c) << endl;
cout << stricmp(a, b) << "  " << stricmp(a, c) << endl;
```

δίνουν:

```
1  -1
0  0
```

Η *strnicmp()* είναι για την *stricmp()* ό,τι είναι η *strncmp* για την *strcmp*.

Η *strlen()* (**str**ing **len**gth), επιστρέφει τιμή τύπου `size_t` που είναι το μήκος του ορμαθού που έχουμε αποθηκεύσει, χωρίς το τελικό `char(0)`: οι

```
char a[100], b[] = "πάν μέτρον εκατό πόντοι";

strcpy( a, "όποιος φυλάει τα ρούχα του είναι demodé" );
cout << strlen(a) << "  " << strlen(b) << endl;
cout << strlen( "μέτρον άριστον" ) << endl;
```

δίνουν:

```
39  23
14
```

Ο ορμαθός a είναι κενός αν:

- είναι ίσος με `""` (`strcmp(a, "") == 0`) ή
- έχει μήκος μηδέν (`strlen(a) == 0`) ή
- έχει `char(0)` στην αρχή του (`a[0] == char(0)`).

Μπορείς να χρησιμοποιείς οποιαδήποτε από αυτές τις συνθήκες στα προγράμματά σου.

Η `strcat(a, b)` κάνει το ίδιο πράγμα με την `a.append(b)`: επισυνάπτει στο τέλος της τιμής του a την τιμή του b. Δες το παρακάτω:

```
char s1[] = "πέντε", s2[] = "στο χέρι παρά δέκα", s3[50];

strcpy( s3, "κάλλιο " );
strcat( s3, s1 );  strcat( s3, " και " );  strcat( s3, s2 );
strcat( s3, " και καρτέρει" );
cout << s3 << endl;
```

Αποτέλεσμα:

```
κάλλιο πέντε και στο χέρι παρά δέκα και καρτέρει
```

Υπάρχει και η *strncat()*, που παίρνει και τρίτο όρισμα, έναν φυσικό αριθμό, που λέει πόσους χαρακτήρες από το δεύτερο όρισμα θα επισυναφθούν στο τέλος του πρώτου.

Αυτά τα λίγα δεν εξαντλούν τα εργαλεία της C++ για τα "C-style strings". Μερικά ακόμη υπάρχουν στην επόμενη παράγραφο. Θα πρέπει όμως να τονίσουμε ότι τα δίνουμε πιο πολύ για να μη σου είναι άγνωστα αν τα δεις σε κάποιο πρόγραμμα. Στα προγράμματα που θα γράφεις εσύ θα πρέπει να δουλεύεις με την κλάση *string*, που είναι σαφώς πιο εύχρηστη.

10.13.1 Ορμαθοί C και Αριθμοί

Και η C σου δίνει δυνατότητα να γράφεις σε (και να διαβάζεις από) πίνακες χαρακτήρων. Ειδικώς για μετατροπές ορμαθών σε αριθμούς και αντιστρόφως σου δίνει ορισμένες συναρτήσεις που θα δούμε εδώ αλλά κι αργότερα. Οι δηλώσεις για τα εργαλεία αυτά υπάρχουν στο `cstdlib` και θα πρέπει να το περιλάβεις στο πρόγραμμά σου, όταν τα χρησιμοποιείς.

Το πιο απλά εργαλεία είναι οι συναρτήσεις *atof()* (**a**lphanumeric **to** **f**loat), *atoi()* (**a**lphanumeric **to** **i**nt) και *atoll()* (**a**lphanumeric **to** **l**ong). Αυτές παίρνουν έναν ορμαθό χαρακτήρων (ή έναν πίνακα με τέτοια τιμή) και μας δίνουν μια τιμή τύπου `double`, η πρώτη, τύπου `int` η δεύτερη και τύπου `long` η τρίτη. Π.χ. οι εντολές:

```
#include <iostream>
```

```
#include <cstdlib>
using namespace std;
int main()
{
   char s1[] = "12345", s2[] = "1.23456789e10",
        s3[] = "1.23ab45678";
   long l;
   double d;

   l = atol( s1 );   d = atof( s2 );
   cout << l << "  " << d << endl;
```

θα δώσουν:

```
12345  1.23457e+10
```

Η μετάφραση από χαρακτήρες σε αριθμό σταματάει μόλις βρεθεί παράνομος χαρακτήρας. Π.χ. οι:

```
   l = atol( s3 );   d = atof( s3 );
   cout << l << "  " << d << endl;
```

θα δώσουν:

```
1  1.23
```

Τι έγινε εδώ; Η *atoll()*, προσπαθώντας να διαβάσει έναν ακέραιο από τον ορμαθό `1.23ab45678`, και αφού δεχτεί το `1`, βρίσκει την `'.'`, που δεν είναι δεκτή σε μια σταθερά τύπου `long`. Έτσι, μας επιστρέφει τιμή `1`. Για την *atof()* είναι δεκτοί οι χαρακτήρες μέχρι `1.23`· πρώτος παράνομος χαρακτήρας είναι ο `'a'`. Έτσι, μας επιστρέφει τιμή `1.23`.

Από αυτές τις συναρτήσεις δεν μπορείς να πάρεις παραπάνω πληροφορίες για το τι δεν πήγε καλά.

Πιο πλήρη δουλειά, σε περίπτωση λάθους, κάνουν οι *strtod()* και *strtol()* που θα δούμε αργότερα.

10.14 * Ο Τύπος *std*::*wstring*

Στην §4.7 είδαμε τον τύπο `wchar_t` που κάθε του τιμή αποθηκεύεται σε 16 δυαδικά ψηφία. Έτσι, όπως λέγαμε, «*έχει 65536 διαφορετικές τιμές, αντί για τις 256 του* `char` *και χρησιμοποιείται για την υλοποίηση του προτύπου Unicode.*» Και ακόμη «*Μια σταθερά τύπου* `wchar_t` *είναι μια σταθερά τύπου* `char` *με το πρόθεμα "*`L`*", π.χ.:*

`L'a'  L'%'  L'ξ'`»

Σε έναν πίνακα με στοιχεία τύπου `wchar_t` μπορούμε να βάζουμε ως τιμή έναν ορμαθό τιμών τύπου `wchar_t` είτε με τη δήλωση:

```
   wchar_t w[5]= L"αβγδ";
```

είτε αργότερα:

```
   wcscpy( w, L"wsx" );
```

Εδώ βλέπουμε τα εξής:

- Όπως μια σταθερά τύπου `char` με το πρόθεμα "`L`" γίνεται σταθερά τύπου `wchar_t`, έτσι και ένας ορμαθός τιμών τύπου `char` με το πρόθεμα "`L`" γίνεται ορμαθός τιμών τύπου `wchar_t`.
- Όπως η αντιγραφή ορμαθών τύπου `char` γίνεται με τη *strcpy*, η αντιγραφή ορμαθών τύπου `wchar_t` γίνεται με τη *wcscpy*.

Γενικώς, για κάθε συνάρτηση *strX* υπάρχει αντίστοιχη συνάρτηση *wcsX* για τη διαχείριση ορμαθών τύπου `wchar_t`.

Η C++ μας δίνει τον τύπο *std*::*wstring*, ίδιο σχεδόν με τον *std*::*string*, στις μεταβλητές του οποίου μπορούμε να αποθηκεύουμε ορμαθούς `wchar_t`. Έτσι, μπορούμε να ορίσουμε:

```
   wchar_t w[5]= L"αβγδ";
   wstring w0, w1( L"abc" ), w2( 4, L'q' ), w3( w );
```

Προσπαθώντας να δούμε τις τιμές των μεταβλητών *wstring* συναντούμε ένα πρόβλημα: Μια από τις διαφορές του *wstring* από τον *string* είναι ότι δεν μπορούμε να βγάλουμε την τιμή μεταβλητής *wstring* σε κάποιο ρεύμα με τον "<<". Δίνουμε μια πρόχειρη λύση –ας πούμε για τη *w1*– ως εξής:

```
int ndx;
for ( ndx = 0; ndx < w1.length(); ndx = ndx+1 )
   cout << static_cast<unsigned char>(w1[ndx]);
cout << endl;
```

Με αυτόν τον τρόπο μπορούμε να δούμε ότι το *w0* είναι κενό ενώ για τα *w1*, *w2* και *w3* παίρνουμε αντιστοίχως:

```
abc
qqqq
αβγδ
```

Μπορείς να αλλάξεις την τιμή μιας τέτοιας μεταβλητής με εκχώρηση, π.χ.:

```
w0 = L"qazw";
```

Με τη *c_str* μπορείς να βγάλεις το περιεχόμενο σε πίνακα και με τη *wcscpy* να το αντιγράψεις σε έναν πίνακα με στοιχεία `wchar_t`:

```
wcscpy( w, w1.c_str() );
```

Τέλος, για να μη σου μείνουν αμφιβολίες, δίνοντας:

```
cout << typeid(w1[0]).name() << "  " << sizeof(w1[0]) << endl;
```

παίρνεις αποτέλεσμα:[10]

```
wchar_t  2
```

10.15 Εν Κατακλείδι ...

Η C++ μας δίνει πολλά και αξιόλογα εργαλεία για να διαχειριστούμε κείμενο. Όλα περιλαμβάνονται (ή σχετίζονται) με την κλάση *string*. Μάθαμε λοιπόν:

- Να δηλώνουμε μεταβλητές τύπου *string*.
- Να αλλάζουμε την τιμή τους.
- Να διαβάζουμε από και να γράφουμε σε ρεύματα αρχείων (ή τα *cin*, *cout*) τιμές μεταβλητών *string*.
- Να συγκρίνουμε τιμές τύπου *string*.
- Να αναζητούμε κείμενο μέσα σε άλλο κείμενο.
- Να μετατρέπουμε τιμές *string* σε αριθμητικές και αντιστρόφως.

Είδαμε εν συντομία και τα αντίστοιχα εργαλεία της C· θα τα δεις να χρησιμοποιούνται σε πολλές περιπτώσεις.

Ασκήσεις

Α Ομάδα

10-1 Γράψε πρόγραμμα που θα διαβάζει έναν ορμαθό χαρακτήρων και θα αποφαίνεται αν είναι ή δεν είναι της μορφής *qq*. Π.χ. οι "αβγαβγ", "zppquzppqu" είναι της μορφής *qq*, ενώ οι "αβγδε", "zppqqppz" δεν είναι.

10-2 Ξαναγράψε το πρόγραμμα του παραδείγματος της §10.5, έτσι ώστε να διαβάζει τα ονόματα των γλωσσών από αρχείο text.

[10] Borland C++ 5.5.

10-3 Ξαναγράψε το πρόγραμμα του παράδ. 1 της §8.10 έτσι ώστε να διαβάζει από το αρχείο ολόκληρες γραμμές και όχι χαρακτήρα προς χαρακτήρα.

10-4 Κάνε το ίδιο για το πρόγραμμα του παραδ. 2 της §8.10.

Β Ομάδα

10-5 Γράψε μια:

```
string getRight( string s, int n )
```

που θα επιστρέφει ως τιμή τους *n* τελευταίους (δεξιότερους) του *s*. Αν $n \le 0$ η τιμή θα είναι κενή, ενώ αν $n \ge s.length()$ η τιμή θα είναι αυτή της *s*.

10-6 Ο αριθμός μιας πιστωτικής κάρτας δημιουργείται ως εξής:

- Τα πρώτα ψηφία χαρακτηρίζουν την κάρτα. Στον Πίν. 10-1 βλέπεις τα χαρακτηριστικά για ορισμένες πολύ γνωστές πιστωτικές κάρτες.
- Στη συνέχεια υπάρχουν τα ψηφία που δείχνουν την τράπεζα.
- Μετά υπάρχει ο αριθμός του πελάτη.
- Το τελευταίο ψηφίο είναι **ψηφίο ελέγχου** (check digit) επιλεγμένο έτσι ώστε να ισχύει η συνθήκη εγκυρότητας LUHN.

Τύπος Κάρτας	**Προθεμα**	**Μήκος**	**Αλγόριθμος Ψηφίου Ελέγχου**
MASTERCARD	51-55	16	mod 10
VISA	4	13, 16	mod 10
AMEX	34, 37	15	mod 10
Diners Club/ Carte Blanche	300-305 36, 38	14	mod 10
Discover	6011	16	mod 10
enRoute	2014 2149	15	any
JCB	3	16	mod 10
JCB	2131 1800	15	mod 10

Πίν. 10-1 Στη στήλη **Πρόθεμα** βλέπεις το χαρακτηριστικό κάθε κάρτας και στη στήλη **Μήκος** το μήκος του αριθμού που χρησιμοποιεί.

Η συνθήκη εγκυρότητας LUHN υπολογίζεται ως εξής:

Βήμα 1: Πολλαπλασίασε επί δύο τα ψηφία που έχουν άρτιες θέσεις (θέση 1 η τελευταία, 2 η προτελευταία κ.ο.κ.)

Βήμα 2: Πρόσθεσε όλα τα ψηφία των αριθμών που προέκυψαν από τους διπλασιασμούς και αυτά που βρίσκονται σε μονές θέσεις.

Αν το άθροισμα είναι πολλαπλάσιο του 10 ο αριθμός είναι έγκυρος.

Παράδειγμα

Έστω ότι έχουμε τον αριθμό:

49927398716

Βήμα 1:

```
        4 9 9 2 7 3 9 8 7 1 6
         x2  x2  x2  x2  x2
------------------------------
         18   4   6  16   2
```

Βήμα 2: 4 +(1+8)+ 9 + (4) + 7 + (6) + 9 +(1+6) + 7 + (2) + 6 = 70

Άρα ο αριθμός είναι έγκυρος.

α) Γράψε συνάρτηση *isValidCrCardNum* που θα παίρνει (ως όρισμα) μια τιμή τύπου *string* που θα είναι αριθμός πιστωτικής κάρτας, θα εξετάζει το ψηφίο ελέγχου και θα επιστρέφει ως τιμή `true` αν ο αριθμός είναι έγκυρος (σωστό ψηφίο ελέγχου) και `false` αν δεν είναι.

β) Γράψε συνάρτηση *appendChkDigit* που θα παίρνει (ως όρισμα) μια τιμή τύπου *string* που θα είναι αριθμός πιστωτικής κάρτας χωρίς το τελευταίο ψηφίο. Η συνάρτηση θα συμπληρώνει το τελευταίο ψηφίο με βάση τα όσα είπαμε παραπάνω.

Γράψε πρόγραμμα που θα σου επιτρέπει να δοκιμάσεις τις συναρτήσεις που έγραψες.

10-7 Γράψε πρόγραμμα που θα διαβάζει ένα κείμενο από ένα αρχείο text και θα το αποθηκεύει σε μια μεταβλητή τύπου `string`. Κατά την ανάγνωση θα αντικαθιστά κάθε αλλαγή γραμμής με ένα διάστημα (κενό). Στη συνέχεια θα διαβάζει λέξεις από το πληκτρολόγιο και θα μας λέει πόσες φορές υπάρχει κάθε μια από αυτές στο κείμενο. Θα τελειώνει όταν πληκτρολογήσουμε τη λέξη `zzz`.

Γ Ομάδα

10-8 (*Πιο ρεαλιστική παραλλαγή της άσκ.* 8-7) Ένα αρχείο text, με όνομα στο δίσκο misthoi.txt, είναι γραμμένο ως εξής: Σε κάθε γραμμή έχει ένα επώνυμο εργαζόμενου και τρεις αριθμούς ως εξής:

```
ΣΙΩΠΑΚΗΣ\t1234.5\t7\t2\n
```

(το '`\t`' παριστάνει τον χαρακτήρα tab και το '`\n`' την αλλαγή γραμμής.) Ο πρώτος αριθμός (πραγματικός) –εδώ 1234.5– είναι ο βασικός μισθός ενός υπαλλήλου, ο δεύτερος –εδώ 7– ο αριθμός ετών υπηρεσίας του ίδιου υπαλλήλου και ο τρίτος –εδώ 2– ο αριθμός των παιδιών του. Το πλήθος των στοιχείων είναι άγνωστο.

Το περιεχόμενο του αρχείου δεν έχει ελεγχθεί. Έτσι, μπορεί να έχει:

- Κενές γραμμές (μικρό το κακό).
- Γραμμές από τις οποίες λείπουν στοιχεία.
- Λαθεμένα στοιχεία που παραβιάζουν κάποια από τα παρακάτω:
  - Το επώνυμο του εργαζόμενου δεν πρέπει να είναι κενό.
  - 800 ≤ βασικός μισθός ≤ 3500,
  - 0 ≤ χρόνια υπηρεσίας ≤ 35,
  - 0 ≤ αριθμός παιδιών ≤ 12.

Γράψε πρόγραμμα που θα διαβάζει το αρχείο και θα δημιουργεί ένα νέο –το log.txt– με καταγραφή μιας γραμμής για κάθε λάθος που θα βρίσκει. Π.χ.:

```
Line 7 salary: 35650
Line 7 number of children: -4
Line 11 number of years: 51
Line 18 empty
Line 26 salary: 955000
Line 28 incomplete line
Line 36 name empty
```

Υπόδ.: Διάβαζε ολόκληρες γραμμές (μέχρι να βρεις '`\n`') με τη *getline*. Σε κάθε γραμμή θα πρέπει να υπάρχει το "`\t`" ακριβώς 3 φορές.