Προγραμματισμός Ι
Εγγραφές

Κωνσταντίνος Τσερπές
(βασισμένο στις διαφάνειες του κ. Δημήτρη Μιχαήλ)

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Η Ανάγκη Ομαδοποίησης

Πολλές φορές έχουμε πληροφορίες διαφορετικού τύπου οι οποίες όμως έχουν μεγάλη σχέση μεταξύ τους.

Παράδειγμα

Θέλουμε να κρατήσουμε μια συλλογή από φοιτητές όπου ο κάθε φοιτητής έχει

- αριθμό μητρώου
- όνομα
- ηλικία

Εγγραφές

Η C μας παρέχει την δυνατότητα να φτιάξουμε εγγραφές οι οποίες ομαδοποιούν άλλους τύπους.

```
1  struct  student
2  {
3          int   AM ;
4          char  name [ 4 0 ] ;
5          int   age ;
6  };
```

Η δήλωση αυτή φτιάχνει μια εγγραφή η οποία περιέχει 2 ακεραίους και 1 πίνακα με 40 χαρακτήρες.

Αφού ορίσουμε αυτή την εγγραφή μπορούμε να δηλώσουμε μεταβλητές με τύπο αυτήν την εγγραφή.

Εγγραφές

```
1  #include <stdio.h>
2
3  struct student
4  {
5      int AM;
6      char name[40];
7      int age;
8  };
9
10 main()
11 {
12     struct student s1;
13
14     return 0;
15 }
```

Αφού ορίσουμε μια εγγραφή, μπορούμε να δηλώσουμε μεταβλητές με τύπο την εγγραφή αυτή.

Ο κώδικας ορίζει μια μεταβλητή με τύπο την εγγραφή `struct student` και όνομα **s1**.

Εγγραφές

```
1   #include <stdio.h>
2
3   struct student
4   {
5       int AM;
6       char name[40];
7       int age;
8   };
9
10  main()
11  {
12      struct student s1;
13
14      return 0;
15  }
```

Δηλώνοντας την μεταβλητή **s1** έχουμε ουσιαστικά δηλώσει μια μεταβλητή που περιέχει 2 ακεραίους και 1 πίνακα 40 χαρακτήρων.

Αρχικοποίηση Εγγραφών

- Όπως όλες οι μεταβλητές στην C οι εγγραφές δεν αρχικοποιούνται κατά την δήλωση.
- Μπορούμε όμως να τις αρχικοποιήσουμε με τρόπο παρόμοιο με αυτόν που χρησιμοποιούμε για τους πίνακες.

```
1  #include <stdio.h>
2
3  struct student
4  {
5      int AM;
6      char name[40];
7      int age;
8  };
9
10 main()
11 {
12     struct student s1 = {1, "John", 20};
13
14     return 0;
15 }
```

Προσπέλαση Εγγραφών

Οι εγγραφές είναι μια συλλογή μεταβλητών. Τις μεταβλητές αυτές μπορούμε να τις προσπελάσουμε.

Για να προσπελάσουμε μια μεταβλητή σε μια εγγραφή χρησιμοποιούμε το όνομα της μεταβλητής της εγγραφής, μια τελεία και μετά το όνομα της μεταβλητής που θέλουμε να προσπελάσουμε.

Προσπέλαση Εγγραφών

```c
#include <stdio.h>

struct student
{
        int   AM ;
        char  name [40] ;
        int   age ;
};

main ()
{
        struct student s1 = { 1 , "John" , 21};

        printf ( "AM = %d\n" , s1 . AM ) ;
        printf ( "name = %s\n" , s1 . name ) ;
        printf ( "age = %d\n" , s1 . age ) ;

        return 0 ;
}
```

Γράφοντας **s1.name** έχουμε πρόσβαση στην μεταβλητή **name** που βρίσκεται μέσα στην εγγραφή **s1**.

Με τον ίδιο τρόπο μπορούμε να κάνουμε ανάθεση. Για παράδειγμα γράφοντας **s1.age** = 22 μπορούμε να αλλάξουμε την ηλικία.

Προσπέλαση Εγγραφών

```
1  #include <stdio.h>
2
3  struct student
4  {
5      int AM;
6      char name[40];
7      int age;
8  };
9
10 main()
11 {
12     struct student s1 = {1, "John", 21}
13     struct student s2 = {2, "Mary", 23}
14
15     printf("AM = %d\n", s1.AM);
16     printf("name = %s\n", s1.name);
17     printf("age = %d\n", s1.age);
18     printf("\n");
19
20     printf("AM = %d\n", s2.AM);
21     printf("name = %s\n", s2.name);
22     printf("age = %d\n", s2.age);
23
24     return 0;
25 }
```

Δηλώνοντας δύο εγγραφές **student** έχουμε δύο ξεχωριστά σύνολα μεταβλητών, ένα για κάθε φοιτητή.

s1.name είναι το όνομα του 1ου φοιτητή,

του δεύτερου.

Εγγραφές και Πίνακες

Η χρησιμότητα των δομών φαίνεται ακόμη παραπάνω σε συνδυασμό με την χρήση πινάκων.

Μπορούμε να δηλώσουμε π.χ ένα πίνακα με 100 φοιτητές.

```
struct student array [ 1 0 0 ] ;
```

Εγγραφές και Πίνακες

```
1  #include <stdio.h>
2
3  struct student {
4          int AM;
5          char name[40];
6          int age;
7  };
8
9  main()
10 {
11         int i;
12         struct student a[3] = {
13                 {1, "Mary", 21},
14                 {2, "John", 22},
15                 {3, "Helen", 18},
16         };
17
18         for (i = 0; i< 3; i++) {
19                 printf("AM=%d, name=%s, age=%d\n",
20                         a[i].AM, a[i].name, a[i].age);
21         }
22
23         return 0;
24 }
```

Με την χρήση πινάκων μπορούμε να κρατάμε συλλογές από εγγραφές και να τις χειριζόμαστε με επαναλήψεις.

Εγγραφές και Δείκτες

Όπως με όλους τους τύπους, μπορούμε να ορίσουμε ένα δείκτη σε μια εγγραφή.

```
1  #include <stdio.h>
2
3  struct student {
4          int AM;
5          char name[40];
6          int age;
7  };
8
9  main()
10 {
11         struct student s = {1, "Mary", 21};
12         struct student *ptr = &s;
13
14         return 0;
15 }
```

Προσπέλαση Εγγραφών μέσω Δεικτών

Η πρόσβαση όμως των μελών μιας εγγραφής μέσω ενός δείκτη δεν γίνεται πλέον με τον τελεστή μέλους εγγραφής (τελεία), αλλά με τον τελεστή δείκτη εγγραφής (->) όπως φαίνεται παρακάτω.

```
1  #include <stdio.h>
2
3  struct student {
4          int AM;
5          char name[40];
6          int age;
7  };
8
9  main()
10 {
11         struct student s = {1, "Mary", 21};
12         struct student *ptr = &s;
13
14         printf("AM=%d, name=%s, age=%d\n", ptr->AM, ptr->name, ptr->age);
15
16         return 0;
17 }
```

Προτεραιότητα Τελεστών

1. **παρενθέσεις:** (), [], . , −>, **expr**++ ή **expr**−−
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι τελεστές:** +, −, ++**expr**, −−**expr**, !, ∗ και &
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** ∗, /, ή %
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** + ή −
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** <, >, <=, >=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** ==, ! =
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **λογικό AND:** && Από αριστερά προς τα δεξιά.
8. **λογικό OR:** || Από αριστερά προς τα δεξιά.
9. **εκχώρησης:** =, + =, − =, ∗ =, / =, % =
   Από δεξιά προς τα αριστερά.

Αναπαράσταση Εγγραφών στη Μνήμη

Η αναπαράσταση των δομών στην μνήμη είναι ως συλλογή των μεταβλητών που αποτελούνται.

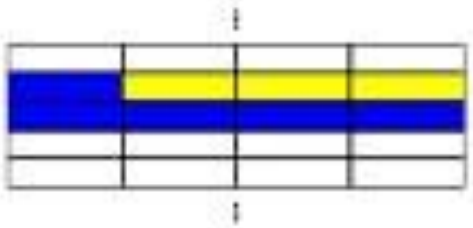
- Πρέπει όμως να παίρνουμε υπόψη μας πως οι υπολογιστές διαβάζουν από την μνήμη σε πολλαπλάσια των 4 bytes (ή των 8 σε 64-bit αρχιτεκτονικές).

- Ο μεταγλωττιστής λοιπόν, για να επιτύχει μέγιστη ταχύτητα προσθέτει πολλές φορές και διάφορα άχρηστα bytes, τα οποία χρησιμεύουν ώστε να υπάρχει σωστή ευθυγράμμιση.

- Στόχος του μεταγλωττιστή είναι για βασικούς τύπους (int, float, κ.τ.λ) να μην ζητάει 2 φορές πληροφορία από την μνήμη.

Αναπαράσταση Εγγραφών στη Μνήμη

Ενώ ένας ακέραιος έχει μέγεθος 4 bytes και ένας χαρακτήρας έχει μέγεθος 1 byte, η παρακάτω εγγραφή



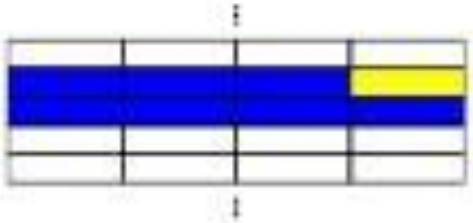
```
struct  test
{
        char  a ;
        int  b ;
};
```

Το πρόβλημα είναι πως ένας ακέραιος πρέπει να ξεκινάει σε πολλαπλάσιο του 4.

έχει μέγεθος 8 bytes, αφού ο μεταγλωττιστής προσθέτει 3 ακόμη μετά τον χαρακτήρα **a**.

Αναπαράσταση Εγγραφών στη Μνήμη

Ενώ ένας ακέραιος έχει μέγεθος 4 bytes και ένας χαρακτήρας έχει μέγεθος 1 byte, η παρακάτω εγγραφή



Το πρόβλημα είναι πως ένας ακέραιος πρέπει να ξεκινάει σε πολλαπλάσιο του 4.

```
struct  test
{
        char  a ;
        char  b ;
        char  c ;
        int  z ;
 };
```

έχει μέγεθος 8 bytes, αφού ο μεταγλωττιστής προσθέτει 1 ακόμη μετά τον χαρακτήρα **c**.

Χρήση Εγγραφών με Συναρτήσεις

Μπορούμε να περάσουμε εγγραφές σε συναρτήσεις όπως οποιαδήποτε άλλη μεταβλητή.

```
1  #include <stdio.h>
2
3  struct student {
4          int AM;
5          char name[20];
6          int age;
7  };
8
9  void print(struct student s)
10 {
11         printf("AM=%d, name=%s, age=%d\n", s.AM, s.name, s.age);
12 }
13
14 main()
15 {
16         struct student s = {1, "John",          20};
17         print(s);
18
19         return 0;
20 }
```

Χρήση Εγγραφών με Συναρτήσεις

Προσοχή όμως γιατί οι εγγραφές χρησιμοποιούν κλήση μέσω τιμής (call by value).

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct student {
5        int AM;
6        char name[20];
7        int age;
8  };
9
10 void setName(struct student t, char *name)
11 {
12       strcpy(t.name, name);
13 }
14
15 main()
16 {
17       struct student s = {1, "John",          20};
18       setName(s, "Fred");
19       printf("name␣=␣%s\n", s.name);
20
21       return 0;
22 }
```

Όλη η εγγραφή αντιγράφεται και οι αλλαγές της συνάρτησης καταστρέφονται με το τέλος της συνάρτησης.

Το πρόγραμμα τυπώνει

name = John

Χρήση Εγγραφών με Συναρτήσεις

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct student {
5        int AM;
6        char name[20];
7        int age;
8  };
9
10 void setName(struct student *t, char *name)
11 {
12       strcpy(t->name, name);
13 }
14
15 main()
16 {
17       struct student s = {1, "John",          20};
18       setName(&s, "Fred");
19       printf("name = %s\n", s.name);
20
21       return 0;
22 }
```

Για κλήση μέσω αναφοράς πρέπει να δώσουμε δείκτη.

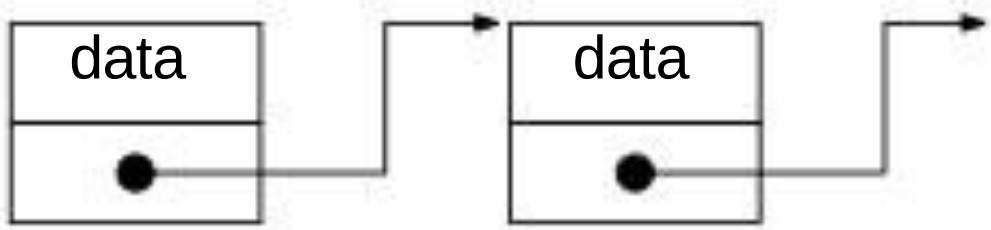
Το πρόγραμμα τυπώνει

name = Fred

Αυτοαναφορά Εγγραφών

Μία εγγραφή δεν μπορεί να περιέχει τον εαυτό της. Μπορεί όμως να περιέχει ένα δείκτη στον εαυτό της.

```
struct  node
{
        int  data ;
        struct  node  * next ;
};
```



Δημιουργία Καινούριων Τύπων (typedef)

Η C μας παρέχει την δυνατότητα να φτιάξουμε δικούς μας τύπους. Για παράδειγμα θέλουμε να φτιάξουμε ένα τύπο φοιτητή ο οποίος να περιέχει τον αριθμό μητρώου, το όνομα και την ηλικία.

```
1
2  struct stud
3  {
4          int AM ;
5          char name [ 4 0 ] ;
6          int age ;
7  };
8
9  typedef struct stud student ;
```

Ουσιαστικά η δεσμευμένη λέξη typedef μας επιτρέπει να δηλώσουμε συνώνυμα τύπων.

Δημιουργία Καινούριων Τύπων (typedef)

Πλέον μπορούμε να γράψουμε απευθείας

```
main ( )
{
        student  s1 ,  s2 ;

        return  0 ;
}
```

για να δηλώσουμε δυο μεταβλητές τύπου **student**.

Δημιουργία Καινούριων Τύπων (typedef)

Μπορούμε να συνδυάσουμε την δήλωση της εγγραφής με την δημιουργία συνώνυμου.

```
typedef  struct  {
        int  AM ;
        char  name [ 4 0 ] ;
        int  age ;
} student ;

main ( )
{
        student  s1 ,  s2 ;

        return  0 ;
}
```

Δημιουργία Καινούριων Τύπων (typedef)

Προσοχή, το typedef μπορούμε να το χρησιμοποιήσουμε για οποιοδήποτε τύπο.

```
typedef  long  myType ;

main ( )
{
      myType  x  =  1 0 0 ;

      return  0 ;
}
```

Παραπάνω δημιουργούμε ένα συνώνυμο με όνομα myType για τον τύπο long.

Δημιουργία Καινούριων Τύπων (typedef)

Προσοχή, το typedef μπορούμε να το χρησιμοποιήσουμε για οποιοδήποτε τύπο.

```
1  #include <stdio.h>
2
3  typedef void (*funcptr)(int);
4
5  void foo(int x)
6  {
7      printf("%d\n", x);
8  }
9
10 int main()
11 {
12     funcptr f = &foo;
13     (*f)(5);
14
15     return 0;
16 }
```

Εδώ δηλώνουμε ένα συνώνυμο με όνομα funcptr για τον τύπο δείκτη σε συνάρτηση που επιστρέφει void και δέχεται ένα ακέραιο.

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Θέλουμε να φτιάξουμε μια μικρή βιβλιοθήκη που να υποστηρίζει την χρήση ρητών (rational) αριθμών.

Οι ρητοί αριθμοί είναι αυτοί που μπορούν να γραφούν ως κλάσμα δύο ακεραίων όπου ο παρανομαστής είναι διάφορος του μηδέν.

Ξεκινάμε με τον ορισμό του τύπου:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5        int a, p;
6  } rational;
```

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Παρέχουμε μια συνάρτηση με παραμέτρους δύο ακεραίους **a** και **p** η οποία μας επιστρέφει τον ρητό αριθμό.

```
7  rational rational_create ( int a , int p )
8  {
9      if ( p == 0) {
10         printf ( "Division by zero!\n" ) ;
11         abort ( ) ;
12     }
13     if ( p < 0) {
14         p = -p;
15         a = -a;
16     }
17     rational x ;
18     x.a = a;
19     x.p = p;
20     return x ;
21 }
```

Η συνάρτηση κάνει έλεγχο ώστε ο παρανομαστής να είναι διάφορος του μηδενός.

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Παρέχουμε επίσης δυο συναρτήσεις:

- μία για την δημιουργία του ρητού μηδέν
- και μία για να βρούμε τον αντίθετο ενός ρητού αριθμού

```
22  rational rational_zero ( )
23  {
24      rational x = { 0 ,      1};
25      return x ;
26  }
27
28  rational rational_minus ( rational a )
29  {
30      a.a   = −a . a ;
31      return a ;
32  }
```

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Στην συνέχεια υλοποιούμε πρόσθεση και αφαίρεση δύο ρητών αριθμών.

```
33  rational  rational_add ( rational  a ,  rational  b )
34  {
35        rational  t ;
36        t.p    = a.p    * b.p;
37        t.a    = a.a    * b.p    + b.a    * a.p;
38        return  t ;
39  }
40
41  rational  rational_subtract ( rational  a ,  rational  b )
42  {
43        return  rational_add ( a ,  rational_minus ( b ) ) ;
44  }
```

Η συνάρτηση της αφαίρεσης για ευκολία χρησιμοποιεί τις συναρτήσεις πρόσθεσης και υπολογισμού του αντιθέτου.

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Παρακάτω φαίνεται η υλοποίηση μιας συνάρτησης που υπολογίζει τον αντίστροφο ενός αριθμού.

```
45  rational  rational_inverse ( rational  a )
46  {
47        if  ( a . a  ==    0)   {
48              printf ( "Zero␣does␣not␣have␣an␣inverse!\n" ) ;
49              abort ( ) ;
50        }
51        rational  x ;
52        x.a    = a.p;
53        x.p    = a.a;
54        return  x ;
55  }
```

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Παρέχουμε επίσης δυνατότητα πολλαπλασιασμού δύο ρητών ή ενός ρητού με έναν ακέραιο.

```
56  rational rational_scalar_multiply ( rational a , int s )
57  {
58      a.a = a.a * s;
59      return a ;
60  }
61
62  rational rational_multiply ( rational a , rational b )
63  {
64      rational t ;
65      t.a = a.a * b.a;
66      t.p = a.p * b.p;
67      return t ;
68  }
```

Παράδειγμα Εγγραφών

Ρητοί Αριθμοί

Τέλος μια συνάρτηση εκτύπωσης και είμαστε έτοιμοι για να χρησιμοποιήσουμε την βιβλιοθήκη μας.

```
69  void rational_print ( rational a )
70  {
71      printf ( "%d/%d" , a . a , a . p ) ;
72  }
73
74  int main ( )
75  {
76      rational r1 = rational_create ( 1 , 5);
77      rational r2 = rational_create ( 1 , 3);
78      rational r3 = rational_add ( r1 , r2 ) ;
79      r3 = rational_scalar_multiply ( r3 , 2);
80      rational_print ( rational_inverse ( r3 ) ) ;
81
82      return 0 ;
83  }
```

Το παραπάνω πρόγραμμα εκτυπώνει τον ρητό αριθμό:

$$\frac{1}{(15+3)\cdot 2}$$

Ενώσεις
union

Πολλές φορές θέλουμε να κρατήσουμε 2 ή περισσότερες μεταβλητές διαφορετικού τύπου, αλλά δεν τις χρειαζόμαστε ταυτόχρονα. Για να μην χάνουμε χώρο, η C μας παρέχει την **ένωση**.

```
union  {
        float  a ;
        int  b ;
        char  str [ 2 0 ] ;
};
```

Η σύνταξη είναι παρόμοια με τις εγγραφές, αλλά τα μέλη μιας ένωσης μοιράζονται τον αποθηκευτικό χώρο.

Στο παραπάνω παράδειγμα μπορούμε να αποθηκεύσουμε είτε κάτι στο **a**, είτε κάτι στο **b**, είτε στο **str**.

Ενώσεις
union

```c
#include <stdio.h>

union example {
        float a;
        int b;
        char str[20];
};

main()
{
        union example var;

        var.a = 5.0;
        var.b = 3;

        // here var.b = 3    but var.a= ??

        return 0;
}
```

Αναθέτοντας κάτι στο **a** και μετά στο **b**, ουσιαστικά γράφουμε επάνω στο **a**.

Ενώσεις
union

Ο προγραμματιστής είναι υπεύθυνος ώστε να γράφει και να διαβάζει σωστά από μια ένωση.

```
#include <stdio.h>

union number {
        int x;
        float y;
};

main()
{
        union number t;

        t.x = 100;
        printf("int: %d\n", t.x);
        printf("double: %f\n", t.y);
        printf("\n");

        t.y = 100.0;
        printf("int: %d\n", t.x);
        printf("double: %f\n", t.y);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
int: 100
double: 0.000000

int: 1120403456
double: 100.000000
```

Προσοχή γιατί μια ένωση μπορεί να αρχικοποιηθεί μόνο με μια τιμή του ίδιου τύπου με το πρώτο μέλος της ένωσης.

```
#include <stdio.h>

union number {
        int x;
        float y;
};

main()
{
        union number t = {2.34};        // ERROR

        printf("int: %d\n", t.x);
        printf("double: %f\n", t.y);
        printf("\n");

        t.y = 100.0;
        printf("int: %d\n", t.x);
        printf("double: %f\n", t.y);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
int: 2
double: 0.000000

int: 1120403456
double: 100.000000
```

Σταθερές Απαρίθμησης

enum

Η C μας παρέχει ένα ακόμη τύπο ορισμένο από τον χρήστη, την **απαρίθμηση**.

Μια απαρίθμηση είναι ένα σύνολο ακέραιων σταθερών που αναπαρίστανται από προσδιοριστικά.

Αυτές οι σταθερές απαρίθμησης είναι, στην ουσία, συμβολικές σταθερές των οποίων οι τιμές μπορούν να τεθούν αυτομάτως.

Σταθερές Απαρίθμησης

enum

Μια απαρίθμηση ορίζεται με την λέξη κλειδί enum.

```
enum months {
        JAN , FEB , MAR , APR , MAY , JUN ,
        JUL , AUG , SEP , OCT , NOV , DEC
};
```

Σταθερές Απαρίθμησης

enum

```
#include <stdio.h>

enum months {
        JAN, FEB, MAR, APR, MAY, JUN,
        JUL, AUG, SEP, OCT, NOV, DEC
};

main ()
{
        enum months month;

        for (month = JAN; month <= DEC; month++)
                printf ("%d\n", month);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
0
1
2
3
4
5
6
7
8
9
10
11
```

Οι σταθερές είναι ουσιαστικά ακέραιοι. Η γλώσσα C ξεκινάει το μέτρημα από το 0.

Σταθερές Απαρίθμησης

enum

Μπορούμε όμως να επηρεάσουμε τις τιμές, π.χ

```
#include <stdio.h>

enum months {
        JAN = 3, FEB, MAR, APR, MAY, JUN,
        JUL, AUG, SEP, OCT, NOV, DEC
};

main()
{
        enum months month;

        for (month = JAN; month <= DEC; month++)
                printf("%d\n", month);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
3
4
5
6
7
8
9
10
11
12
13
14
```

Σταθερές Απαρίθμησης

enum

Μπορούμε μέχρι και διπλές τιμές να δώσουμε:

```
#include <stdio.h>

enum months {
        JAN = 3, JANUARY = 3,
        FEB = 5, FEBRUARY = 5,
        MAR,
        APR = 9, APRIL = 9,
        MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
};

main()
{
        enum months month;

        printf("JAN = %d\n", JAN);
        printf("FEB = %d\n", FEB);
        printf("MAR = %d\n", MAR);
        printf("APR = %d\n", APR);
        printf("MAY = %d\n", MAY);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
JAN = 3
FEB = 5
MAR = 6
APR = 9
MAY = 10
```

Σταθερές Απαρίθμησης

Σφάλμα

```c
#include <stdio.h>

enum months {
        JAN = 3, JANUARY = 3,
        FEB = 5, FEBRUARY = 5,
        MAR,
        APR = 2, APRIL = 2,
        MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
};

main()
{
        enum months month;

        printf("JAN = %d\n", JAN);
        printf("FEB = %d\n", FEB);
        printf("MAR = %d\n", MAR);
        printf("APR = %d\n", APR);
        printf("MAY = %d\n", MAY);

        return 0;
}
```

Το πρόγραμμα τυπώνει

```
JAN = 3
FEB = 5
MAR = 6
APR = 2
MAY = 3
```

Προσοχή γιατί έτσι έχουμε ίδια τιμή π.χ μεταξύ **MAY** και **JAN**.

Σταθερές Απαρίθμησης

Παράδειγμα Χρήσης

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef enum {
5        HEADS = 0,
6        TAILS = 1
7  } side;
8
9  char sidetochar (side s)       {
10       return s == HEADS ? 'H' : 'T';
11 }
12
13 side flipcoin ()       {
14       return (rand () % 2) ? TAILS : HEADS;
15 }
16
17 main ()
18 {
19       int i;
20       for (i = 0; i     < 20; i++)
21             printf ("%2c", sidetochar (flipcoin ()));
22
23       return 0;
24 }
```