
C

Εφαρμογές

Εφαρμογή 1η : Υπολογισμός μοριακού βάρους πρωτεΐνης από την πρωτοταγή της δομή.

Αρχικοποίηση του πίνακα ...

```
for ( i = 0 ; i < 127 ; i++ )      /* Για αρχή δώσε αρνητικές   */
    weights[i] = -1.0;             /* τιμές σε όλα τα μορ. βάρη */

weights['A'] = 71.08 ;             /* ... και στη συνέχεια δώσε */
weights['C'] = 103.14;             /* (θετικές) τιμές σε όσα    */
weights['D'] = 115.09;             /* σύμβολα (αμινοξέων)       */
weights['E'] = 129.12;             /* γνωρίζουμε.               */
weights['F'] = 147.18;
weights['G'] = 57.05 ;
weights['H'] = 137.14;
weights['I'] = 113.16;
weights['K'] = 128.17;
weights['L'] = 113.16;
weights['M'] = 131.19;
weights['N'] = 114.10;
weights['P'] = 97.12 ;
weights['Q'] = 128.13;
weights['R'] = 156.19;
weights['S'] = 87.08 ;
weights['T'] = 101.11;
weights['V'] = 99.13 ;
weights['W'] = 186.21;
weights['Y'] = 163.18;
```

```
#include <stdio.h>

int     main()
{
    float   mw;
    float   weights[127];
    char    s[10000];
    int i;

    /*... ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΟΥ ΠΙΝΑΚΑ weights[] ...*/

    mw = 0.0;
    while( scanf("%s", s) != EOF )
        {
            i=0;
            while( s[i] != 0 )
                {
                    if ( weights[ s[i] ] < 0.0 )
                        printf("Unknown character : %c\n", s[i] );
                    else
                        mw += weights[ s[i] ];

                    i++;
                            }
                                }
    printf("The molecular weight is %f\n", (mw+18.0) );
}
```

Εφαρμογή 2η : Μετατροπή της αλληλουχίας πρωτεϊνών από το συμβολισμό του ενός γράμματος στο πλήρες αμινοξικό όνομα.

Τι θέλουμε να επιτύχουμε ;

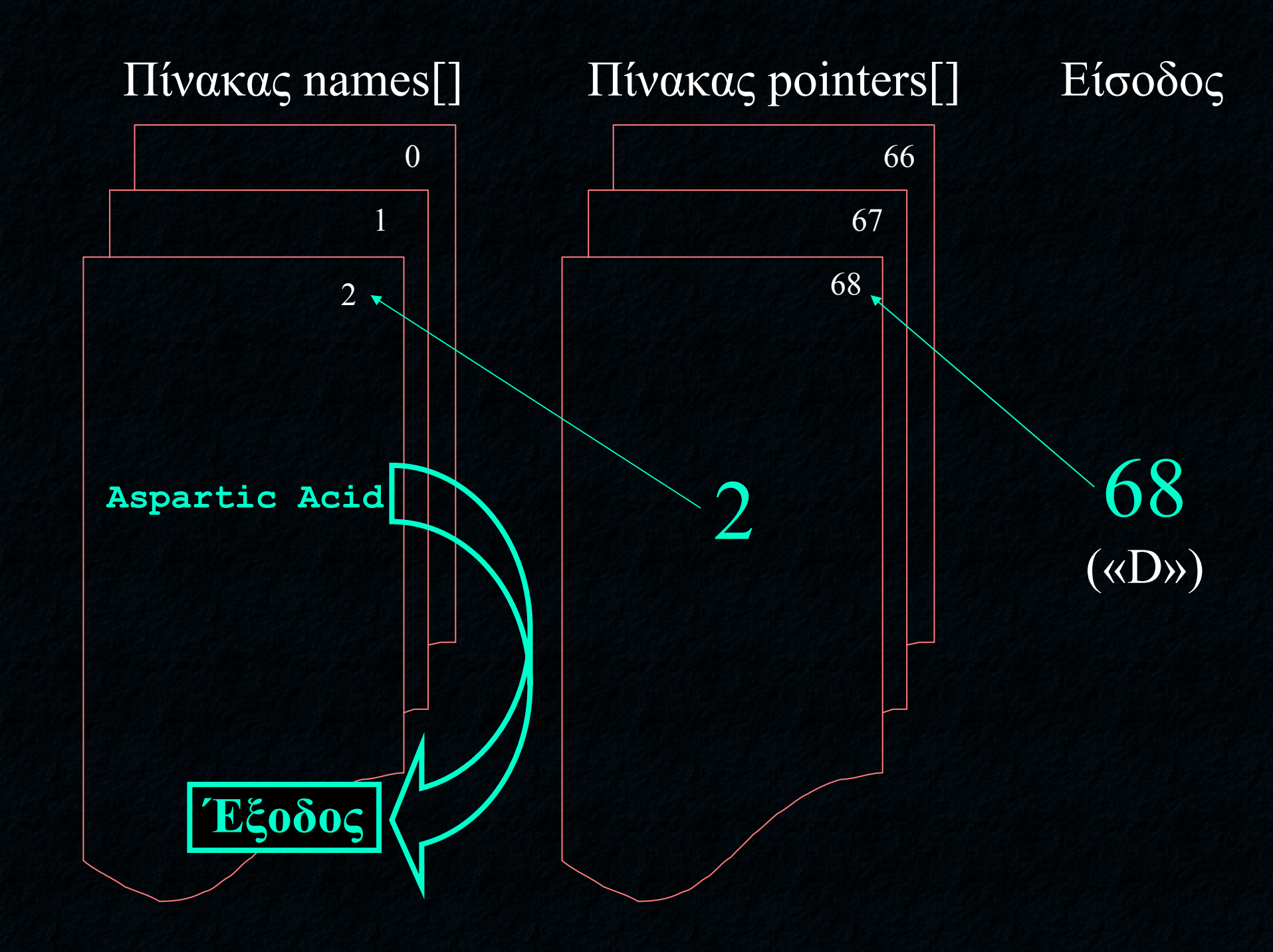
```
# ./a.out
FSDFSGFASD
     1  Phenylalanine
     2  Serine
     3  Aspartic Acid
     4  Phenylalanine
     5  Serine
     6  Glycine
     7  Phenylalanine
     8  Alanine
     9  Serine
    10  Aspartic Acid
```

- Όρισε ένα πίνακα χαρακτήρων (20 επί 15) στον οποίο θα αποθηκεύσουμε τα πλήρη ονόματα των 20 αμινοξέων

```
char    names[20][15] = {
      "Alanine       ", "Cysteine      ", "Aspartic Acid ",
      "Glutamic Acid ", "Phenylalanine ", "Glycine       ",
      "Histidine     ", "Isoleucine    ", "Lysine        ",
      "Leucine       ", "Methionine    ", "Asparagine    ",
      "Proline       ", "Glutamine     ", "Arginine      ",
      "Serine        ", "Threonine     ", "Valine        ",
      "Tryptophan    ", "Tyrosine      "   };
```

- Όρισε ένα πίνακα ακεραίων 127 θέσεων και σε όσες θέσεις του πίνακα αντιστοιχούν σε αμινοξύ, βάλε τη θέση του ονόματος του αμινοξέος στον πίνακα names[][] :

```
for ( i = 0 ; i < 127 ; i++ )
        pointers[i] = -1;
pointers['A'] = 0;
pointers['C'] = 1;
pointers['D'] = 2;
pointers['E'] = 3;
pointers['F'] = 4;
pointers['G'] = 5;
pointers['H'] = 6;
pointers['I'] = 7;
pointers['K'] = 8;
pointers['L'] = 9;
pointers['M'] = 10;
pointers['N'] = 11;
pointers['P'] = 12;
pointers['Q'] = 13;
pointers['R'] = 14;
pointers['S'] = 15;
pointers['T'] = 16;
pointers['V'] = 17;
pointers['W'] = 18;
pointers['Y'] = 19;
```


Πίνακας names[]
Πίνακας pointers[]
Είσοδος
0
1
2
66
67
68
Aspartic Acid
2
68
(«D»)
Έξοδος

```
#include <stdio.h>

int   main()
{
  char  s[10000];
  int i;
  int total;
  int pointers[127];
  char    names[20][15] = {  /* Αρχικοποιήσεις πινάκων
  . . . . . . . . . . . . . . . . . . . . . . . .  .. . */
  total = 1;
  while( scanf("%s", s) != EOF )
    {
      i=0;
      while( s[i] != 0 )
        {
          if ( pointers[ (int)(s[i]) ] < 0 )
            printf("Unknown character : %c\n", s[i] );
          else
            {
            printf("%6d  %s\n", total, names[ pointers[(int)(s[i])]]);
            total++;
            }
          i++;
              }
                }
  return(0);
}
```

Εφαρμογή 3η : Εύρεση πρωτεϊνικής αλληλουχίας από την κωδικοποιούσα DNA αλληλουχία.

```
ID   HHBRODA    standard; DNA; PRO; 1229 BP.
AC   M11720;
SV   M11720.1
DT   16-JUL-1988 (Rel. 16, Created)
DT   05-JUL-1999 (Rel. 60, Last updated, Version 3)
DE   Halobacterium bacteriorhodopsin gene, complete cds.
KW   bacteriorhodopsin.
OS   Halobacterium salinarum
OC   Archaea; Euryarchaeota; Halobacteriales; Halobacteriaceae; Halobacterium.
RN   [1]
RP   1-1229
RX   MEDLINE; 84206613.
RA   Dunn R.J., Hackett N.R., Huang K.-S., Jones S., Khorana H.G., Lee D.-S.,
RA   Liao M.-J., Lo K.-M., McCoy J., Noguchi S., Radhakrishnan R.,
RA   RajBhandary U.L.;
RT   "Studies on the light-transducing pigment bacteriorhodopsin";
RL   Cold Spring Harb. Symp. Quant. Biol. 48:853-862(1983).
DR   SWISS-PROT; P02945; BACR_HALHA.
SQ   Sequence 1229 BP; 222 A; 359 C; 386 G; 262 T; 0 other;
     gggtgcaacc gtgaagtccg ccacgaccgc gtcacgacag gagccgacca gcgacaccca        60
     gaaggtgcga acggttgagt gccgcaacga tcacgagttt ttcgtgcgct tcgagtggta       120
     acacgcgtgc acgcatcgac ttcaccgcgg gtgtttcgac gccagccggc cgttgaacca       180
     gcaggcagcg ggcatttaca gccgctgtgg cccaaatggt ggggtgcgct attttggtat       240
     ggtttggaat ccgcgtgtcg gctccgtgtc tgacggttca tcggttctaa attccgtcac       300
     gagcgtacca tactgattgg gtcgtagagt tacacacata tcctcgttag gtactgttgc       360
     atgttggagt tattgccaac agcagtggag ggggtatcgc aggcccagat caccggacgt       420
     ccggagtgga tctggctagc gctcggtacg gcgctaatgg gactcgggac gctctatttc       480
     ctcgtgaaag ggatgggcgt ctcggaccca gatgcaaaga aattctacgc catcacgacg       540
     ctcgtcccag ccatcgcgtt cacgatgtac ctctcgatgc tgctggggta tggcctcaca       600
     atggtaccgt tcggtgggga gcagaacccc atctactggg cgcggtacgc tgactggctg       660
     ttcaccacgc cgctgttgtt gttagacctc gcgttgctcg ttgacgcgga tcagggaacg       720
     atccttgcgc tcgtcggtgc cgacggcatc atgatcggga ccggcctggt cggcgcactg       780
     acgaaggtct actcgtaccg cttcgtgtgg tgggcgatca gcaccgcagc gatgctgtac       840
     atcctgtacg tgctgttctt cgggttcacc tcgaaggccg aaagcatgcg ccccgaggtc       900
     gcatccacgt tcaaagtact gcgtaacgtt accgttgtgt tgtggtccgc gtatcccgtc       960
     gtgtggctga tcggcagcga aggtgcggga atcgtgccgc tgaacatcga gacgctgctg      1020
     ttcatggtgc ttgacgtgag cgcgaaggtc ggcttcgggc tcatcctcct gcgcagtcgt      1080
     gcgatcttcg gcgaagccga agcgccggag ccgtccgccg gcgacggcgc ggccgcgacc      1140
     agcgactgat cgcacacgca ggacagcccc acaaccggcg cggctgtgtt caacgacaca      1200
     cgatgagtcc cccactcggt cttgtactc                                        1229
//
```

Απόσπασμα από την καταχώρηση για την DNA αλληλουχία της βακτηριοροδοψίνης όπως είναι στη βάση δεδομένων του EMBL (Nucleotide Sequence database).

atgttggagt tattgccaac agcagtggag ggggtatcgc aggcccagat caccggacgt
ccggagtgga tctggctagc gctcggtacg gcgctaatgg gactcgggac gctctatttc
ctcgtgaaag ggatgggcgt ctcggaccca gatgcaaaga aattctacgc catcacgacg
ctcgtcccag ccatcgcgtt cacgatgtac ctctcgatgc tgctggggta tggcctcaca
atggtaccgt tcggtgggga gcagaacccc atctactggg cgcggtacgc tgactggctg
ttcaccacgc cgctgttgtt gttagacctc gcgttgctcg ttgacgcgga tcagggaacg
atccttgcgc tcgtcggtgc cgacggcatc atgatcggga ccggcctggt cggcgcactg
acgaaggtct actcgtaccg cttcgtgtgg tgggcgatca gcaccgcagc gatgctgtac
atcctgtacg tgctgttctt cgggttcacc tcgaaggccg aaagcatgcg ccccgaggtc
gcatccacgt tcaaagtact gcgtaacgtt accgttgtgt tgtggtccgc gtatcccgtc
gtgtggctga tcggcagcga aggtgcggga atcgtgccgc tgaacatcga gacgctgctg
ttcatggtgc ttgacgtgag cgcgaaggtc ggcttcgggc tcatcctcct gcgcagtcgt
gcgatcttcg gcgaagccga agcgccggag ccgtccgccg gcgacggcgc ggccgcgacc
agcgactga

atgttggagt tattgccaac agcagtggag ggggtatcgc aggcccagat caccggacgt
ccggagtgga tctggctagc gctcggtacg gcgctaatgg gactcgggac gctctatttc
ctcgtgaaag ggatgggcgt ctcggaccca gatgcaaaga aattctacgc catcacgacg
ctcgtcccag ccatcgcgtt cacgatgtac ctctcgatgc tgctggggta tggcctcaca
atggtaccgt tcggtgggga gcagaacccc atctactggg cgcggtacgc tgactggctg
ttcaccacgc cgctgttgtt gttagacctc gcgttgctcg ttgacgcgga tcagggaacg
atccttgcgc tcgtcggtgc cgacggcatc atgatcggga ccggcctggt cggcgcactg
acgaaggtct actcgtaccg cttcgtgtgg tgggcgatca gcaccgcagc gatgctgtac
atcctgtacg tgctgttctt cgggttcacc tcgaaggccg aaagcatgcg ccccgaggtc
gcatccacgt tcaaagtact gcgtaacgtt accgttgtgt tgtggtccgc gtatcccgtc
gtgtggctga tcggcagcga aggtgcggga atcgtgccgc tgaacatcga gacgctgctg
ttcatggtgc ttgacgtgag cgcgaaggtc ggcttcgggc tcatcctcct gcgcagtcgt
gcgatcttcg gcgaagccga agcgccggag ccgtccgccg gcgacggcgc ggccgcgacc
agcgactga

Μεταγραφή, Μετάφραση

Πρωτεΐνη

atgttggagt tattgccaac agcagtggag ggggtatcgc aggcccagat caccggacgt
ccggagtgga tctggctagc gctcggtacg gcgctaatgg gactcgggac gctctatttc
ctcgtgaaag ggatgggcgt ctcggaccca gatgcaaaga aattctacgc catcacgacg
ctcgtcccag ccatcgcgtt cacgatgtac ctctcgatgc tgctggggta tggcctcaca
atggtaccgt tcggtgggga gcagaacccc atctactggg cgcggtacgc tgactggctg
ttcaccacgc cgctgttgtt gttagacctc gcgttgctcg ttgacgcgga tcagggaacg
atccttgcgc tcgtcggtgc cgacggcatc atgatcggga ccggcctggt cggcgcactg
acgaaggtct actcgtaccg cttcgtgtgg tgggcgatca gcaccgcagc gatgctgtac
atcctgtacg tgctgttctt cgggttcacc tcgaaggccg aaagcatgcg ccccgaggtc
gcatccacgt tcaaagtact gcgtaacgtt accgttgtgt tgtggtccgc gtatcccgtc
gtgtggctga tcggcagcga aggtgcggga atcgtgccgc tgaacatcga gacgctgctg
ttcatggtgc ttgacgtgag cgcgaaggtc ggcttcgggc tcatcctcct gcgcagtcgt
gcgatcttcg gcgaagccga agcgccggag ccgtccgccg gcgacggcgc ggccgcgacc
agcgactga

«Γενετικός κώδικας»

Πρωτεΐνη

	T	C	A	G
T	TTT Phe (F)	TCT Ser (S)	TAT Tyr (Y)	TGT Cys (C)
	TTC "	TCC "	TAC "	TGC "
	TTA Leu (L)	TCA "	TAA Ter	TGA Ter
	TTG "	TCG "	TAG Ter	TGG Trp (W)
C	CTT Leu (L)	CCT Pro (P)	CAT His (H)	CGT Arg (R)
	CTC "	CCC "	CAC "	CGC "
	CTA "	CCA "	CAA Gln (Q)	CGA "
	CTG "	CCG "	CAG "	CGG "
A	ATT Ile (I)	ACT Thr (T)	AAT Asn (N)	AGT Ser (S)
	ATC "	ACC "	AAC "	AGC "
	ATA "	ACA "	AAA Lys (K)	AGA Arg (R)
	ATG Met (M)	ACG "	AAG "	AGG "
G	GTT Val (V)	GCT Ala (A)	GAT Asp (D)	GGT Gly (G)
	GTC "	GCC "	GAC "	GGC "
	GTA "	GCA "	GAA Glu (E)	GGA "
	GTG "	GCG "	GAG "	GGG "

	T	C	A	G
T	TTT Phe (F)	TCT Ser (S)	TAT Tyr (Y)	TGT Cys (C)
	TTC "	TCC "	TAC "	TGC "
	TTA Leu (L)	TCA "	TAA Ter	TGA Ter
	TTG "	TCG "	TAG Ter	TGG Trp (W)
C	CTT Leu (L)	CCT Pro (P)	CAT His (H)	CGT Arg (R)
	CTC "	CCC "	CAC "	CGC "
	CTA "	CCA "	CAA Gln (Q)	CGA "
	CTG "	CCG "	CAG "	CGG "
A	ATT Ile (I)	ACT Thr (T)	AAT Asn (N)	AGT Ser (S)
	ATC "	ACC "	AAC "	AGC "
	ATA "	ACA "	AAA Lys (K)	AGA Arg (R)
	ATG Met (M)	ACG "	AAG "	AGG "
G	GTT Val (V)	GCT Ala (A)	GAT Asp (D)	GGT Gly (G)
	GTC "	GCC "	GAC "	GGC "
	GTA "	GCA "	GAA Glu (E)	GGA "
	GTG "	GCG "	GAG "	GGG "

Τι θέλουμε να επιτύχουμε ;

```
# ./a.out
tt cTC GG a CGGC
    ttc      1  Phenylalanine
    TCG      2  Serine
    GaC      3  Aspartic Acid
    GGC      4  Glycine
```

Τι θέλουμε να επιτύχουμε ;

Αλληλουχία του DNA.

```
# ./a.out
tt cTC GG a CGGC
    ttc      1  Phenylalanine
    TCG      2  Serine
    GaC      3  Aspartic Acid
    GGC      4  Glycine
```

Τι θέλουμε να επιτύχουμε ;

Αρίθμηση και αλληλουχία της πρωτεΐνης.

```
# ./a.out
tt cTC GG a CGGC
    ttc      1  Phenylalanine
    TCG      2  Serine
    GaC      3  Aspartic Acid
    GGC      4  Glycine
```

Τι θέλουμε να επιτύχουμε ;

Αρίθμηση και αλληλουχία της πρωτεΐνης.

```
# ./a.out
tt cTC GG a CGGC
    ttc      1  Phenylalanine
    TCG      2  Serine
    GaC      3  Aspartic Acid
    GGC      4  Glycine
```

Πάλι, θέλουμε να μην βάλουμε όρους για τη μορφή της εισόδου, να είμαστε σε θέση να ανιχνεύσουμε σφάλματα εισόδου, και να αναγνωρίζουμε τόσο μικρά, όσο και κεφαλαία γράμματα για την αλληλουχία του DNA. Πέρα από τα 20 αμινοξέα, πρέπει να είμαστε σε θέση να χειριστούμε και τους κώδικες τερματισμού (termination codons, TAG, TGA, TAA).

Το καινούργιο πρόβλημα που έχουμε είναι :

Στην αλληλουχία εισόδου ΔΕΝ είναι πλέον κάθε χαρακτήρας ανεξάρτητος : προκειμένου να κάνουμε ένα κύκλο υπολογισμού (μετάφρασης DNA-πρωτεΐνη) χρειαζόμαστε τρεις νόμιμους χαρακτήρες που καθένας τους να αντιστοιχεί σε κάποια από τις τέσσερις βάσεις (κάποιους εκ των χαρακτήρων a,g,c,t,A,G,C,T).

Εάν, για παράδειγμα, η είσοδος είναι :

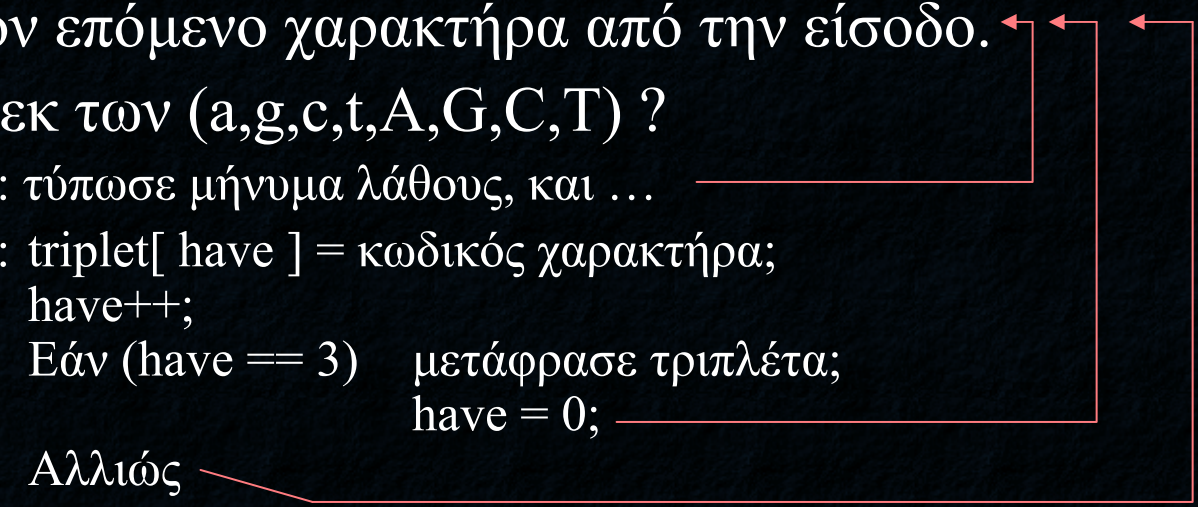
a c g gg 1xty

ttg

το πρόγραμμα μας θα πρέπει να «δει» μόνο τρεις νόμιμες τριπλέτες (τις ACG GGT TTG), και να δώσει μηνύματα λάθους για τους τρεις άγνωστους χαρακτήρες (1, x, y).

Μια λύση είναι να αποθηκεύουμε προσωρινά τους νόμιμους χαρακτήρες που λαμβάνουμε από την είσοδο (ας πούμε σε έναν πίνακα `triplet[3]`) μέχρι να έχουμε μια πλήρη τριπλέτα.
Με το που έχουμε γεμίσει τον πίνακα, προχωράμε στη μετάφραση της τρέχουσας τριπλέτας, και αρχίζουμε να ξανά (για την επόμενη τριπλέτα) να προσθέτουμε στοιχεία στον πίνακα (αρχίζοντας από το πρώτο του στοιχείο). Δηλαδή :

- int have = 0; (πόσες βάσεις έχω ήδη διαβάσει ;)
- int triplet[3];
- Διάβασε τον επόμενο χαρακτήρα από την είσοδο.
- Είναι ένας εκ των (a,g,c,t,A,G,C,T) ?
  - Εάν ΟΧΙ : τύπωσε μήνυμα λάθους, και …
  - Εάν ΝΑΙ : triplet[have] = κωδικός χαρακτήρα;
    have++;
    Εάν (have == 3) μετάφρασε τριπλέτα;
    have = 0;
    Αλλιώς



Η λύση που θα ακολουθήσουμε για αυτό το παράδειγμα είναι τέτοια ώστε ο καθαρός κώδικας του προγράμματος να χωράει σε μια διαφάνεια. Αυτό σημαίνει αναγκαστικά ότι η δομή των δεδομένων θα είναι πιο πολύπλοκη απ' ότι στα προηγούμενα παραδείγματα.

Επίσης, θα **αποφύγουμε** να κατασπαταλήσουμε χώρο στη μνήμη μέσω χρήσης πινάκων του τύπου names[127][127][127][15]

Η λύση που θα ακολουθήσουμε για αυτό το παράδειγμα είναι τέτοια ώστε ο καθαρός κώδικας του προγράμματος να χωράει σε μια διαφάνεια. Αυτό σημαίνει αναγκαστικά ότι η δομή των δεδομένων θα είναι πιο πολύπλοκη απ' ότι στα προηγούμενα παραδείγματα.

Επίσης, θα **αποφύγουμε** να κατασπαταλήσουμε χώρο στη μνήμη μέσω χρήσης πινάκων του τύπου names[127][127][127][15]

Συνολικό μέγεθος πίνακα :
127 x 127 x 127 x 15 =
30,725,745 bytes (δηλ. 29 Mbytes)

Ορίζουμε έναν πίνακα με το όνομα names[][] στον οποίο θα αποθηκεύσουμε τα ονόματα των 20 αμινοξέων και το μήνυμα που θα εμφανίζεται όταν βρεθεί termination codon (όπως ακριβώς και με την δεύτερη εφαρμογή) :

```
char    names[21][15] = {
  "Alanine       ", "Cysteine      ", "Aspartic Acid ",
  "Glutamic Acid ", "Phenylalanine ", "Glycine       ",
  "Histidine     ", "Isoleucine    ", "Lysine        ",
  "Leucine       ", "Methionine    ", "Asparagine    ",
  "Proline       ", "Glutamine     ", "Arginine      ",
  "Serine        ", "Threonine     ", "Valine        ",
  "Tryptophan    ", "Tyrosine      ", "TERMINATION   "
                                       };
```

Για να κάνουμε τη συγγραφή του προγράμματος πιο καθαρή, ορίζουμε στην επικεφαλίδα του προγράμματος, σύμβολα για τη θέση του κάθε αμινοξέος στον πίνακα names[][] :

```
#define   ALA 0
#define   CYS 1
#define   ASP 2
#define   GLU 3
#define   PHE 4
#define   GLY 5
#define   HIS 6
#define   ILE 7
#define   LYS 8
#define   LEU 9
#define   MET 10
#define   ASN 11
#define   PRO 12
#define   GLN 13
#define   ARG 14
#define   SER 15
#define   THR 16
#define   VAL 17
#define   TRP 18
#define   TYR 19
#define  TERM 20
```

Για να κάνουμε τη συγγραφή του προγράμματος πιο καθαρή, ορίζουμε στην επικεφαλίδα του προγράμματος, σύμβολα για τη θέση του κάθε αμινοξέος στον πίνακα names[][] :

```
#define   ALA 0
#define   CYS 1
#define   ASP 2
#define   GLU 3
#define   PHE 4
#define   GLY 5
#define   HIS 6
#define   ILE 7
#define   LYS 8
#define   LEU 9
#define   MET 10
#define   ASN 11
#define   PRO 12
#define   GLN 13
#define   ARG 14
#define   SER 15
#define   THR 16
#define   VAL 17
#define   TRP 18
#define   TYR 19
#define  TERM 20
```

Τα ALA, CYS, ASP, …, TERM ***δεν είναι*** μεταβλητές του προγράμματος. Είναι συμβολικά ονόματα για αριθμητικές σταθερές που χρησιμοποιούνται από το πρόγραμμα.
Η χρήση τους διευκολύνει την ανάγνωση προγραμμάτων (π.χ. με το να βλέπουμε «ARG» αντί «14»), άλλα δεν αλλάζει αυτό καθαυτό το πρόγραμμα.

Θέλουμε να βάλουμε τον καθαυτό κώδικα μετάφρασης σε ένα πίνακα με όσο το δυνατό μικρότερο αριθμό στοιχείων. Ο πλέον οικονομικός τρόπος είναι (βέβαια) ένας πίνακας 4x4x4, ας πούμε ο πίνακας pointers[4][4][4]

Ποια από τις τέσσερις βάσεις είναι στην πρώτη θέση της τριπλέτας ? Η βάση
0 (ας πούμε T),
1 (ας πούμε C),
2 (ας πούμε A),
3 (ας πούμε G) ?

Ποια από τις τέσσερις βάσεις είναι στη δεύτερη θέση της τριπλέτας ? Η βάση
0 (ας πούμε T),
1 (ας πούμε C),
2 (ας πούμε A),
3 (ας πούμε G) ?

Ποια από τις τέσσερις βάσεις είναι στην τρίτη θέση της τριπλέτας ? Η βάση
0 (ας πούμε T),
1 (ας πούμε C),
2 (ας πούμε A),
3 (ας πούμε G) ?

Για να κάνουμε τη συγγραφή του προγράμματος πιο καθαρή, ορίζουμε στην επικεφαλίδα του προγράμματος, σύμβολα για τις βάσεις του DNA :

```
#define   dT  0
#define   dC  1
#define   dA  2
#define   dG  3
```

και τις χρησιμοποιούμε μαζί με τους προηγούμενους ορισμούς για να ορίσουμε τον βασικό πίνακα μετάφρασης (τον γενετικό κώδικα σε C) ως εξής :

```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dG][dA] = TERM;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
. . . . . .
```

```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dG][dA] = TERM;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
. . . . . .
```

Αριθμητική σταθερά που δηλώνει ποια είναι η πρώτη βάση :

```
#define   dT  0
#define   dC  1
#define   dA  2
#define   dG  3
```

```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dG][dA] = TERM;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
. . . . . .
```

Αριθμητική σταθερά που δηλώνει ποια είναι η δεύτερη βάση :

```
#define   dT  0
#define   dC  1
#define   dA  2
#define   dG  3
```

```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dG][dA] = TERM;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
. . . . . .
```

Αριθμητική σταθερά που δηλώνει ποια είναι η τρίτη βάση :

```
#define   dT  0
#define   dC  1
#define   dA  2
#define   dG  3
```

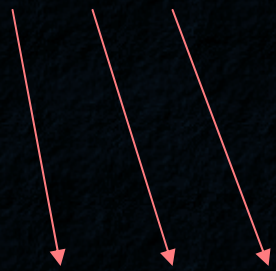
```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dG][dA] = TERM;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
. . . . . .
```

Αριθμητική σταθερά που δηλώνει σε ποια θέση του πίνακα names[][] είναι το όνομα του αντίστοιχου αμινοξέος :

```
#define   ALA 0
#define   CYS 1
#define   ASP 2
#define   GLU 3
#define   PHE 4
. . . . . .
#define   ASN 11
#define   PRO 12
#define   GLN 13
#define   ARG 14
#define   SER 15
#define   THR 16
#define   VAL 17
#define   TRP 18
#define   TYR 19
#define  TERM 20
```

Με μόνο αυτούς τους πίνακες, το πρόβλημα είναι σχεδόν έτοιμο για την λύση του :

Είσοδος : **A T G**



Τα περιεχόμενα του pointers[2][0][3] μας λένε …

. . . ότι το όνομα του αντίστοιχου αμινοξέος είναι στη θέση 10 του

… πίνακα names[][], δηλαδή η ακολουθία “Methionine”

Με μόνο αυτούς τους πίνακες, το πρόβλημα είναι σχεδόν έτοιμο για την λύση του :

Είσοδος : **A T G**

Τα περιεχόμενα του pointers[2][0][3] μας λένε …

```
pointers[dA][dT][dG] = MET;
```

```
#define   MET 10
```

. . . ότι το όνομα του αντίστοιχου αμινοξέος είναι στη θέση 10 του

… πίνακα names[][], δηλαδή η ακολουθία “Methionine”

Με μόνο αυτούς τους πίνακες, το πρόβλημα είναι σχεδόν έτοιμο για την λύση του :

Είσοδος : **A T G**

Τα περιεχόμενα του pointers[2][0][3] μας λένε …

```
pointers[dA][dT][dG] = MET;
```

```
#define   MET 10
```

. . . ότι το όνομα του αντίστοιχου αμινοξέος είναι στη θέση 10 του

```
char    names[21][15] = { . . .
          "Isoleucine    ", "Lysine        ",
          "Leucine       ", "Methionine    ",
```

… πίνακα names[][], δηλαδή η ακολουθία “Methionine”

Το μόνο πρόβλημα είναι ότι το πρόγραμμα μας θα πρέπει να περιέχει μια μεγάλη σειρά από if καθένα από τα οποία θα πρέπει να ελέγχει τους χαρακτήρες εισόδου και να τους αντιστοιχεί σε κάποια αριθμητική σταθερά (μια εκ των 0,1,2,3) για κάθε μια από τις βάσεις (T, C, A, G) :

```
if ( s[i] == 't' || s[i] == 'T' )
        val = 0;
else
        if ( s[i] == 'c' || s[i] == 'C')
                val = 1;
else
        if ( s[i] == 'a' || s[i] == 'A')
                val = 2;
else
        if ( s[i] =='g' || s[i] == 'G')
                val = 3;
else
        printf("Unknown character %c\n", s[i]);
```

Το μόνο πρόβλημα είναι ότι το πρόγραμμα μας θα πρέπει να περιέχει μια μεγάλη σειρά από if καθένα από τα οποία θα πρέπει να ελέγχει τους χαρακτήρες εισόδου και να τους αντιστοιχεί σε κάποια αριθμητική σταθερά (μια εκ των 0,1,2,3) για κάθε μια από τις βάσεις (T, C, A, G) :

```
if ( s[i] == 't' || s[i] == 'T' )
        val = dT;
else
        if ( s[i] == 'c' || s[i] == 'C')
                val = dC;
else
        if ( s[i] == 'a' || s[i] == 'A')
                val = dA;
else
        if ( s[i] =='g' || s[i] == 'G')
                val = dG;
else
        printf("Unknown character %c\n", s[i]);
```

Μπορούμε να αποφύγουμε όλα αυτά τα if -else με τον ίδιο τρόπο που είχαμε χρησιμοποιήσει στην πρώτη εφαρμογή (και χωρίς μεγάλη σπατάλη χώρου) :

```
int   symbols[127];

for ( i=0 ; i < 127 ; i++ )
   symbols[i] = -1;

symbols['a'] = dA;
symbols['A'] = dA;
symbols['t'] = dT;
symbols['T'] = dT;
symbols['c'] = dC;
symbols['C'] = dC;
symbols['g'] = dG;
symbols['G'] = dG;
```

Μπορούμε να αποφύγουμε όλα αυτά τα if -else με τον ίδιο τρόπο που είχαμε χρησιμοποιήσει στην πρώτη εφαρμογή (και χωρίς μεγάλη σπατάλη χώρου) :

```
int   symbols[127];

for ( i=0 ; i < 127 ; i++ )
   symbols[i] = -1;

symbols['a'] = dA;
symbols['A'] = dA;
symbols['t'] = dT;
symbols['T'] = dT;
symbols['c'] = dC;
symbols['C'] = dC;
symbols['g'] = dG;
symbols['G'] = dG;
```

Ο πίνακας symbols[] περιέχει την τιμή -1 για κάθε **μη** νόμιμο χαρακτήρα εισόδου, και μια εκ των αριθμητικών τιμών 0,1,2,3 για κάθε έναν από τους χαρακτήρες που αντιστοιχούν σε κάποια βάση του DNA με βάση την αντιστοιχία

```
#define   dT  0
#define   dC  1
#define   dA  2
#define   dG  3
```

Άρα, η είσοδος χαρακτήρων και ο έλεγχος του κατά πόσο είναι νόμιμοι, έχει τη μορφή :

```
while ( scanf("%s", s) != EOF) /* για κάθε αλληλουχία της εισόδου */
   {

      i=0;
      while( s[i] != 0 )  /* για κάθε χαρακτήρα της αλληλουχίας...*/
        {

                          /* εάν δεν είναι νόμιμος, τύπωσε μήνυμα */
                          /* λάθους και αγνόησε τον               */
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {

              /* Αλλιώς συνέχισε με τον υπολογισμό */

          }

        i++;
        }

   }
```

Άρα, η είσοδος χαρακτήρων και ο έλεγχος του κατά πόσο είναι νόμιμοι, έχει τη μορφή :

```
while ( scanf("%s", s) != EOF)
   {

      i=0;
      while( s[i] != 0 )
        {

          if ( symbols[ (int)(s[i]) ] < 0.0 )
              printf("Unknown character : %c\n", s[i] );
          else
            {

                /* ΥΠΟΛΟΓΙΣΜΟΣ */

            }

        i++;
        }

   }
```

Με τι μοιάζει αυτός ο «Υπολογισμός» ;

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Πόσους από τους τρεις απαιτούμενους (και νόμιμους) χαρακτήρες έχω ήδη βρει (για την επόμενη τριπλέτα που θα μεταφράσω) ;

```
int   i, have, totProt;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
       i++;
       }
 }
```

Πόσα αμινοξέα έχω ήδη μεταφράσει και γράψει (στην καθιερωμένη έξοδο) ;

```
int   i, have, totProt;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Διάβασε επόμενο νόμιμο χαρακτήρα.

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
            triplet[have] = symbols[ (int)(s[i]) ];
            seqDNA[have] = s[i];
            have++;
            if ( have == 3 )
              {
                totProt++;
                base1 = triplet[0];
                base2 = triplet[1];
                base3 = triplet[2];
                printf(" %s  %5d  %s\n", seqDNA, totProt,
                     names[pointers[base1][base2][base3] ]);
                have = 0;
              }
          }
      i++;
      }
 }
```

Βάλε στην επόμενη διαθέσιμη θέση της τριπλέτας που συμπληρώνω την αριθμητική σταθερά που αντιστοιχεί στον χαρακτήρα (δηλ. 0,1,2 ή 3).

```
char  seqDNA[4];
int   triplet[3];
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
            triplet[have] = symbols[ (int)(s[i]) ];
            seqDNA[have] = s[i];
            have++;
            if ( have == 3 )
              {
                totProt++;
                base1 = triplet[0];
                base2 = triplet[1];
                base3 = triplet[2];
                printf(" %s  %5d  %s\n", seqDNA, totProt,
                     names[pointers[base1][base2][base3] ]);
                have = 0;
              }
          }
      i++;
      }
 }
```

Αποθήκευσε επίσης αυτόν καθ' αυτόν το χαρακτήρα (ώστε να μπορούμε να τυπώσουμε στην έξοδο την τριπλέτα).

```
char  seqDNA[4];
int   triplet[3];
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
       i++;
       }
 }
```

Και αύξησε κατά ένα τον αριθμό των βάσεων (της τρέχουσας τριπλέτας) που έχω ήδη βρει.

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
       i++;
       }
 }
```

Εάν έχω μια πλήρη τριπλέτα, προχωράω με την εκτύπωση του αμινοξέος στο οποίο αντιστοιχεί, αλλιώς ...

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Πάω στο επόμενο χαρακτήρα εισόδου.

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Βρήκαμε ακόμα ένα αμινοξύ της πρωτεΐνης.

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Ο αριθμητικός κωδικός (0,1,2 ή 3) της πρώτης βάσης (στην τριπλέτα), ανατίθεται στην μεταβλητή base1.

```
int   base1, base2, base3;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Το ίδιο για την δεύτερη και τρίτη βάση (των οποίων οι τιμές πάνε στις μεταβλητές base2 και base3).

```
int   base1, base2, base3;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                      names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Τύπωσε την τριπλέτα, τον αύξοντα αριθμό αμινοξέος και το όνομα του αμινοξέος που αντιστοιχεί στη θέση pointers[base1][base2][base3]

```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                       names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
      i++;
      }
 }
```

Τύπωσε την τριπλέτα, τον αύξοντα αριθμό αμινοξέος και το όνομα του αμινοξέος που αντιστοιχεί στη θέση
pointers[base1][base2][base3]

```
pointers[0][0][0] = 4;
pointers[0][0][1] = 4;
pointers[0][0][2] = 9;
```

```
have = 0;
totProt = 0;
while ( scanf("%s", s) != EOF)
 {
    i=0;
    while( s[i] != 0 )
      {
        if ( symbols[ (int)(s[i]) ] < 0.0 )
            printf("Unknown character : %c\n", s[i] );
        else
          {
             triplet[have] = symbols[ (int)(s[i]) ];
             seqDNA[have] = s[i];
             have++;
             if ( have == 3 )
               {
                 totProt++;
                 base1 = triplet[0];
                 base2 = triplet[1];
                 base3 = triplet[2];
                 printf(" %s  %5d  %s\n", seqDNA, totProt,
                       names[pointers[base1][base2][base3] ]);
                 have = 0;
               }
          }
       i++;
       }
 }
```

Αρχή μιας νέας τριπλέτας ...

Με τι μοιάζει ολόκληρο το πρόγραμμα (επί τέλους) ;

```
#include <stdio.h>

#define         dT      0
#define         dC      1
#define         dA      2
#define         dG      3

#define         ALA     0
#define         CYS     1
#define         ASP     2
#define         GLU     3
#define         PHE     4
#define         GLY     5
#define         HIS     6
#define         ILE     7
#define         LYS     8
#define         LEU     9
#define         MET     10
#define         ASN     11
#define         PRO     12
#define         GLN     13
#define         ARG     14
#define         SER     15
#define         THR     16
#define         VAL     17
#define         TRP     18
#define         TYR     19
#define         TERM    20
```

```
int main()
{
        int     i, have, totProt;
        int     base1, base2, base3;
        char    s[10000];
        char    seqDNA[4];
        int     triplet[3];
        int     symbols[127];
        int     pointers[4][4][4];
        char    names[21][15] = {
          "Alanine       ", "Cysteine      ", "Aspartic Acid ",
          "Glutamic Acid ", "Phenylalanine ", "Glycine       ",
          "Histidine     ", "Isoleucine    ", "Lysine        ",
          "Leucine       ", "Methionine    ", "Asparagine    ",
          "Proline       ", "Glutamine     ", "Arginine      ",
          "Serine        ", "Threonine     ", "Valine        ",
          "Tryptophan    ", "Tyrosine      ", "TERMINATION   "
                                              };
        seqDNA[3] = 0;
        for ( i=0 ; i < 127 ; i++ )
                symbols[i] = -1;
        symbols['a'] = dA;
        symbols['A'] = dA;
        symbols['t'] = dT;
        symbols['T'] = dT;
        symbols['c'] = dC;
        symbols['C'] = dC;
        symbols['g'] = dG;
        symbols['G'] = dG;
```

```
int main()
{
        int     i, have, totProt;
        int     base1, base2, base3;
        char    s[10000];
        char    seqDNA[4];
        int     triplet[3];
        int     symbols[127];
        int     pointers[4][4][4];
        char    names[21][15] = {
          "Alanine       ", "Cysteine      ", "Aspartic Acid ",
          "Glutamic Acid ", "Phenylalanine ", "Glycine       ",
          "Histidine     ", "Isoleucine    ", "Lysine        ",
          "Leucine       ", "Methionine    ", "Asparagine    ",
          "Proline       ", "Glutamine     ", "Arginine      ",
          "Serine        ", "Threonine     ", "Valine        ",
          "Tryptophan    ", "Tyrosine      ", "TERMINATION   "
                                              };
        seqDNA[3] = 0;
        for ( i=0 ; i < 127 ; i++ )
                symbols[i] = -1;
        symbols['a'] = dA;
        symbols['A'] = dA;
        symbols['t'] = dT;
        symbols['T'] = dT;
        symbols['c'] = dC;
        symbols['C'] = dC;
        symbols['g'] = dG;
        symbols['G'] = dG;
```

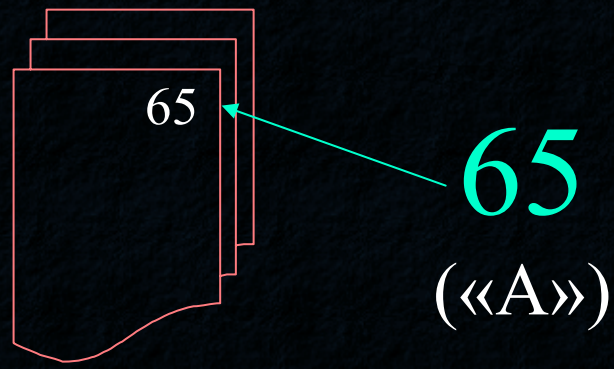
```
pointers[dT][dT][dT] = PHE;
pointers[dT][dT][dC] = PHE;
pointers[dT][dT][dA] = LEU;
pointers[dT][dT][dG] = LEU;
pointers[dC][dT][dT] = LEU;
pointers[dC][dT][dC] = LEU;
pointers[dC][dT][dA] = LEU;
pointers[dC][dT][dG] = LEU;
pointers[dA][dT][dT] = ILE;
pointers[dA][dT][dC] = ILE;
pointers[dA][dT][dA] = ILE;
pointers[dA][dT][dG] = MET;
pointers[dG][dT][dT] = VAL;
pointers[dG][dT][dC] = VAL;
pointers[dG][dT][dA] = VAL;
pointers[dG][dT][dG] = VAL;
pointers[dT][dC][dT] = SER;
pointers[dT][dC][dC] = SER;
pointers[dT][dC][dA] = SER;
pointers[dT][dC][dG] = SER;
pointers[dC][dC][dT] = PRO;
pointers[dC][dC][dC] = PRO;
pointers[dC][dC][dA] = PRO;
pointers[dC][dC][dG] = PRO;
pointers[dA][dC][dT] = THR;
pointers[dA][dC][dC] = THR;
pointers[dA][dC][dA] = THR;
pointers[dA][dC][dG] = THR;
```

```
    have = 0;
    totProt = 0;
    while ( scanf("%s", s) != EOF)
        {
            i=0;
            while( s[i] != 0 )
                {
                    if ( symbols[ (int)(s[i]) ] < 0.0 )
                        printf("Unknown character : %c\n", s[i] );
                    else
                        {
                            triplet[have] = symbols[ (int)(s[i]) ];
                            seqDNA[have] = s[i];
                            have++;
                            if ( have == 3 )
                                {
                                totProt++;
                                base1 = triplet[0];
                                base2 = triplet[1];
                                base3 = triplet[2];
                                printf(" %s  %5d  %s\n", seqDNA, totProt,
                                    names[pointers[base1][base2][base3] ]);
                                have = 0;
                                }
                                    }
            i++;
                                        }
                                            }
    if ( have != 0 )
        printf("INCOMPLETE TRIPLET.\n");
    return(0);
}
```

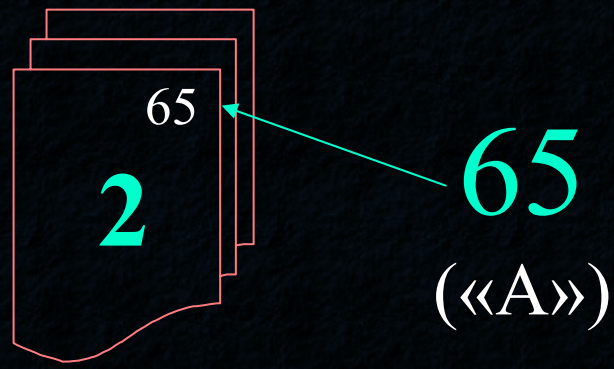
```
    have = 0;
    totProt = 0;
    while ( scanf("%s", s) != EOF)
        {
            i=0;
            while( s[i] != 0 )
                {
                    if ( symbols[ (int)(s[i]) ] < 0.0 )
                        printf("Unknown character : %c\n", s[i] );
                    else
                        {
                            triplet[have] = symbols[ (int)(s[i]) ];
                            seqDNA[have] = s[i];
                            have++;
                            if ( have == 3 )
                                {
                                totProt++;
                                base1 = triplet[0];
                                base2 = triplet[1];
                                base3 = triplet[2];
                                printf(" %s  %5d  %s\n", seqDNA, totProt,
                                    names[pointers[base1][base2][base3] ]);
                                have = 0;
                                }
                                    }
            i++;
                                        }
                                            }
    if ( have != 0 )
        printf("INCOMPLETE TRIPLET.\n");
    return(0);
}
```

names[]	pointers[]	triplet[]	symbols[]	Είσοδος
				65 («A»)

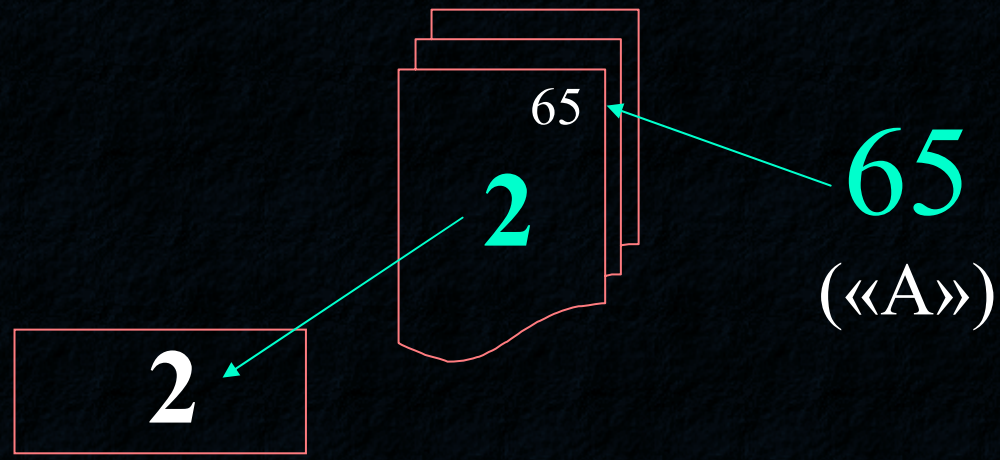
names[] pointers[] triplet[] symbols[] Είσοδος



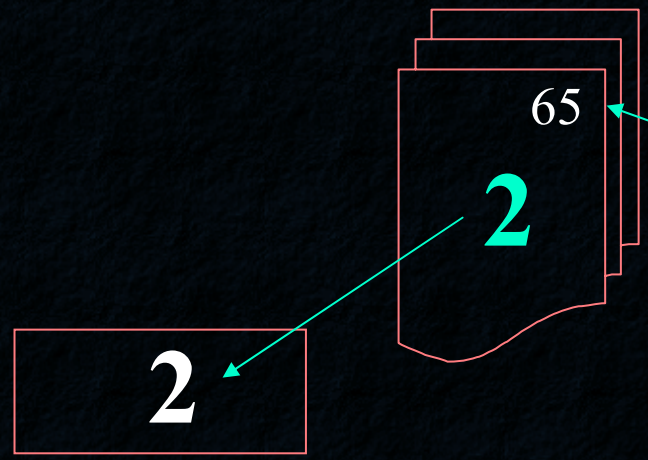
names[] pointers[] triplet[] symbols[] Είσοδος



names[] pointers[] triplet[] symbols[] Είσοδος



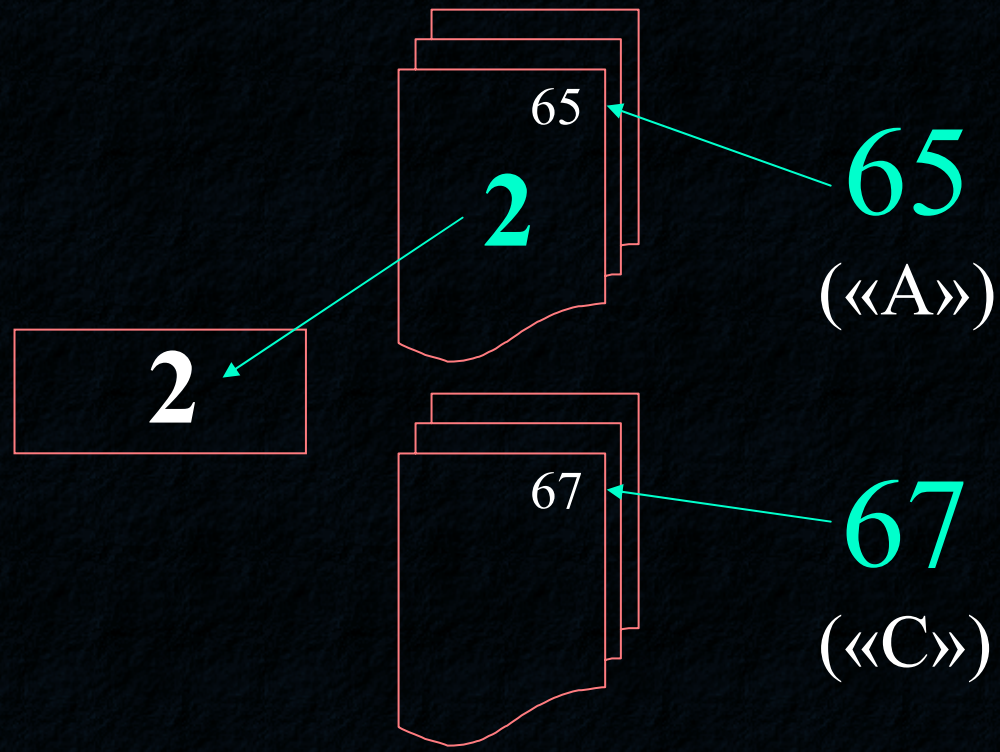
names[] pointers[] triplet[] symbols[] Είσοδος



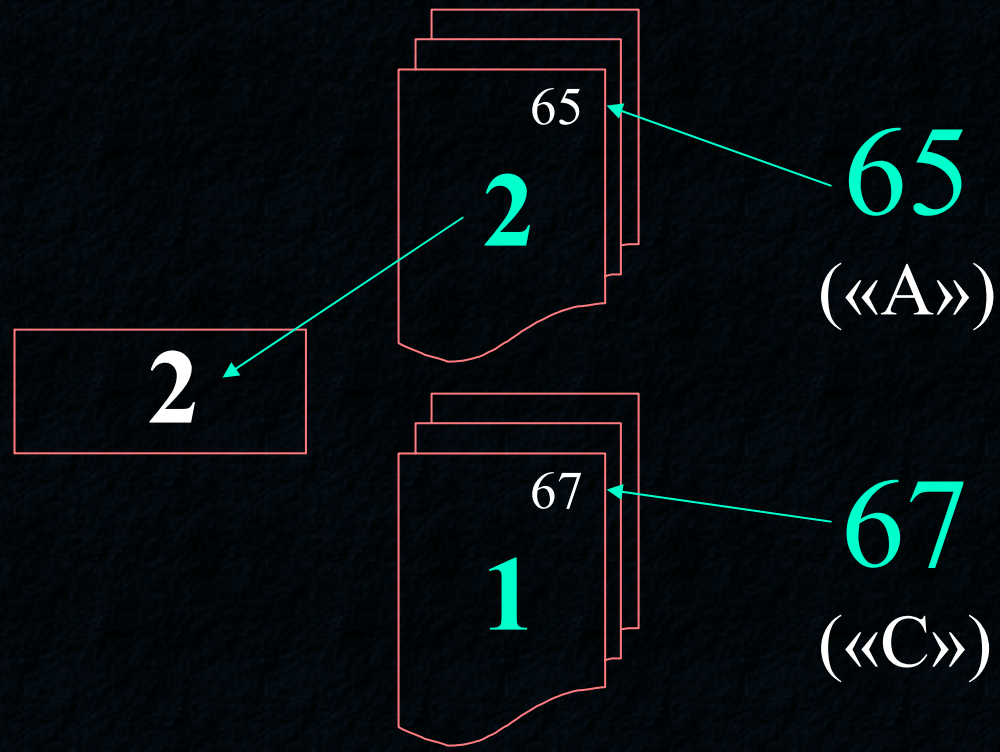
65
(«A»)

67
(«C»)

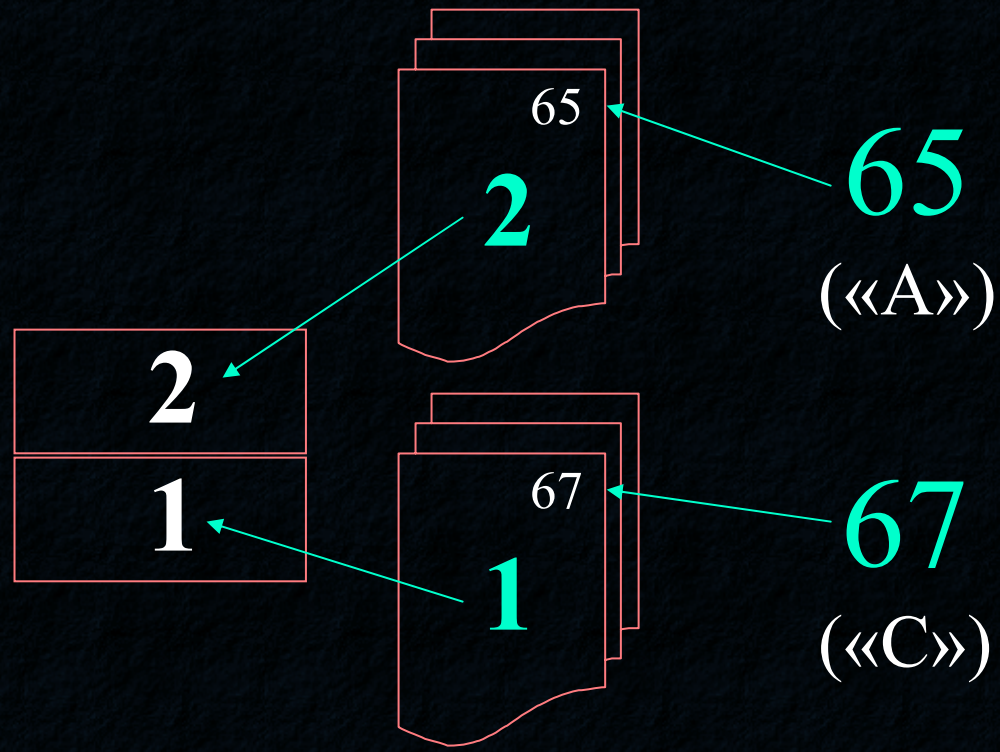
names[] pointers[] triplet[] symbols[] Είσοδος

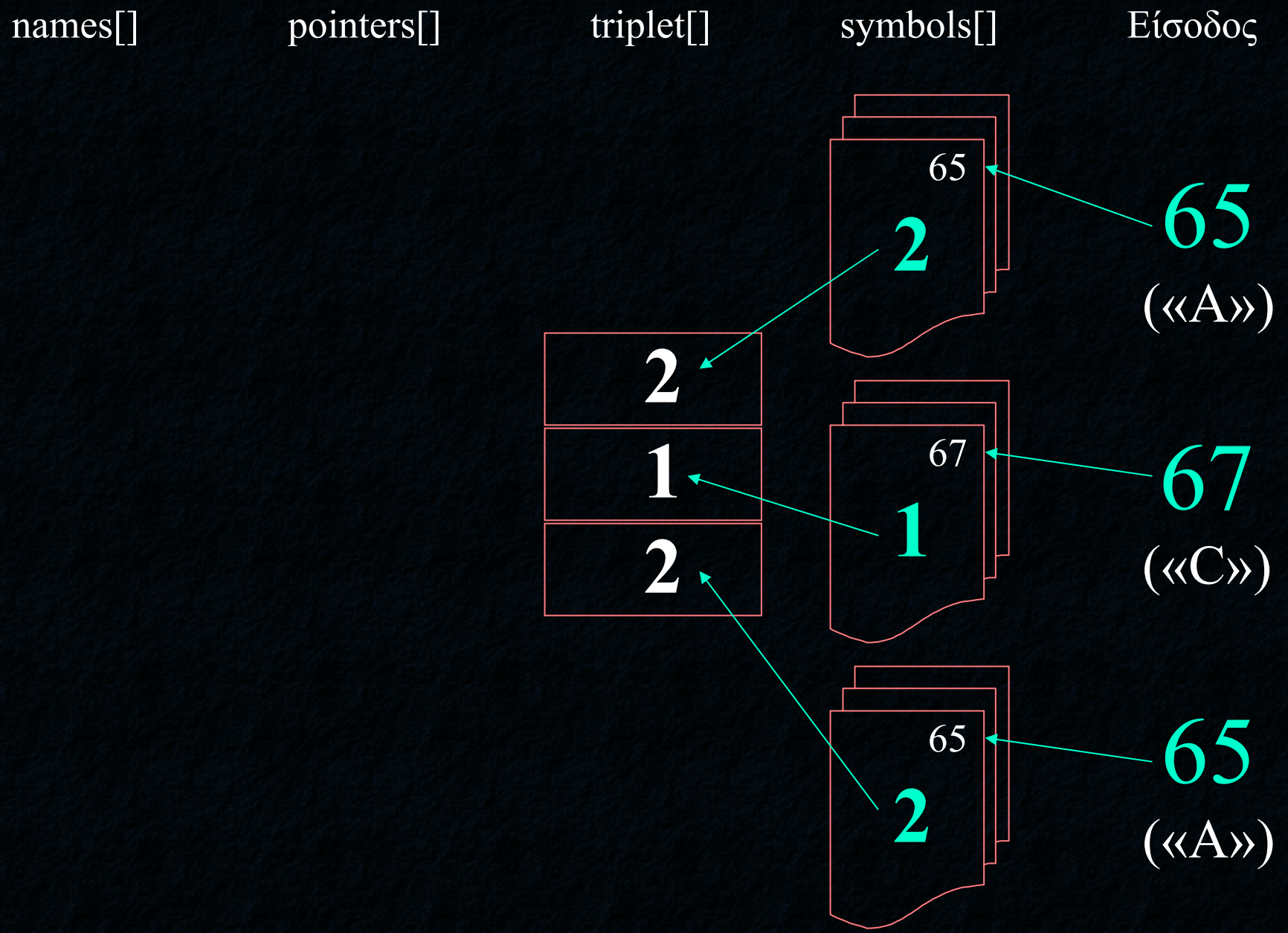


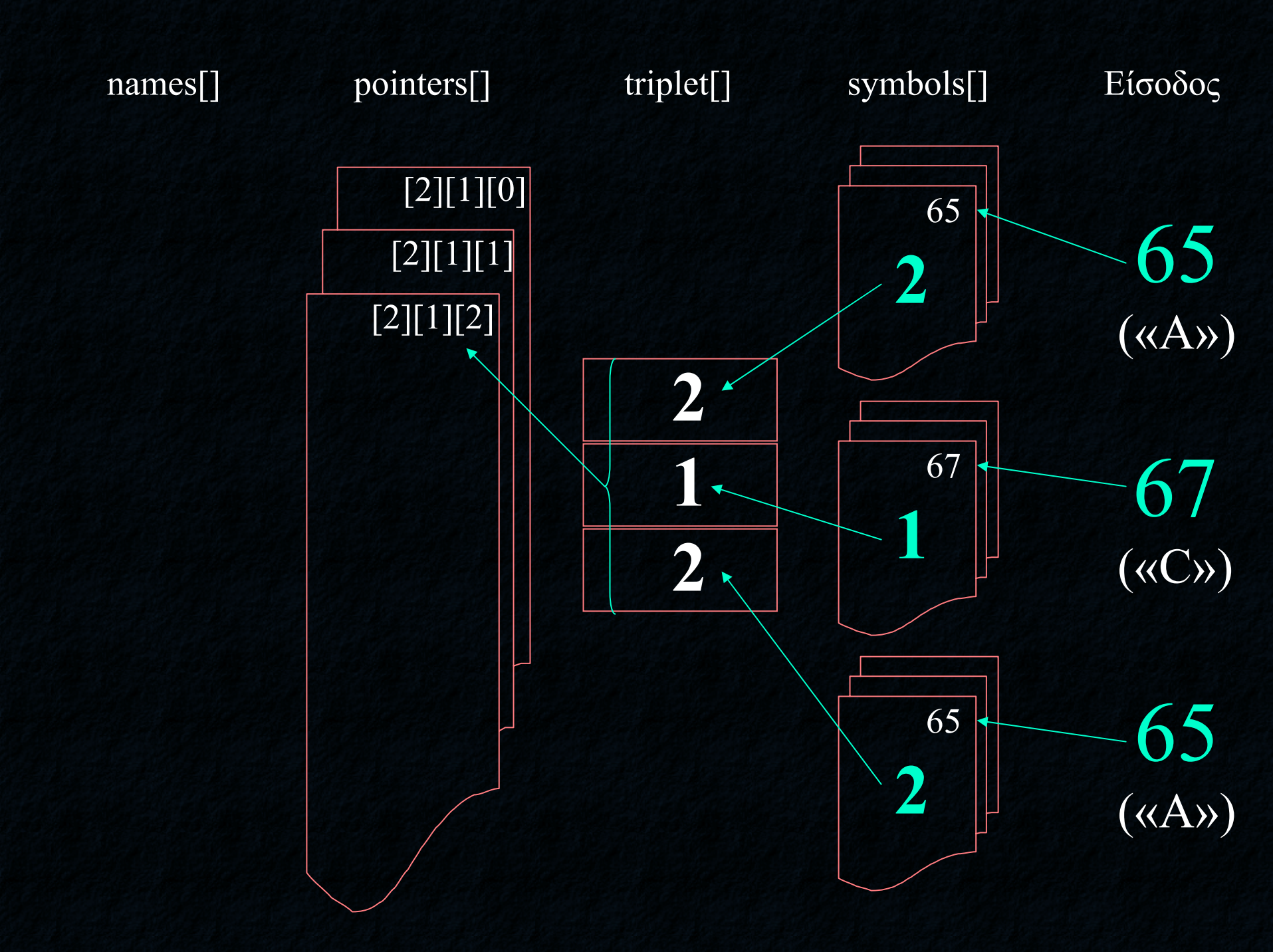
names[] pointers[] triplet[] symbols[] Είσοδος

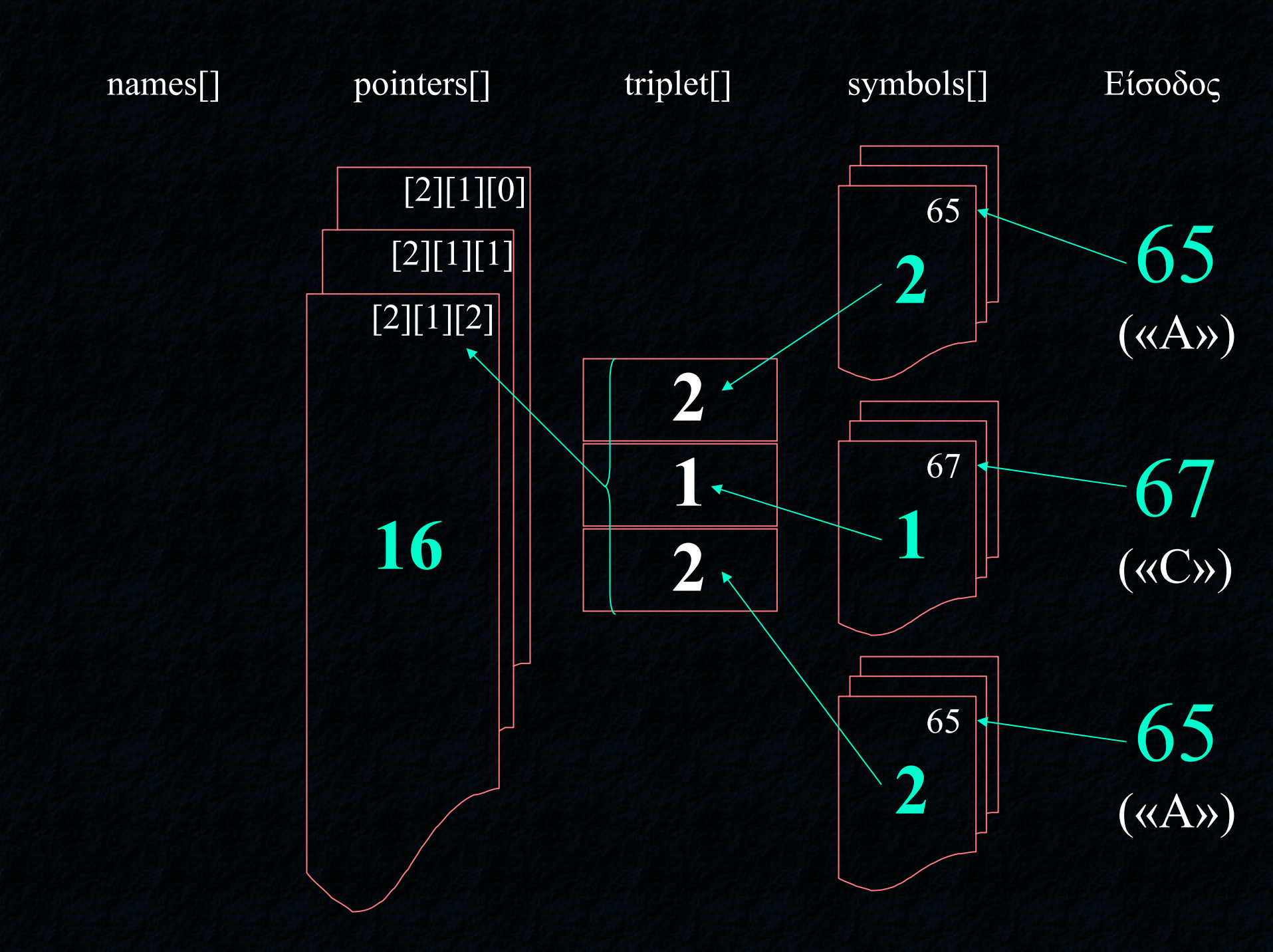


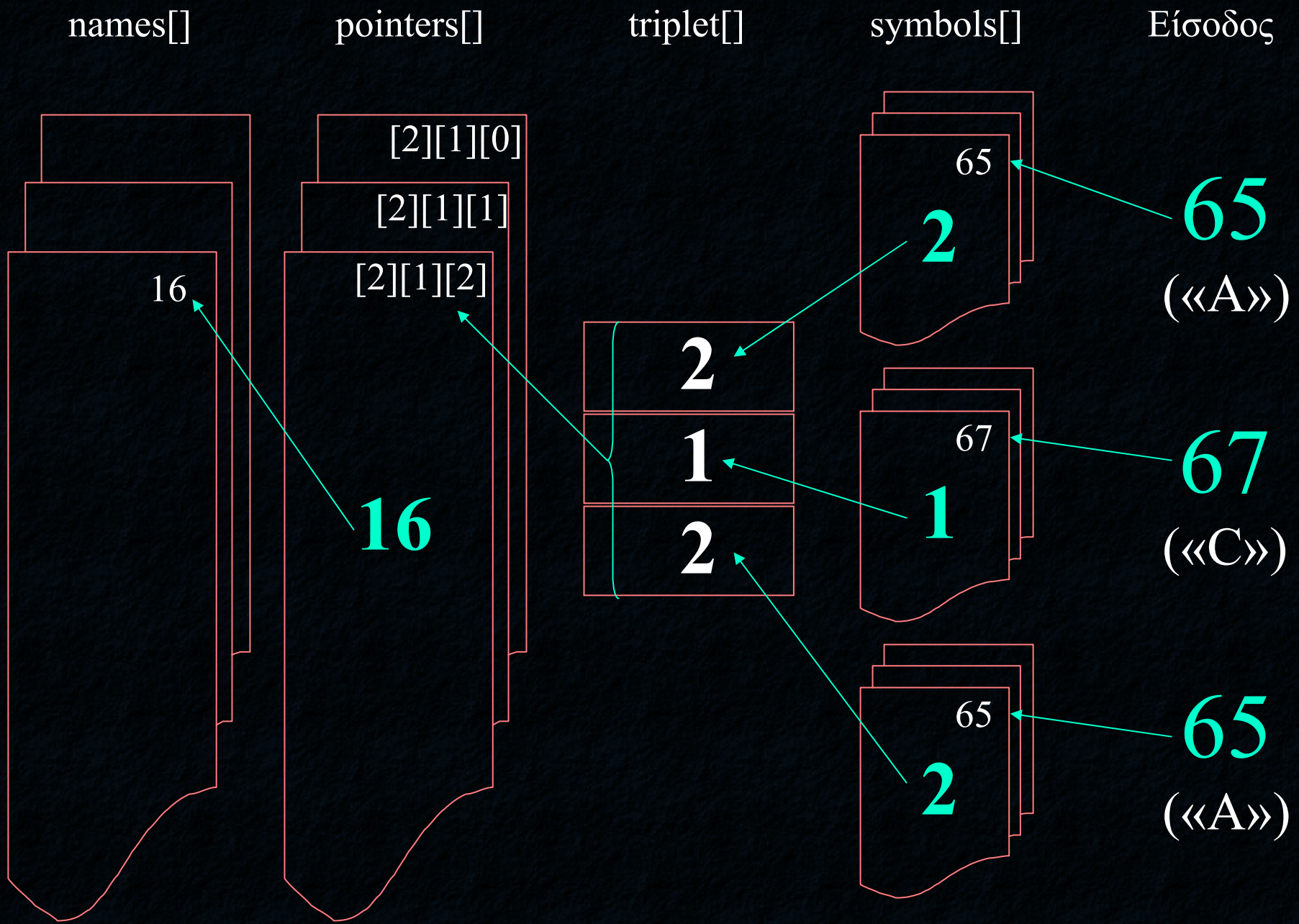
names[] pointers[] triplet[] symbols[] Είσοδος

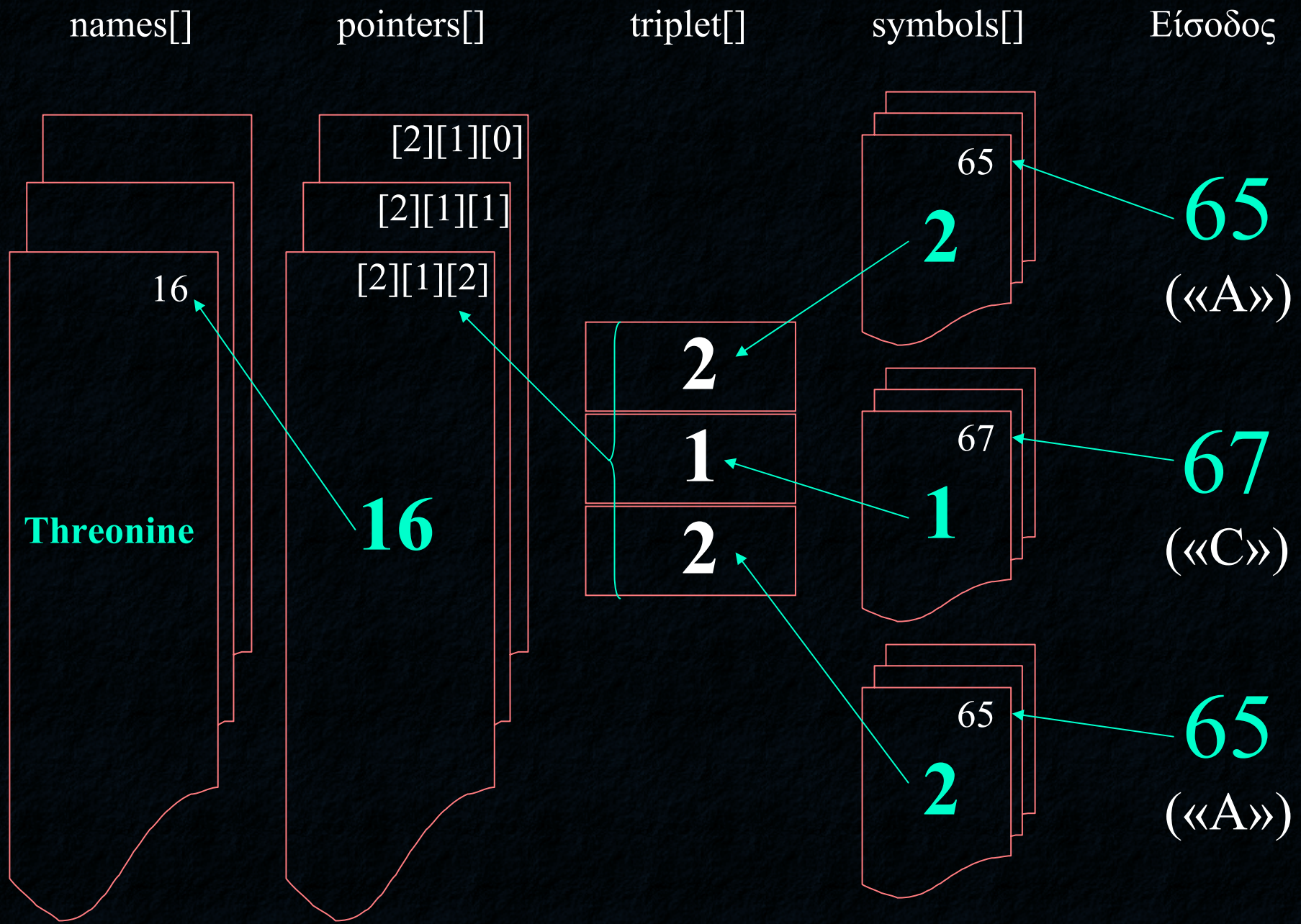


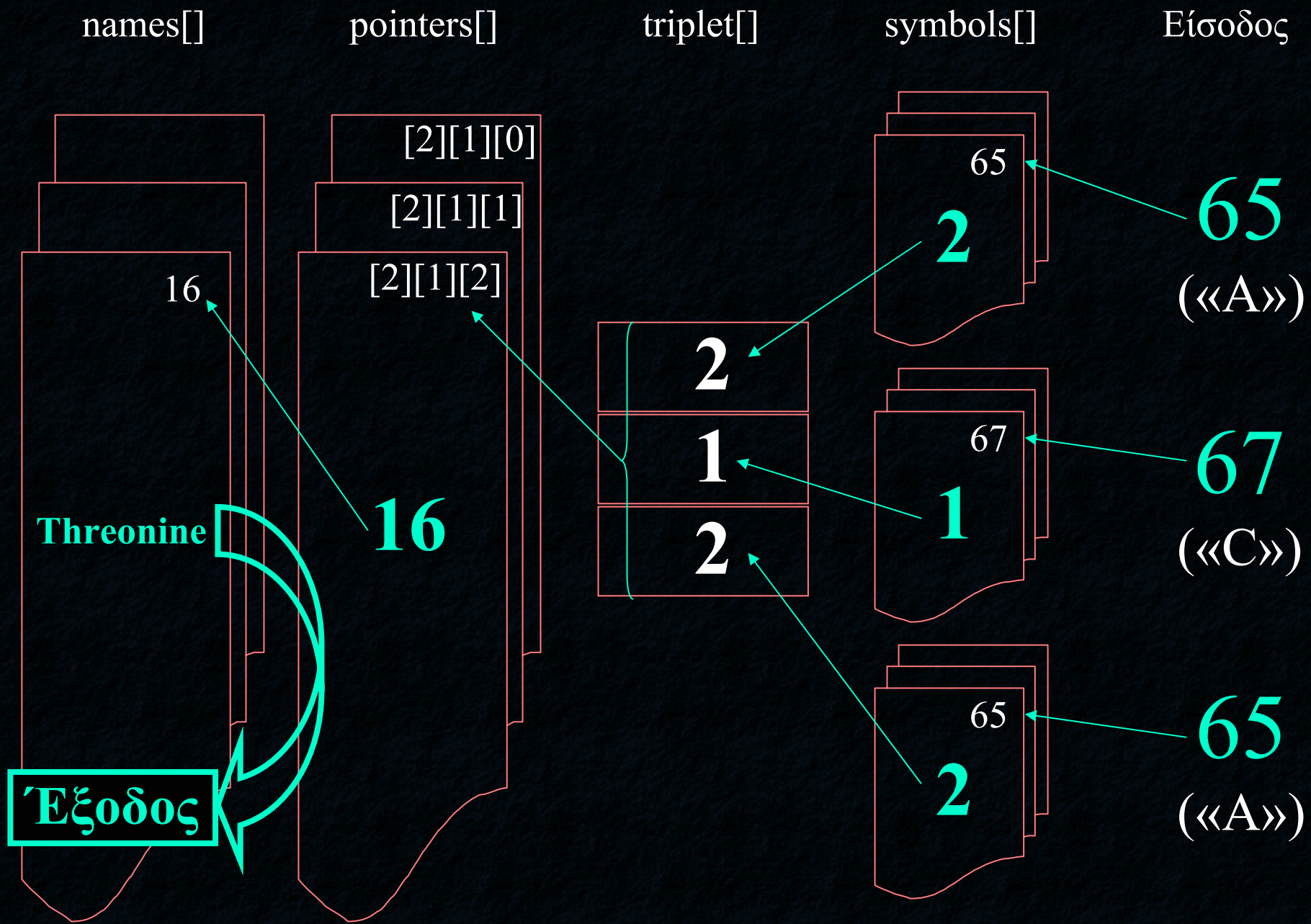

names[]
pointers[]
triplet[]
symbols[]
Είσοδος
65
2
65
(«A»)
2
1
2
67
1
67
(«C»)
65
2
65
(«A»)

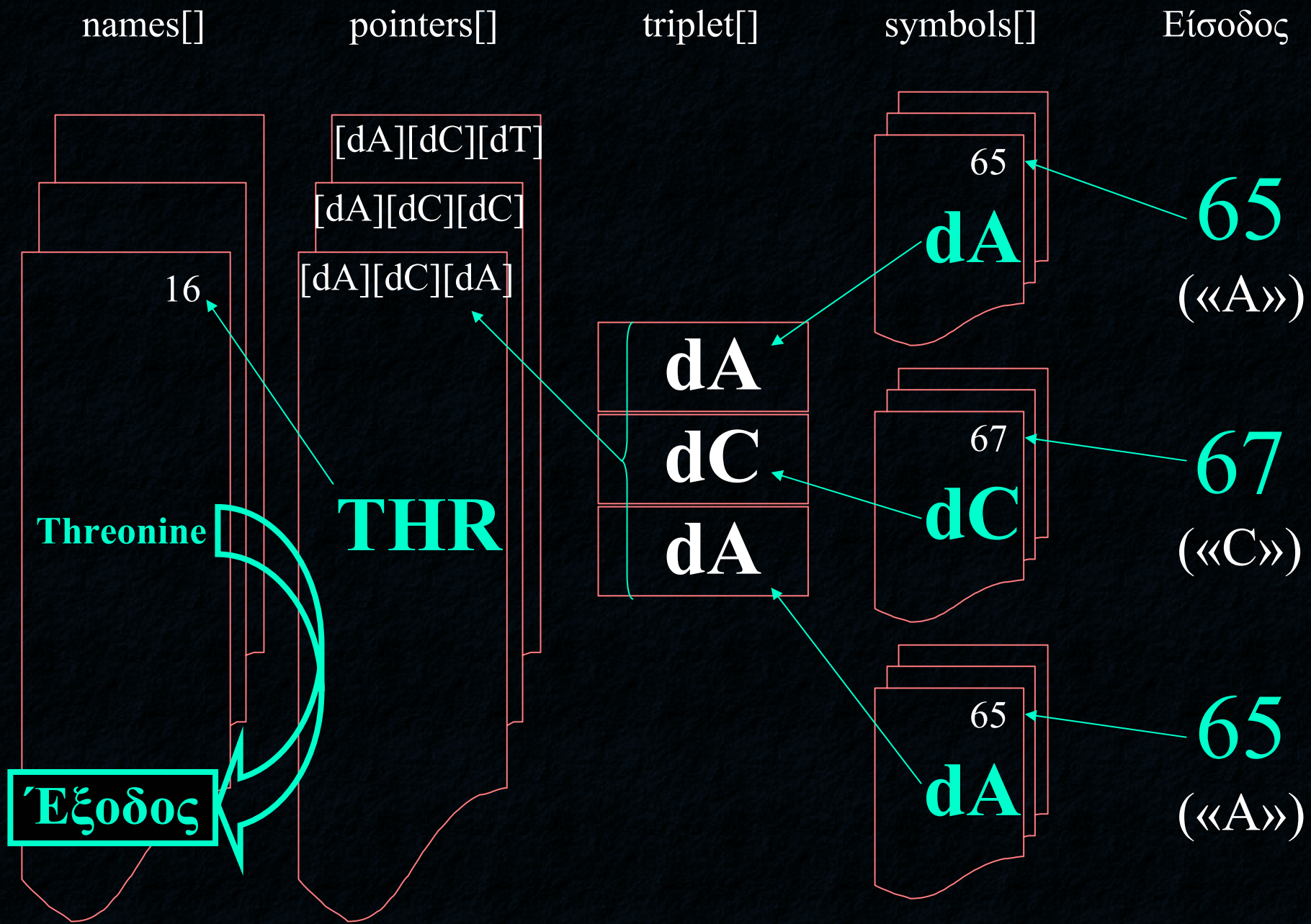

names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[2][1][0]
[2][1][1]
[2][1][2]
2
1
2
65
2
67
1
65
2
65
(«A»)
67
(«C»)
65
(«A»)


names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[2][1][0]
[2][1][1]
[2][1][2]
16
2
1
2
65
2
65
(«A»)
67
1
67
(«C»)
65
2
65
(«A»)


names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[2][1][0]
[2][1][1]
[2][1][2]
16
16
2
1
2
65
2
67
1
65
2
65
(«A»)
67
(«C»)
65
(«A»)


names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[2][1][0]
[2][1][1]
[2][1][2]
16
Threonine
16
2
1
2
65
2
65
(«A»)
67
1
67
(«C»)
65
2
65
(«A»)


names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[2][1][0]
[2][1][1]
[2][1][2]
16
16
Threonine
Έξοδος
2
1
2
65
2
67
1
65
2
65
(«A»)
67
(«C»)
65
(«A»)


names[]
pointers[]
triplet[]
symbols[]
Είσοδος
[dA][dC][dT]
[dA][dC][dC]
[dA][dC][dA]
16
THR
Threonine
Έξοδος
dA
dC
dA
65
dA
65
(«A»)
67
dC
67
(«C»)
65
dA
65
(«A»)

Εφαρμογή 4η :
?

```
#include <stdio.h>

main()
{
    char    s[1000];
    char    t[10000];
    int     i, k, tot;

    if ( scanf("%s", s ) != 1 )
        exit(1);

    tot = 0;
    while ( scanf("%s", t ) != EOF )
        {
            i = 0;
            while( t[i] != 0 )
            {
                k = 0;
                while( s[k] == t[i+k] && s[k] != 0 && t[i+k] != 0 )
                    k++;

                if ( s[k] == 0 )
                    printf(" ----> %5d\n", tot );

                tot++;
                i++;
            }
        }
}
```

```
#include <stdio.h>

main()
{
    char    s[1000];
    char    t[10000];
    int     i, k, tot;

    if ( scanf("%s", s ) != 1 )
        exit(1);

    tot = 0;
    while ( scanf("%s", t ) != EOF )
        {
            i = 0;
            while( t[i] != 0 )
            {
                k = 0;
                while( s[k] == t[i+k] && s[k] != 0 && t[i+k] != 0 )
                    k++;

                if ( s[k] == 0 )
                    printf(" ----> %5d\n", tot );

                tot++;
                i++;
            }
        }
}
```