Α.Τ.Ε.Ι. ΚΑΛΑΜΑΤΑΣ - ΠΑΡΑΡΤΗΜΑ ΣΠΑΡΤΗΣ

Τμήμα Τεχνολογίας Πληροφορικής & Τηλεπικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΕΥΡΕΣΗΣ ΠΡΩΤΩΝ ΠΑΡΑΓΟΝΤΩΝ ΚΑΙ ΑΛΓΟΡΙΘΜΩΝ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ

Επιβλέπων Καθηγητής: Μακροδημήτρης Γεώργιος

Φοιτητής:

Αριστοτέλης Τριανταφυλλίδης *ΑΜ: 2006051*

Σπάρτη, Νοέμβριος 2012

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ	ΥΠΟΓΡΑΦΕΣ
1.	
2.	
3.	

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς, είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών του Α.Τ.Ε.Ι. Καλαμάτας.

Ο συγγραφέας,

Αριστοτέλης Τριανταφυλλίδης

Ευχαριστίες

Έχοντας φτάσει στο τέλος της πτυχιακής μου εργασίας, αισθάνομαι υποχρεωμένο.. να μιλήσω για κάποιους ανθρώπους, που ο καθένας με τον δικό του τρόπο σημαντοδότησε την πορεία των χρόνων μου στις προπτυχιακές σπουδές μου και να τους ευχαριστήσω.

Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα μου, κύριο Μακροδημήτρη, Επιστημονικό Συνεργάτη του Τμήματος Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών του Α.Τ.Ε.Ι. Καλαμάτας, διότι η συνεργασία μαζί του ήταν ένας καταλύτης για την ολοκλήρωση των προπτυχιακών σπουδών μου. Τα αποτελέσματα της εργασίας αυτής είναι από τη συνεργασία με τον κ. Μακροδημήτρη. Η συνεργασία μας ξεκίνησε όταν ήμουν προπτυχιακός φοιτητής στο χειμερινό εξάμηνο του 2011 – 2012, στο μάθημα «Διαχείριση Έργων Πληροφορικής». Από τη συνεργασία αυτή, είχα την πρώτη εμπειρία στονΠρογραμματισμό. Η πλήρη ηθική στήριξή του και η εμπιστοσύνη του στο πρόσωπό μου, με όπλισαν με κουράγιο, δύναμη και μου έδωσε το θάρρος να αναλάβω την προσπάθεια για τη υλοποίηση των αλγορίθμων πρώτων παραγόντων και αλγορίθμων κρυπτογράφησης.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου Λεωνίδα και Ιωάννα για την αμέριστη υποστήριξή τους όλα αυτά τα χρόνια, των προπτυχιακών σπουδών μου. Αφιερώνω αυτή την εργασία στους γονείς μου, ως ελάχιστη ευγνωμοσύνη για την κατανόηση και την υπομονή τους όλα αυτά τα χρόνια.

Αριστοτέλης Τριανταφυλλίδης

Σπάρτη, Νοέμβριος 2012

Πίνακας περιεχομένων

Εισαγωγή

Σύντομη ιστορία της C

Η C είναι μια διαδικαστική γλώσσα προγραμματισμού γενικής χρήσης η οποία αναπτύχτηκε στις αρχές της δεκαετίας 1970-1980 από τον ΝτένιςΡίτσι στα εργαστήρια Bell Labs για να χρησιμοποιηθεί για την ανάπτυξη του λειτουργικού συστήματος UNIX. Από τότε χρησιμοποιείται ευρύτατα, και ιδιαίτερα για ανάπτυξη προγραμμάτων συστήματος (system software) αλλά και για απλές εφαρμογές. Οι λόγοι της ραγδαίας ανάπτυξης της συγκεκριμένης γλώσσας προγραμματισμού είναι η ταχύτητα της, καθώς και το γεγονός ότι είναι διαθέσιμη στα περισσότερα σημερινά λειτουργικά συστήματα.

Η C είναι μια σχετικά μινιμαλιστική γλώσσα προγραμματισμού. Ανάμεσα στους σχεδιαστικούς στόχους που έπρεπε να καλύψει η γλώσσα περιλαμβανόταν το ότι θα μπορούσε να μεταγλωττιστεί (να γίνεται compile) άμεσα με τη χρήση single-pass compiler — με άλλα λόγια, ότι θα απαιτούνταν μόνο ένας μικρός αριθμός από εντολές (instructions) σε γλώσσα μηχανής (machine language) για κάθε βασικό στοιχείο της, χωρίς εκτεταμένη run-time υποστήριξη. Ως αποτέλεσμα, είναι δυνατό να γραφτεί κώδικας σε C σε low level επίπεδο προγραμματισμού με ακρίβεια ανάλογη της συμβολικής γλώσσας, στην πραγματικότητα η C ορισμένες φορές αποκαλείται (και χωρίς να υπάρχει πάντα αντιπαράθεση) "high-level assembly" ή "portable assembly." Επίσης, γίνονται αναφορές στη C ως mid-level γλώσσα προγραμματισμού.

1 Λεξιλογικά Στοιχεία

Αυτό το κεφάλαιο περιγράφει τα λεξιλογικά στοιχεία που συνθέτουν το πηγαίο κώδικα της C μετά τηνπροεπεξεργασία. Τα στοιχείααυτάονομάζονταιtokens. Υπάρχουν 5 είδητokensστηνC: keywords, identifiers, constants, operators, and separators. Το κενό, το οποίο μερικές φορές χρειάζεται για να διαχωρίσει τα tokens, περιγράφεται και αυτό στο κεφάλαιο.

1.1 Identifiers (αναγνωριστικά)

Οι identifiers είναι ακολουθίες χαρακτήρων που χρησιμοποιούνται για την ονομασία μεταβλητών, τις λειτουργίες, τους νέους τύπους δεδομένων, και για τις μακροεντολές του προεπεξεργαστή. Μπορούν να περιλαμβάνουν γράμματα, δεκαδικά ψηφία, και το χαρακτήρα υπογράμμισης «_». Ο πρώτος χαρακτήρας του αναγνωριστικού ΔΕΝ μπορεί να είναι ένα ψηφίο. Επίσης τα πεζά και τα κεφαλαία γράμματα δεν αναγνωρίζονται ως ίδια, έτσι το foo και το FOO είναι 2 διαφορετικά αναγνωριστικά.

1.2 Keywords (Λέξεις κλειδιά)

Λέξεις-κλειδιά είναι ειδικά αναγνωριστικά που προορίζονται για χρήση ως μέρος της γλώσσας προγραμματισμού. Δεν μπορείτε να τις χρησιμοποιήσετε για οποιοδήποτε άλλο σκοπό.

Εδώ είναι μια λίστα από λέξεις-κλειδιά που αναγνωρίζονται από την ANSI C89:

```
auto break case char const continue default do double else enum extern
float for goto if int long register return short signed sizeof static
struct switch typedef union unsigned void volatile while
```

HISO C99πρόσθεσε τις παρακάτω λέξεις- κλειδιά:

```
inline _Bool _Complex _Imaginary
```

και η GNU επεκτάσεις προσθέτουν αυτές τις λέξεις-κλειδιά:

```
__FUNCTION__ __PRETTY_FUNCTION__ __alignof __alignof__ __asm
__asm__ __attribute __attribute__ __builtin_offsetof __builtin_va_arg
__complex __complex__ __const __extension__ __func__ __imag __imag__
```

```
__inline __inline__ __label__ __null __real __real__
__restrict __restrict__ __signed __signed__ __thread __typeof
__volatile __volatile__
```

1.3 Constants (Σταθερές)

Μια σταθερά είναι μιας κυριολεκτική αξία χαρακτήρα ή αριθμού, όπως το 5 ή το `m'. Όλες οι σταθερές έχουν ένα συγκεκριμένο τύπο δεδομένων. Μπορείτε να καθορίσετε ρητά το είδος μιας σταθεράς, ή να αφήστε τον compiler να χρησιμοποιεί τον προεπιλεγμένο τύπο με βάση την τιμή της.

Στην C υπάρχουν οι παρακάτω τύποι σταθερών :

- Ακέραιες (Integer Constants)
- Χαρακτήρων (Character Constants)
- Πραγματικών Αριθμών (Real Number Constants)
- Συμβολοσειρές(String Constants)

1.3.1 Integer Constants (Ακέραιες σταθερές)

Μια ακέραια σταθερά είναι μια ακολουθία ψηφίων.

Εάν στην ακολουθία των ψηφίων έχει προηγηθεί 0x ή 0X (μηδέν χ ή μηδέν Χ), τότε η σταθερά θεωρείται ότι είναι στο δεκαεξαδικό σύστημα (βάση 16).Οι δεκαεξαδικές τιμές μπορούν να χρησιμοποιούν τα ψηφία από 0 έως 9, καθώς και τα γράμματα α έως f και A έως F.

```
Π.χ:0x2f,0x88,0xAB43
```

Εάν το πρώτο ψηφίο είναι 0 (μηδέν), και ο επόμενος χαρακτήρας δεν είναι «χ» ή «Χ», τότε η σταθερά θεωρείται ότι είναι στο οκταδικό σύστημα (βάση 8). Οι οκταδικές τιμές μπορούν να χρησιμοποιούν μόνο τα ψηφία 0-7. Το 8 και το 9 δεν επιτρέπονται.

```
Π.χ:057,012,0241
```

Σε όλες τις άλλες περιπτώσεις, η ακολουθία των ψηφίων υποτίθεται ότι είναι στο δεκαδικό σύστημα (βάση 10).Οι δεκαδικές τιμές μπορούν να χρησιμοποιούν τα ψηφία 0-9.

```
Π.χ:459, 23901,8
```

Υπάρχουν διάφοροι τύποι ακέραιων δεδομένων, μικρός ακέραιος (short), μεγάλος ακέραιος (long), προσημασμένος (signed) ή μη προσημασμένος (unsigned) ακέραιος. Μπορείτε να αναγκάσετε έναν

ακέραιο να είναι τύπου long και unsigned με μια ακολουθία από ένα ή περισσότερα γράμματα στο τέλος της σταθεράς:

u

U

Με το γράμμα u ή U μπορούμε να υποδηλώσουμε τον μη προσημασμένο ακέραιο

l

L

Ενώ με το γράμμα l ή L μπορούμε να δηλώσουμε το μεγάλο ακέραιο (long)

Για παράδειγμα ο αριθμός 45UL είναι ένας μη προσημασμένος μεγάλος ακέραιος.

1.3.2 Character Constants (Σταθερές χαρακτήρων)

Μια μεταβλητή χαρακτήρα είναι συνήθως ένα μεμονωμένο γράμμα που περικλείεται μέσα σε μονά εισαγωγικά, όπως «Q». Μια σταθερά χαρακτήρα είναι τύπου ακεραίου από προεπιλογή. Κάποιοι χαρακτήρες όπως το ερωτηματικό δεν μπορούν να αναπαραχθούν χρησιμοποιώντας μόνο ένα γράμμα για αυτό λοιπόν υπάρχουν διάφοροι «τρόποι διαφυγής» που μπορείτε να χρησιμοποιήσετε όπως:

\\

Αντίστροφη κάθετος.

\?

Ερωτηματικό.

\'

Μονό εισαγωγικό.

\"

Διπλό εισαγωγικό.

\a

Ηχητική ειδοποίηση.

\b

Χαρακτήρα Backspace.

\e

Χαρακτήρα <ESC>. (Αυτός ο χαρακτήρας είναι μια προέκταση της GNU.)

Για να χρησιμοποιήσετε οποιαδήποτε από αυτές τις ακολουθίες διαφυγής, κλείστε την σε μονά εισαγωγικά, και να το αντιμετωπίσουμε σαν να ήταν οποιοδήποτε άλλο χαρακτήρα. Για παράδειγμα, το γράμμα m είναι «m» και ο χαρακτήρας νέας γραμμής είναι '\ n'.

1.3.3 Real Number Constants (Σταθερές πραγματικών αριθμών)

Μια σταθερά πραγματικού αριθμού είναι μια τιμή που αντιπροσωπεύει μια κλασματικό (floating point) αριθμό. Αποτελείται από μια ακολουθία ψηφίων που αντιπροσωπεύει το ακέραιο (ή "όλο") μέρος του αριθμού, ένα δεκαδικό σημείο, και μία ακολουθία ψηφίων που αντιπροσωπεύει το κλασματικό μέρος.

Είτε το ακέραιο μέρος ή το κλασματικό μέρος μπορεί να παραληφθεί, αλλά όχι και τα δύο. Εδώ είναι μερικά παραδείγματα:

```
doublea, b, c, d, e, f;

a = 4.7;

     b = 4.;

     c = 4;

     d = .7;

     e = 0.7;
```

1.3.4 String Constants (Αλφαριθμητικές σταθερές)

Μια συμβολοσειρά είναι μια ακολουθία από μηδέν ή περισσότερους χαρακτήρες, ψηφία, χαρακτήρες διαφυγής και περικλείεται σε διπλά εισαγωγικά. Όλες οι αλφαριθμητικές σταθερές περιέχουν ένα χαρακτήρα τερματισμού «\0». Οι συμβολοσειρές αποθηκεύονται με την μορφή πίνακα χαρακτήρων και χωρίς συγκεκριμένο μέγεθος. Ο τελεστής τερματισμού βοηθάει της αλφαριθμητικές λειτουργίες επεξεργασίας να γνωρίζουν που τελειώνει η συμβολοσειρά.

Μια συμβολοσειρά δεν μπορεί να περιέχει διπλά εισαγωγικά αφού χρησιμοποιούνται στην αρχή και στο τέλος του αλφαριθμητικού. Αν θέλετε να τα χρησιμοποιήσετε παρ' όλα αυτά μπορείτε να

χρησιμοποιήσετε το backslash και μετά το διπλό εισαγωγικό. Με αυτό τον τρόπο θα καταλάβουμε ότι θέλετε να χρησιμοποιήσετε το διπλό εισαγωγικό ως γράμμα.

Παρακάτω παραθέτουμε μερικά παραδείγματα αλφαριθμητικών :

```
/* Αυτό είναι ένα αλφαριθμητικό */

"Helloworld "

    /* Εδώ χρησιμοποιούμε το backslash για να χρησιμοποιήσουμε τα διπλά
εισαγωγικά */

"\"hello, world!\""
```

Αν το αλφαριθμητικό είναι πολύ μεγάλο για να χωρέσει σε μία γραμμή μπορείτε να χρησιμοποιήσετε το «\»backslash για να το "σπάσετε" σε δύο γραμμές . Ένας άλλος τρόπος για να το επιτύχετε αυτό είναι να χρησιμοποιήσετε τον χαρακτήρα "διαφυγής"\n.

1.4 Operators(Τελεστές)

Ένας τελεστής είναι ένα ειδικό σύμβολο που εκτελεί μια λειτουργία όπως η πρόσθεση ή η αφαίρεση . Θα καλύψουμε σε επόμενο κεφάλαιο την λειτουργία των τελεστών .

1.5 Separators (Διαχωριστές)

Οι διαχωριστές όπως το λέει και το όνομά τους διαχωρίζουν τα tokens. Το κενό είναι ένας διαχωριστής αλλά δεν είναι token.

Παραδείγματα διαχωριστών :

```
( ) { } [ ] ; , . :
```

1.6 White Space (Κενό)

Ως white space μπορούμε να ορίσουμε τα κενά και τις γραμμές που χρησιμοποιούμε για να κάνουμε το πρόγραμμα ευανάγνωστο για κάποιον άλλο.

Για παράδειγμα:

```
#include<stdio.h>

int
```

```
main()
     {
printf( "hello, world\n" );
return 0;
}
```

Και

```
#include<stdio.h>intmain(){printf("hello, world\n");
return 0;}
```

στην ουσία είναι το ίδιο πρόγραμμα.

2 Data Types (Τύποι δεδομένων)

Οι τύποι δεδομένων που θα συναντήσουμε στην γλώσσα C είναι οι εξής:

- Πρωταρχικοί Τύποι
- Δομές
- Πίνακες
- Δείκτες

2.1 Primitive Data Types (Πρωταρχική τύποι δεδομένων)

Στους πρωταρχικούς τύπους δεδομένων εντάσσεται ο :

- Ακέραιος
- Πραγματικός αριθμός

2.1.1 Ακέραιοι τύποι

Οι ακέραιοι τύποι δεδομένων κυμαίνονται σε μέγεθος από 8 bits μέχρι 32 bits. Το πρότυπο C99 επεκτείνει το εύρος αυτό και συμπεριλάβει μεγέθη έως 64 bit. Θα πρέπει να χρησιμοποιείτε μεταβλητές ακέραιου τύπου για την αποθήκευση ακέραιου αριθμού και μεταβλητές χαρακτήρα για την αποθήκευση ενός χαρακτήρα. Τα μεγέθη και τιμές αυτών που αναφέρονται για αυτά τα είδη είναι ελάχιστα. Ανάλογα με την παλαιότητα του υπολογιστή σας, αυτά τα μεγέθη και οι κλίμακες μπορεί να είναι μεγαλύτερες.

Αν και αυτά τα μεγέθη παρέχουν μια φυσική σειρά, το πρότυπο δεν απαγορεύει δυο διαφορετικοί τύποι μεταβλητών να έχουν και διαφορετικό μέγεθος. Για παράδειγμα είναι συνηθισμένο μια int μεταβλητή και μια long να έχουν το ίδιον μέγεθος. Το πρότυπο επιτρέπει ακόμα και string με long να έχουν το ίδιο μέγεθος αν και αυτό είναι κάπως ασυνήθιστο

- Signed char
  Το 8-bit signed char μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από -128 έως 127.
- unsigned char
  Το 8-bit unsigned char μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από 0 ως 255.

- char
  Ανάλογαμετοσύστημάσας,ochartύποςδεδομένωνμπορείναλάβειτιμέςσανναήτανunsignedήsing ed (Παρ' όλα αυτά είναι 3 διαφορετικοί τύποι δεδομένων). Θα πρέπει να χρησιμοποιείτε τον τύπο δεδομένων char για να αποθηκεύεται συγκεκριμένα χαρακτήρες ASCII (όπως το "m") και τους χαρακτήρες διαφυγής (όπως "\n").

- short int
  Το 16-bit short int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από −32,768 έως 32,767. Μπορεί επίσης να αναφερθείτε σε αυτό το είδος και ως short, signed short int, ή singed short

- unsigned short int .
  Το 16-bit unsigned short int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από 0 έως 65,535. Μπορείτε επίσης να αναφερθείτε σε αυτό το είδος και ως unsignted short.

- int
  Το 32-bit int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από −2,147,483,648 ως 2,147,483,647. Μπορείτε επίσης να αναφερθείτε σε αυτό το είδος και ως signed int ή signed.

- Unsigned int
  Το 32-bit unsigned int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από 0 ως 4,294,967,295. Μπορείτε επίσης να αναφερθείτε σε αυτό το είδος και ως unsigned.

- Long int
  Το 32-bit long int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από τουλάχιστον −2,147,483,648 έως 2,147,483,647. (Ανάλογα με το σύστημα σας, αυτός ο τύπος δεδομένων μπορεί να είναι 64-bit, στην οποία περίπτωση το φάσμα είναι ταυτόσημο με εκείνο του longlong int.) Μπορείτε επίσης να αναφερθείτε σε αυτό το είδος και ως long, signed long int, ή signed long.

- Unsigned long int
  Το 32-bit unsigned long int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από τουλάχιστον0 ως 4,294,967,295. ((Ανάλογα με το σύστημα σας, αυτός ο τύπος δεδομένων μπορεί να είναι 64-bit, στην οποία περίπτωση το φάσμα είναι ταυτόσημο με εκείνο του long long int.)Μπορείτε να αναφερθείτε σε αυτό το είδος και ως unsigned long.

- Long long int
  Το64-bit long long int μπορεί να κρατήσει ακέραιες τιμές από −9,223,372,036,854,775,808 ως 9,223,372,036,854,775,807. Μπορείτε να αναφερθείτε σε αυτό το είδος ως long long, signed long long int ή signed long long. Αυτός ο τύπος δεν είναι μέρος του C89, αλλά και οι δύο είναι μέρος του C99 και της GNU C επέκτασης.

- Unsigned long long int
  Το 64-bit unsigned long long int μπορεί να κρατήσει ακέραιες τιμές στην περιοχή από τουλάχιστον 0 ως 18,446,744,073,709,551,615. Μπορείτε να αναφερθείτε σε αυτό το είδος και ως unsigned long long. Αυτός ο τύπος δεν είναι μέρος του C89, αλλά και οι δύο είναι μέρος του C99 και της GNU C επέκτασης.

2.1.2 Real Number Types (Τύποι πραγματικών αριθμών)

Υπάρχουν τρεις τύποι δεδομένων που αντιπροσωπεύουν κλασματικούς αριθμούς. Ενώ τα μεγέθη και τα εύρη των τύπων αυτών είναι ομοιόμορφα στα περισσότερα συστήματα υπολογιστών σε χρήση σήμερα, ιστορικά τα μεγέθη αυτών των τύπων ποικίλλουν από σύστημα σε σύστημα. Ως εκ τούτου, οι ελάχιστες και μέγιστες τιμές είναι αποθηκευμένες σε μάκρο ορισμούς στην βιβλιοθήκη float.h . Σε αυτή την ενότητα, θα περιλαμβάνουν τα ονόματα των μάκρο ορισμούς στη θέση των πιθανών τιμών τους. Ελέγξτε το Float.h του σύστημα σας για συγκεκριμένους αριθμούς.

- `float`
  Ο τύπος δεδομένων float είναι το μικρότερο από τα τρία είδη. Ελάχιστη τιμή της είναι αποθηκευμένη στο FLT_MIN, και δεν πρέπει να είναι μεγαλύτερη από 37e1. Μέγιστη τιμή του είναι αποθηκευμένη στο FLT_MAX, και δεν πρέπει να είναι μικρότερο από 1e37.

- `double`
  Η τύπος δεδομένων double είναι τουλάχιστον τόσο μεγάλος όσο float, και μπορεί να είναι μεγαλύτερος. Ελάχιστη τιμή του είναι αποθηκευμένη στο DBL_MIN, και η μέγιστη τιμή είναι αποθηκευμένη στο DBL_MAX.

- `long double`
  Η τύπος δεδομένων long double είναι τουλάχιστον τόσο μεγάλος όσο float, και μπορεί να είναι μεγαλύτερος. Ελάχιστη τιμή του είναι αποθηκευμένη στο DBL_MIN, και η μέγιστη τιμή είναι αποθηκευμένη στο DBL_MAX.

Όλοι οι τύποι δεδομένων float είναι προσημασμένοι. Προσπαθήστε να χρησιμοποιήσετε ένα unsigned float και θα προκαλέσει compile time error στο πρόγραμμά σας

2.2 Structures (Δομές)

Μία δομή (structure) είναι μια συλλογή μεταβλητών κάτω από ένα συγκεκριμένο όνομα. Αυτές οι μεταβλητές μπορεί να είναι διαφορετικών τύπων και κάθε μία έχει ένα όνομα που βοηθάει στην επιλογή της μέσα από τη δομή. Μία δομή είναι ένας εύκολος τρόπος για ομαδοποίηση διαφορετικών κομματιών με παρόμοιες πληροφορίες.

Μια δομή μπορεί να οριστεί ως τύπος με νέο όνομα, επεκτείνοντας τον αριθμό των διαθέσιμων τύπων. Μπορεί να χρησιμοποιεί ως μέλη της άλλες δομές, πίνακες ή δείκτες, αν και αυτό μπορεί να γίνει πολύπλοκο αν δεν είστε προσεκτικοί.

Ορισμός δομής

Ο τύπος μιας δομής ορίζεται συνήθως κοντά στην έναρξη του αρχείου, δηλώνεται με typedef και ορίζει ένα νέο τύπο, επιτρέποντας τη χρήση του στο πρόγραμμα. Τα typedef βρίσκονται σε ένα αρχείο συνήθως αμέσως μετά τις ορισμούς #define και #include.

Παράδειγμα ορισμού δομής:

```
typedef struct {
char name[64];
char course[128];
int age;
int year;
 } student;
```

Αυτό ορίζει ένα νέο τύπο μεταβλητών student (τύπου student), που μπορούν να δηλωθούν ως ακολούθως.

```
studentst_rec;
```

Παρατηρείστε πόσο παρόμοιο είναι με τον ορισμό ενός int ή float.

Το όνομα της μεταβλητής είναι st_rec, και τα μέλη του ονομάζονται name (όνομα), course(κύκλος σπουδών), age (ηλικία) και year (χρονιά).

Πρόσβαση στα μέλη μιας δομής

Κάθε **μέλος** μιας δομής μπορεί να χρησιμοποιηθεί ακριβώς όπως μια κανονική μεταβλητή, αλλά το όνομά της θα είναι λίγο μεγαλύτερο. Για να επιστρέψουμε στα παραπάνω παραδείγματα, το μέλος name της δομής st_rec θα συμπεριφέρεται όπως ένας πίνακας χαρακτήρων, αναφερόμαστε όμως σε αυτόν με το όνομα

```
st_rec.name
```

Εδώ η τελεία είναι τελεστής (operator) ο οποίος επιλέγει το μέλος μιας δομής.

Δομές ως ορίσματα συναρτήσεων

Μία δομή μπορεί να δοθεί ως όρισμα συνάρτησης ακριβώς όπως κάθε άλλη μεταβλητή. Αυτό φέρνει στην επιφάνεια ορισμένα πρακτικά ζητήματα.

Όπου θέλουμε να αλλάξουμε την τιμή από ένα μέλος της δομής, πρέπει να δώσουμε ως όρισμα ένα δείκτη προς τη δομή. Αυτό είναι παρόμοιο με το να δώσουμε ένα δείκτη (pointer) προς έναν int, αν θέλουμε να αλλάξουμε την τιμή του.

Αν ενδιαφερόμαστε για ένα μέλος της δομής, είναι πιθανόν ευκολότερο να δώσουμε μόνο αυτό το μέλος στη συνάρτηση. Αυτός ο τρόπος είναι εντάξει για απλές συναρτήσεις που εύκολα ξαναχρησιμοποιούνται. Φυσικά αν θέλουμε να αλλάξουμε την τιμή από αυτό το μέλος πρέπει να δώσουμε ένα δείκτη (pointer) προς αυτό.

Όταν δίνεται ως όρισμα μια δομή, κάθε μέλος της δομής αντιγράφεται. Αυτό μπορεί να έχει μεγάλο κόστος αν οι δομές είναι μεγάλες ή αν οι συναρτήσεις καλούνται συχνά. Το πέρασμα δεικτών σε μεγάλες δομές είναι καλύτερη πρακτική σε τέτοιες περιπτώσεις.

Περισσότερα για της δομές

Όπως είδαμε, οι δομές είναι ένας καλός τρόπος για να ομαδοποιούμε δεδομένα που σχετίζονται μεταξύ τους. Είναι επίσης ένας καλός τρόπος για αναπαράσταση από συγκεκριμένους τύπους δεδομένων. Οι μιγαδικοί αριθμοί στα μαθηματικά αναπαριστούνται στο επίπεδο (με μία πραγματική και μια φανταστική διάσταση). Αυτό μπορεί εύκολα να αναπαρασταθεί ως:

```
typedef struct {
        double real;
        Double imag;
} complex;
```

Για κάθε μέλος έχουν χρησιμοποιηθεί double επειδή η ακρίβειά τους είναι μεγαλύτερη από τους float και επειδή οι περισσότερες συναρτήσεις της βιβλιοθήκης μαθηματικών πραγματεύονται αξίες double.

Με παρόμοιο τρόπο, οι δομές μπορεί να χρησιμοποιηθούν για να αναπαραστήσουν σημεία στο χώρο. Εκτός από το να κρατούν δεδομένα, οι δομές μπορούν να χρησιμοποιηθούν ως μέλη άλλων δομών. Είναι δυνατό να έχουμε πίνακες δομών και είναι ένας καλός τρόπος για αποθήκευση δεδομένων με πολλά πεδία (π.χ. βάσεις δεδομένων).

2.3 Arrays (Πίνακες)

Σε όλες τις γλώσσες προγραμματισμού υπάρχει η δυνατότητα ομαδοποίησης των δεδομένων ίδιου τύπου. Συμβατικά ονομάζουμε την δομή που προκύπτει **πίνακα**, θεωρώντας πως τα δεδομένα αποθηκεύονται σε γραμμές και στήλες αν και στην μνήμη η αποθήκευση γίνεται σειριακά. Αναφορά στον πίνακα γίνεται μέσω ενός ονόματος, όπως στις μεταβλητές, ενώ για τα δεδομένα ενός πίνακα

χρησιμοποιείται το όνομα και ένας αριθμός που καθορίζει την θέση του στοιχείου μέσα στον πίνακα. Στην C, η αρίθμηση των πινάκων ξεκινά πάντα από το μηδέν.

Μονοδιάστατοι Πίνακες

Οι μονοδιάστατοι πίνακες είναι οι πιο απλοί πίνακες και αποτελούνται από μια μόνο *γραμμή*.

Σύνταξη

```
data type array Identifier [size];
```

Όπου data type ένας οποιοσδήποτε τύπος δεδομένων (int,float,char, κτλ.).

Όπου array Identifier ένα οποιοδήποτε όνομα για την ονομασία του πίνακα που να υπόκειται στους κανόνες των identifiers.

Και τέλος το size είναι το μέγεθος του πίνακα αρχίζοντας από το 0 και φτάνοντας μέχρι το size-1.

```
/* Δήλωση */
intnumbers[10];
floattemp[30];
char name[15];// Αυτό είναι μια σειρά χαρακτήρων, δηλαδή ένα string.
/* Δήλωση και Αρχικοποίηση */
int numbers[10]={8,0,69,167,349,-428,29}// αρχικοποίηση των θέσεων 0-6, οι
θέσεις 7-9 μένουν κενές
inttemp[]={8,0,69,167,349,-428,29}// δημιουργία πίνακα ακριβώς 7 θέσεων
// Εκχώρηση τιμών
numbers[0]=8;
numbers[1]=69;// κτλ.
```

Πολυδιάστατοι πίνακες

Στην C, θεωρητικά δεν υπάρχει όριο στις διαστάσεις ενός πίνακα, αν και σε ορισμένους μεταγλωττιστές το όριο είναι 256 διαστάσεις. Δημιουργούνται τόσες διαστάσεις όσοι είναι και οι αριθμοί που ακολουθούν στα brackets. Πίνακες μεγαλύτεροι των 2 διαστάσεων δεν μπορούν να εκτυπωθούν σε δύο διαστάσεις.

Σύνταξη

```
datatypearrayIdentifier[size1][size2][size3]...[sizeN];
```

```
int numbers[10][10];// πίνακας 10x10, δηλαδή 100 θέσεων
float temp[2][7][5];// τρισδιάστατος πίνακας 2x7x5, δηλαδή 70 θέσεων
```

2.4 Pointers (Δείκτες)

Οι δείκτες (pointers) δεν χρησιμοποιούνται αποκλειστικά σε συναρτήσεις, αλλά αυτό είναι καλό σημείο να εισάγουμε τον τύπο του δείκτη.

Φανταστείτε ότι έχουμε έναν int που λέγεται i. Η διεύθυνσή του αναπαριστάνεται με το σύμβολο &i. Αν ο δείκτης πρόκειται να αποθηκευθεί ως μεταβλητή θα αποθηκευόταν ως εξής:

int *pi = &i;

int * είναι ο συμβολισμός για ένα **δείκτη σε int** (ακέραιο).

- **&** είναι ο τελεστής που επιστρέφει τη διεύθυνση του ορίσματος. Όταν χρησιμοποιείται, όπως στο &i λέμε ότι κάνει αναφορά στον i (reference).
- Ο αντίθετος τελεστής, ο οποίος δίνει την τιμή στην άλλη άκρη του δείκτη είναι ο *.

Παραδείγματος χάρη χρησιμοποιώντας τον - η διαδικασία είναι γνωστή ως έμμεση αναφορά του pi (dereference) - θα είχαμε:

i = *pi;

Προσοχή να μη μπερδέψετε τις χρήσεις του συμβόλου *, πολλαπλασιασμός, δήλωση δείκτη και έμμεση αναφορά δείκτη (dereference).

Είναι ένα πολύπλοκο θέμα, για αυτό ας το παρουσιάσουμε με ένα παράδειγμα. Η επόμενη συνάρτηση fiddle παίρνει δύο ορίσματα, το x είναι int (ακέραιος) ενώ το y είναι δείκτης σε ακέραιο (pointer σε int). Αλλάζει και τις δύο τιμές.

```
fiddle(intx, int *y)
{   printf ("Arxhthsfiddle: x = %d, y = %d\n", x, *y);
x ++;
    (*y)++;
printf("Telosths fiddle: x = %d, y = %d\n", x, *y);
}
```

αφού το y είναι δείκτης (pointer), χρειάζεται να κάνουμε έμμεση αναφορά πριν αυξήσουμε την τιμή του.

Ένα πολύ απλό πρόγραμμα για να καλέσουμε αυτή τη συνάρτηση μπορεί να είναι το ακόλουθο:

```
main()
{   int i = 0;
int j = 0;

printf("Arxhths main: i = %d, j = %d\n", i, j);
printf("Klhshths fiddle twra\n");
fiddle(i, &j);
printf("Epistrofh (return) apoth fiddle\n");
printf("Telosths main: i = %d, j = %d\n", i, j);
}
```

Παρατηρείστε πως ένας δείκτης σε ακέραιο (pointer to int) δημιουργείται με χρήση του τελεστή & κατά την κλήση **fiddle (i, &j);**

Το αποτέλεσμα της εκτέλεσης του προγράμματος φαίνεται παρακάτω:

```
Arxhths main: i = 0,j = 0
Klhshths fiddle twra
Arxhths fiddle: x = 0, y = 0
Telosths fiddle: x = 1, y = 1
Epistrofh (return) apoth fiddle
Telosths main: i = 0, j = 1
```

Μετά την επιστροφή (return) από τη fiddle η τιμή του i είναι ίδια, ενώ του j που περάστηκε ως δείκτης, έχει αλλάξει.

Συνοψίζοντας, αν θέλετε να χρησιμοποιείτε παραμέτρους για να αλλάξετε την τιμή μεταβλητών σε μια συνάρτηση, αυτές οι παράμετροι πρέπει να δίνονται ως δείκτες και να γίνεται έμμεση αναφορά μέσα στη συνάρτηση.

Όπου η τιμή της παραμέτρου δεν αλλάζει, στη συνάρτηση μπορεί να δίνεται τιμή χωρίς καμία ανησυχία για δείκτες.

3 Operators(Τελεστές)

Η C έχει τέσσερις τύπους τελεστών: **αριθμητικοί, σύγκρισης (συσχεσιακοί), λογικοί και τελεστές χειρισμού bits (bit wise operators).**Παρακάτω δίνουμε ένα πίνακα με όλους τους τελεστές διατεταγμένους σύμφωνα με την προτεραιότητά τους.

Προτεραιότητα	Τελεστές
Υψηλότερη	( ) [ ] ->
	! ~ ++ -- -(type) * &sizeof
	* / %
	- +
	<<>>
	<<= >>=
	== !=
	&
	^
	|
	&&
	||
	?
Χαμηλότερη	= += -= *= /=

Συμβατότητα με εκχωρήσεις

Ο τελεστής εκχώρησης έχει την γενική μορφή

```
<αναγνωριστικό> = <παράσταση>
```

Κατά την εκχώρηση η τιμή της παράστασης μετατρέπεται στον τύπο της μεταβλητής στο αριστερό μέρος της παράστασης. Για παράδειγμα:

```
inti;

char ch;

float f;

void function (void)

{

ch=i; /* ch= ο χαρακτήρας που αντιστοιχεί στο δεύτερο byte του i */

i=f; /* i= το ακέραιο μέρος του f */

f=ch; /* f= ο αριθμός που αντιστοιχεί στο ένα byte του ch */
```

```
f=i; /* f= ο αριθμός που αντιστοιχεί στα δύο bytes του int */

}
```

Όταν σε μία παράσταση υπάρχουν τελεστές διαφορετικών τύπων τότε μετασχηματίζονται στον τύπο του "ισχυρότερου" τελεστή. Στο παρακάτω σχήμα φαίνονται όλοι οι μετασχηματισμοί τύπων που γίνονται κατά τον υπολογισμό της παράστασης

```
r=(ch / i) + (f * d) - (f + i)

------- ------- ------

i d f

-----------

d

-----------------

d
```

Μετασχηματισμός του τύπου μιας παράστασης μπορεί να γίνει προσδιορίζοντας τον νέο τύπο μέσα σε παρένθεση πριν από την παράσταση. π.χ.

```
(τύπος)<παράσταση>
```

Για παράδειγμα

```
int i;

(float) i/2;
```

Τέλος στη C μπορούμε να έχουμε πολλαπλή εκχώρηση x=y=z=0;

Αριθμητικοί τελεστές

-	αφαίρεση, πρόσημο
+	πρόσθεση
*	πολλαπλασιασμός
/	διαίρεση
%	(mod)
--	ελάττωση μεταβλητής κατά 1
++	αύξηση μεταβλητής κατά 1

Οι τελεστές -- και ++ μπορεί να τοποθετηθούν μπροστά ή μετά από ένα τελεσταίο. Η εντολή --x; ισοδυναμεί με την x:=x-1 αλλά η αφαίρεση εκτελείται πριν χρησιμοποιήσουμε την τιμή της x. Όμοια και η εντολή ++x;

Η εντολή x--; ισοδυναμεί με την x:=x-1 αλλά η αφαίρεση εκτελείται αφού χρησιμοποιήσουμε την τιμή της x.

4 Εντολές Ελέγχου

Ένα πρόγραμμα αποτελείται από ένα αριθμό εντολών που συνήθως εκτελούνται σε σειρά. Τα προγράμματα έχουν περισσότερες δυνατότητες αν ελέγχουν τη σειρά εκτέλεσης των εντολών.

Οι εντολές χωρίζονται σε τρεις γενικούς τύπους:

- Ανάθεσης (assignment), όπου τιμές, συνήθως αποτέλεσμα υπολογισμών, αποθηκεύονται σε μεταβλητές.
- Εισόδου / Εξόδου (input / output), δεδομένα διαβάζονται ή εκτυπώνονται.
- Ελέγχου, το πρόγραμμα αποφασίζει τι θα κάνει στη συνέχεια.

Αυτό το κεφάλαιο εξετάζει τη χρήση των εντολών ελέγχου στη C. Θα δείξουμε πως μπορούν να χρησιμοποιηθούν για να γράφουμε προγράμματα με πολλές δυνατότητες:

- Επαναλαμβάνοντας σημαντικά μέρη του προγράμματος.
- Διαλέγοντας ανάμεσα σε διαθέσιμα μέρη από ένα πρόγραμμα.

4.1 Η εντολή if else

Χρησιμοποιείται για να αποφασίσουμε αν θα κάνουμε κάτι σε ένα σημείο ή για να αποφασίσουμε ανάμεσα σε δύο ενέργειες. Το επόμενο test αποφασίζει αν ένας μαθητής πέρασε τις εξετάσεις με βάση 45:

```
if (apotelesma>= 45)
printf ("Perase\n");
else
printf ("Apetyxe\n");
```

Είναι δυνατό να χρησιμοποιούμε το if χωρίς το else.

```
if (thermokrasia< 0)
print ("Pagwnia\n");
```

Κάθε εκδοχή περιλαμβάνει έναν έλεγχο, (αυτό είναι η συνθήκη ανάμεσα στις παρενθέσεις που είναι μετά το if). Εάν η συνθήκη είναι αληθής τότε πραγματοποιείται η επόμενη εντολή. Αν είναι ψευδής τότε ακολουθούμε το else εαν υπάρχει. Μετά από αυτό ακολουθεί το υπόλοιπο του προγράμματος ως συνήθως.

Αν θέλουμε να έχουμε περισσότερες από μια εντολές μετά το if ή το else, πρέπει να είναι ομαδοποιημένες με παρενθέσεις. Μια τέτοια ομαδοποίηση ονομάζεται συμπαγής εντολή ή block.

```
if (result>= 45)
{       printf("Passed\n");
printf("Congratulations\n")
}
else
{       printf("Failed\n");
printf("Good luck in the resits\n");
}
```

Μερικές φορές θα θέλαμε να κάνουμε μια επιλογή ανάμεσα σε πολλές συνθήκες. Ο πιο γενικός τρόπος για να το πετύχουμε είναι να χρησιμοποιήσουμε την παραλλαγή else if της εντολής if. Λειτουργεί κάνοντας πολλές συγκρίσεις. Μόλις κάποια από αυτές δώσει κάποιο αληθές αποτέλεσμα, εκτελείται η επόμενη εντολή και δεν γίνονται περισσότεροι έλεγχοι. Στο επόμενο παράδειγμα δίνουμε βαθμούς ανάλογα με το αποτέλεσμα των εξετάσεων:

```
if (result >= 75)
printf("Passed: Grade A\n");
else if (result >= 60)
printf("Passed: Grade B\n");
else if (result >= 45)
printf("Passed: Grade C\n");
else
printf ("Failed\n");
```

Σε αυτό το παράδειγμα όλες οι συγκρίσεις ελέγχουν μια μόνο μεταβλητή, η οποία ονομάζεται result (αποτέλεσμα). Σε άλλες περιπτώσεις κάθε έλεγχος ίσως να περιλαμβάνει διαφορετική μεταβλητή ή συνδυασμό ελέγχων. Μπορεί να χρησιμοποιηθεί με λιγότερα ή περισσότερα elseif, και το τελευταίο else που είναι μόνο του μπορεί να παραληφθεί. Είναι στην κρίση του προγραμματιστή να ακολουθήσει μια σωστή δομή για κάθε προγραμματιστικό πρόβλημα.

4.2 Η εντολή switch

Είναι μια άλλη μορφή πολλαπλών επιλογών. Είναι καλά δομημένη, αλλά μπορεί να χρησιμοποιηθεί σε συγκεκριμένες περιπτώσεις όπου:

Ελέγχεται μόνο μια μεταβλητή, όλες οι περιπτώσεις πρέπει να στηρίζονται στην τιμή αυτής της μεταβλητής. Η μεταβλητή πρέπει να είναι τύπου integral (int, long, shortorchar). Κάθε πιθανή τιμή της μεταβλητής μπορεί να ελέγξει μόνο μια περίπτωση. Μια τελική, για όλα, default περίπτωση μπορεί να χρησιμοποιηθεί προαιρετικά για να παγιδεύσει όλες τις περιπτώσεις που δεν έχουν καθοριστεί.

Ελπίζουμε ένα παράδειγμα να ξεκαθαρίσει τα πράγματα. Πρόκειται για μια συνάρτηση που μετατρέπει έναν ακέραιο σε μια περιγραφή. Είναι χρήσιμη όταν ενδιαφερόμαστε να μετρήσουμε μια σχετικά μικρή ποσότητα.

```
/* Estimate a number as none, one, two, several, many */
estimate (number)
int number;
{       switch (number) {
case 0 :
printf ("None\n");
break;
case 1 :
printf ("One\n");
break;
case 2 :
printf ("Two\n");
break;
case 3 :
case 4 :
case 5 :
printf ("Several\n");
break;
default :
printf ("Many\n");
break;
        }
}
```

Κάθε ενδιαφέρουσα περίπτωση καταγράφεται μαζί με την αντίστοιχη ενέργεια. Η εντολή break αποτρέπει την εκτέλεση περισσότερων εντολών, εγκαταλείποντας τη switch. Εφόσον οι περιπτώσεις 3 και 4 δεν ακολουθούνται από break συνεχίζουν επιτρέποντας την ίδια ενέργεια να εκτελεστεί για περισσότερες τιμές της μεταβλητής number.

Και η if και η switch επιτρέπουν στον προγραμματιστή να επιλέξει ανάμεσα σε ένα πλήθος ενεργειών.

Ένας άλλος τύπος εντολής ελέγχου είναι ο βρόγχος (loop). Οι βρόγχοι επιτρέπουν σε μια εντολή ή block εντολών να επαναλαμβάνονται. Οι υπολογιστές μπορούν εύκολα να επαναλαμβάνουν μικρές εργασίες αρκετές φορές, ο βρόγχος είναι ο τρόπος της C για να το καταφέρνει.

4.3 Loops (Βρόγχοι)

Στη C υπάρχουν τρεις τύποι βρόγχων (loop), **while**, **do while** και **for**.

- Ο βρόγχος while συνεχίζει να επαναλαμβάνει μια ενέργεια μέχρι ο έλεγχος της συνθήκης να επιστρέψει false (ψευδές) αποτέλεσμα. Είναι χρήσιμος όταν ο προγραμματιστής δεν ξέρει προηγουμένως πόσες φορές θα επαναληφθεί ο βρόγχος.
- Ο βρόγχος do while είναι παρόμοιος, αλλά ο έλεγχος της συνθήκης γίνεται αφού εκτελεστεί το σώμα του βρόγχου. Έτσι είναι σίγουρο ότι το σώμα του βρόγχου θα εκτελεστεί τουλάχιστον μια φορά.
- Ο βρόγχος for χρησιμοποιείται συχνά, συνήθως όταν ο βρόγχος θα επαναληφθεί συγκεκριμένο αριθμό φορών. Είναι πολύ ελαστικός, για αυτό και οι νέοι προγραμματιστές θα πρέπει να προσέχουν ώστε να μην κάνουν κακή χρήση όσων προσφέρει.

Ο βρόγχος while

Ο βρόγχος while επαναλαμβάνει μια εντολή μέχρι ο έλεγχος επάνω να αποδειχτεί false (λανθασμένος). Σε αυτό το παράδειγμα, έχουμε μια συνάρτηση που επιστρέφει το μήκος μιας string. Θυμηθείτε ότι μια string παριστάνεται σαν πίνακας χαρακτήρων που τελειώνουν με το χαρακτήρα null '\0'.

```
int string_length (char string [])
{       int i = 0;

while (string[i] != '\0')
i++;

return(i);
}
```

Η string περνά στη συνάρτηση σαν παράμετρος. Το μέγεθος του πίνακα δεν καθορίζεται, η συνάρτηση θα λειτουργεί με μια string οποιουδήποτε μεγέθους.

Ο βρόγχος while χρησιμοποιείται για να βλέπει τους χαρακτήρες της string (συμβολοσειράς) έναν έναν μέχρι να βρεθεί ο χαρακτήρας null. Τότε γίνεται έξοδος από το βρόγχο και επιστρέφεται ο δείκτης (index) του null. Όσο ο χαρακτήρας δεν είναι null ο δείκτης αυξάνεται και ο έλεγχος επαναλαμβάνεται.

Ο βρόγχος do while

Είναι παρόμοιος με το while, εκτός από το ότι ο έλεγχος γίνεται στο τέλος του σώματος του βρόγχου. Αυτό εγγυάται ότι ο βρόγχος θα εκτελεστεί τουλάχιστον μια φορά πριν να συνεχίσει. Τέτοιο γράψιμο χρησιμοποιείται συχνά όταν είναι να διαβαστούν δεδομένα. Ο έλεγχος τότε επαληθεύει τα δεδομένα και γυρίζει πίσω για να ξαναδιαβάσει αν δεν ήταν αποδεκτά.

```
do
{       printf("Enter 1 for yes, 0 for no :");
```

```
scanf ("%d", &input_value);
} while (input_value != 1 &&input_value != 0);
```

Ο βρόγχος for

Ο βρόγχος for λειτουργεί καλά όπου ο αριθμός των επαναλήψεων είναι γνωστός πριν την είσοδο στο βρόγχο. Η κεφαλή του βρόγχου αποτελείται από τρία μέρη χωρισμένα με ερωτηματικό.

- Το πρώτο εκτελείται πριν την είσοδο στο βρόγχο. Συνήθως δίνει μια αρχική τιμή σε μια μεταβλητή.
- Το δεύτερο είναι μια συνθήκη, ο βρόγχος εξετάζει πότε δεν ικανοποιείται και επιστρέφει false.
- Το τρίτο είναι μια εντολή, η οποία τρέχει κάθε φορά που τελειώνει το σώμα του βρόγχου. Συνήθως είναι μια αύξηση του μετρητή του βρόγχου.

Το παράδειγμα είναι μια συνάρτηση που υπολογίζει το μέσο όρο αριθμών που έχουν αποθηκευτεί σε ένα πίνακα. Η συνάρτηση παίρνει σαν παραμέτρους τον πίνακα και τον αριθμό των στοιχείων του.

```
float average(float array[], int count)
{       float total = 0.0;
int i;

for(i = 0; i < count; i++)
total += array[i];

return(total / count);
}
```

Ο βρόγχος for βεβαιώνει ότι προστίθεται ο σωστός αριθμός στοιχείων πριν τον υπολογισμό του μέσου όρου.

Οι τρεις εντολές στην κεφαλή του βρόγχου (loop) συνήθως κάνουν ένα πράγμα η καθεμία, ωστόσο μπορούν να είναι και κενές. Κενή πρώτη ή τρίτη εντολή σημαίνει ότι δεν υπάρχει αρχικοποίηση ή αύξηση στο μετρητή εκτέλεσης. Αυτό προκαλεί την επ' άπειρο εκτέλεση του βρόγχου, εκτός αν διακοπεί με κάποιο άλλο τρόπο. Ίσως με μια εντολή break.

Είναι επίσης δυνατό να στριμώξουμε πολλές μεταβλητές μαζί στην πρώτη ή στην τρίτη θέση, χωρίζοντάς τες με κόμμα. Αυτό επιτρέπει στο βρόγχο να ελέγχει περισσότερες από μία μεταβλητές. Στο παρακάτω παράδειγμα φαίνεται ο ορισμός ενός τέτοιου βρόγχου, με τις μεταβλητές hi και lo να ξεκινούν από το 100 και το 0 αντιστοίχως και να τείνουν στο ίδιο σημείο.

```
for (hi = 100, lo = 0; hi >= lo; hi--, lo++)
```

Ο βρόγχος for είναι εξαιρετικά ελαστικός και επιτρέπει πολλούς τύπους συμπεριφοράς του προγράμματος να καθορίζονται με απλό και γρήγορο τρόπο.

4.4 Η εντολή break

Έχουμε ήδη συναντήσει την εντολή break στη συζήτηση για την εντολή switch. Χρησιμοποιείται για έξοδο από ένα βρόγχο ή switch, με τον έλεγχο να περνά στην πρώτη εντολή μετά το βρόγχο ή τη switch.

Με τους βρόγχους, η break μπορεί να χρησιμοποιηθεί ώστε να επιβάλει πρόωρη έξοδο από το βρόγχο ή για την υλοποίηση ενός βρόγχου με μια συνθήκη εξόδου στη μέση του βρόγχου. Μια break σε ένα βρόγχο πρέπει πάντα να προστατεύεται από μια εντολή if που παρέχει τη συνθήκη για την προαιρετική έξοδο.

4.5 Η εντολή continue

Είναι παρόμοια με τη break αλλά συναντάται λιγότερο συχνά. Λειτουργεί μόνο μέσα σε βρόγχους με αποτέλεσμα να επιβάλει ένα άμεσο άλμα στη συνθήκη ελέγχου του βρόγχου.

- Σε ένα βρόγχο while, γίνεται άλμα στην εντολή ελέγχου.
- Σε ένα βρόγχο do while, γίνεται άλμα στην εντολή ελέγχου.
- Σε ένα βρόγχο for, γίνεται άλμα στην εντολή ελέγχου, και εκτέλεση της επανάληψης.

Όπως και η break, η continue θα πρέπει να προστατεύεται από μια εντολή if. Είναι απίθανο να τη χρησιμοποιείτε πολύ συχνά.

5 Αλγόριθμοι Πρώτων παραγόντων

5.1 Trial Division

Ο αλγόριθμος trial division είναι ο πιο επίπονος, αλλά πιο εύκολος να καταλάβουμε από τους αλγορίθμους ακέραιας παραγοντοποίησης. Η ευκολία του αλγορίθμου τον κάνει μια από τις βασικές επιλογές για συσκευές με λίγη διαθέσιμη μνήμη (π.χ. γραφικά υπολογιστών) για παραγοντοποίηση ακεραίου αριθμού.

Βασική Ιδέα

Ο αλγόριθμος τεστάρει αν ο αριθμός που του δίνουμε μπορεί να «σπάσει» σε πρώτους αριθμούς

Μεθοδολογία

Δίνουμε έναν ακέραιο n στον αλγόριθμο. Ο αλγόριθμος με την σειρά του ελέγχει αν αυτός ο αριθμός μπορεί να διαιρεθεί με οποιονδήποτε αριθμό. Είναι σαφές το ότι αξίζει να ελέγξει αν διαιρείτε με τιμές οι οποίες είναι μικρότερες από το n και με σειρά από το 2 και πάνω. Διότι ένας αριθμός είναι ποιο πιθανό να διαιρείται με το δύο παρά με το τρία. Με αυτήν την σειρά δεν υπάρχει λόγος να ελέγξει ο αλγόριθμος αν ο αριθμός διαιρείτε με το τέσσερα αν νωρίτερα έχει αποδειχθεί ότι δεν διαιρείτε με το δύο και το ίδιο ισχύει για το τρία και όλα τα πολλαπλάσιά του. Ως εκ τούτου η προσπάθεια μπορεί να μειωθεί αν επιλέξουμε πρώτους αριθμούς ως υποψήφιους παράγοντες.

Επιπλέον, η trial division δεν χρειάζεται να πάει πιο πέρα από $\sqrt{n}$ διότι, αν n διαιρείται με κάποιο αριθμό p, τότε n = p × q και εάν το q ήταν μικρότερο από το p,n θα είχε εντοπιστεί νωρίτερα ως διαιρετέο από το q ή σαν ένας πρώτος παράγοντας .

Μια σύνδεση για τους πρωταρχικούς παράγοντες είναι δυνατή. Ας υποθέσουμε ότι το $P(i)$ είναι ο i-οστός πρώτος αριθμός έτσι ώστε $P(1) = 2$, $P(2) = 3$, $P(3) = 5$,κτλ. Στην συνέχεια ο τελευταίος πρώτος αριθμός που είναι άξιος για να ελέγξουμε ως πιθανός παράγοντας του n είναι ο $P(i)$ όπου $P(i + 1)^2 > n$, σε περίπτωση ισότητας εδώ έχουμε ότι το P(i+1) είναι παράγοντας. Έτσι η δοκιμή με 2, 3 και 5 επαρκεί ως n= 48 (όχι μόνο μέχρι το 25 (τετράγωνο του 5) αλλά μέχρι το 49 διότι το τετράγωνο του επόμενου πρώτου αριθμού είναι το 49), και για αριθμούς κάτω από το n=25. Το 2 και το 3 αρκούν. Πρέπει η τετραγωνική ρίζα του n να είναι ενιαία, τότε είναι ένας παράγοντας και n είναι ένα τέλειο τετράγωνο.

Ταχύτητα αλγορίθμου

Στη χειρότερη περίπτωση, η δοκιμαστική διαίρεση είναι ένας επίπονος αλγόριθμος. Εάν ξεκινά από δύο και λειτουργεί μέχρι την τετραγωνική ρίζα του n, ο αλγόριθμος απαιτεί

$$\pi(\sqrt{n}) \approx \frac{2\sqrt{n}}{\ln n}$$

Trial division, όπου π(x) υποδηλώνει την λειτουργία καταμέτρησης των πρώτων αριθμών. Ο αριθμός των πρώτων είναι λιγότεροι από το χ. Αυτό δεν λαμβάνεται υπόψη ως επιβάρυνση των δοκιμών για την απόκτηση των πρώτων αριθμών ως υποψήφιους παράγοντες. Ένας χρήσιμος πίνακας δεν είναι ανάγκη να είναι παραπάνω από P(3512) = 32749 ο τελευταίος πρώτος αριθμός που ταιριάζει σε ένα 16bito προσημασμένο ακέραιο και P(6542) = 65521 για 16bitous μη προσημασμένους ακεραίους. Αυτό θα αρκούσε για να ελέγξετε για πρώτους αριθμούς για τους αριθμούς μέχρι 65537^2 = 4,295,098,369. Το να προετοιμάσετε έναν τέτοιο πίνακα θα ήταν χρήσιμο μόνο αν είχατε πολλούς αριθμούς για να ελέγξετε. Αν, αντίθετα, χρησιμοποιείται μια παραλλαγή χωρίς δοκιμή πρώτων αριθμών, αλλά απλά διαιρώντας με κάθε περιττό αριθμό μικρότερο από την τετραγωνική ρίζα του n μπορεί να χρειαστεί περίπου

$$\frac{\sqrt{n}}{2}$$

Το οποίο για μεγάλα n το παραπάνω είναι χειρότερο.

Ακόμα κι έτσι, αυτή είναι μια αρκετά ικανοποιητική μέθοδος. Δυσκολία προκύπτει μόνο όταν οι αριθμοί που εξετάζονται είναι μεγάλοι. Τυπικά δεν μιλάμε για το μέγεθος του n αλλά για το μέγεθος των bit του n όπως για παράδειγμα το 1024. Έτσι αν το n είναι ένας αριθμός περίπου 2^{1024} το οποίο είναι περίπου 10^{308} έτσι ώστε οι παράγοντες να είναι 10^{154} ,θα πρέπει να ελεγχθούν, ακόμη και αν έπρεπε να θεωρούνται ως παράγοντες μόνο οι πρώτοι αριθμοί, υπάρχουν περίπου 10^{151}υποψήφιοι. Επιπλέον, επειδή λόγο των τόσο μεγάλων αριθμών που υπερβαίνουν κατά πολύ τα ακέραια μεγέθη των τυπικών ηλεκτρονικών υπολογιστών, οι αυθαίρετες τεχνικές αριθμητικής ακρίβειας που απαιτούνται, έχουν τεράστιο κόστος σε χρόνο για κάθε trial division.

5.2 Pollard Rho

Ο αλγόριθμος Pollard rho είναι ένας ειδικού σκοπού αλγόριθμος για την παραγοντοποίηση ακέραιων .

Βασική ιδέα

Ο αλγόριθμος rho είναι βασισμένος στον αλγόριθμο Floyd'scycle-finding και για την παρατήρηση ότι (όπως στο πρόβλημα γενεθλίων) δύο αριθμούς x και y είναι παρόμοια με το modulo p με πιθανότητα 0.5 μετά από $1.177\sqrt{p}$ αριθμούς που έχουν επιλεχθεί τυχαία. Αν το p είναι παράγοντας του n, όπου

n ο ακέραιος που θέλουμε να παραγοντοποιήσουμε, τότε $p \leq \gcd(x-y,n) \leq n$ δεδομένου ότι ο p διαιρεί $x-y$ και n.

Ο αλγόριθμος rho, παρόλα αυτά, χρησιμοποιεί μια συνάρτηση modulo n ως γεννήτρια μιας ψευδό-τυχαίας ακολουθίας. Τρέχει μία ακολουθία δύο φορές ταχύτερα, από την άλλη. Για παράδειγμα, για κάθε επανάληψη που πραγματοποιείται από ένα αντίγραφο της ακολουθίας, το άλλο αντίγραφο κάνει δύο επαναλήψεις. Έστω x η τρέχουσα κατάσταση της μιας ακολουθίας και y είναι η τρέχουσα κατάσταση της άλλης. Η GCD των | x - y | και n έχει ληφθεί σε κάθε βήμα. Αν αυτό GCD πρόκειται ποτέ να είναι ίσο με n, τότε ο αλγόριθμος τερματίζει με αποτυχία, επειδή αυτό σημαίνει x = y και επομένως, από τον Floyd'scycle-finding αλγόριθμο, η αλληλουχία έχει κάνει τον «κύκλο» της και συνεχίζοντας την περαιτέρω το μόνο που θα καταφέρουμε είναι να επαναλαμβάνουμε προηγούμενο έργο.

Ο αλγόριθμος

Inputs: *n*, ο ακέραιος που θα παραγοντοποιηθεί, και *f*(*x*), η ψευδό-τυχαία συνάρτηση modulo n

Output: ένα σημαντικό παράγοντα του n, ή αποτυχία.

1. $x \leftarrow 2, y \leftarrow 2; d \leftarrow 1$
2. While $d = 1$:
    1. $x \leftarrow f(x)$
    2. $y \leftarrow f(f(y))$
    3. $d \leftarrow \text{GCD}(|x - y|, n)$
3. If $d = n$, return failure.
4. Else, return d.

Σημειώστε ότι ο αλγόριθμος μπορεί να μην βρει τους παράγοντες και θα επιστρέψει false για σύνθετα n. Σε αυτή την περίπτωση, χρησιμοποιήστε ένα διαφορετικό f (x) και δοκιμάστε ξανά. Σημειώστε, επίσης, ότι ο αλγόριθμος αυτός δεν λειτουργεί όταν το n είναι ένας πρώτος αριθμός, δεδομένου ότι, στην περίπτωση αυτή, d θα είναι πάντα 1. Ο αλγόριθμος ονομάζεται έτσι επειδή οι τιμές της f όταν εισέλθει σε μια περίοδο (mod d) και τις διαγραματίσουμε καταλήγουν σε ένα σχήμα ρ.

Πολυπλοκότητα

Ο αλγόριθμος προσφέρει μια εξισορρόπηση μεταξύ χρόνου λειτουργίας και πιθανότητας να βρίσκει έναν παράγοντα. Εάν το n είναι ένα γινόμενο δύο διαφορετικών πρώτων αριθμών ίσου μήκους, που τρέχει τον αλγόριθμο για $O(n^{1/4} \, polylog(n))$ βήματα δίνει έναν παράγοντα με πιθανότητα περίπου το ήμισυ.

5.3 Pollard Rho p -1

OPollardrhop-1 είναι ένας αλγόριθμος παραγοντοποίησης που ανακαλύφθηκε από το John Pollard το 1974. Είναι ένας αλγόριθμος ειδικού σκοπού, που σημαίνει ότι είναι κατάλληλος για ακέραιους με συγκεκριμένο τύπο παραγόντων.

Οι παράγοντες που βρίσκει είναι εκείνοι για τους οποίους ο αριθμός που προηγείται του παράγοντα, $p - 1$, είναι powersmooth. Η ουσιαστική παρατήρηση είναι ότι, μέσω της εργασίας σε πολλαπλασιαστική ομάδα modulo ενός σύνθετου αριθμού N, εργαζόμαστε ταυτόχρονα στις πολλαπλασιαστικές ομάδες modulo όλων των παραγόντων του N.

Η ύπαρξη αυτού του αλγορίθμου οδηγεί στην έννοια των ισχυρών πρώτων αριθμών, όντας πρώτος αριθμός για τα οποία p - 1 έχει τουλάχιστον ένα μεγάλο πρώτο παράγοντα. Όλοι οι πρώτοι αριθμοί είναι δυνατοί. Είναι πολύ πιθανό αν ένας πρώτος αριθμός χρησιμοποιηθεί στην κρυπτογραφία και αποδειχτεί ότι δεν είναι δυνατή η κρυπτογράφηση αυτή, να φταίει κάτι άλλο παρά η τυχαία δημιουργία αριθμών που χρησιμοποιεί ο αλγόριθμος.

Βασική ιδέα

Έστω n είναι ένας σύνθετος ακέραιος με πρωταρχικό παράγοντα p. Από το θεώρημα του Ferma, γνωρίζουμε ότι για όλους τους ακέραιους σχετικά πρώτους του p και για όλους τους θετικούς ακέραιους K:

$$a^{K(p-1)} \equiv 1 \pmod{p}$$

Αν ένας αριθμός χ είναι σύμφωνος με 1 modulo παράγοντα του n, τότε το gcd (x - 1, n) θα είναι διαιρετό από το στοιχείο αυτό.

Η ιδέα είναι να καταστεί ο εκθέτης ένα μεγάλο πολλαπλάσιο του p − 1, γενικά επιλέγουμε όλους τους δυνατούς πρώτους αριθμούς μικρότερους από ένα όριο B, καθιστώντας μια σειρά με πάρα πολλούς πρωταρχικούς παράγοντες. Ξεκινώντας με ένα τυχαίο x και αντικαθιστώντας το επανειλημμένα με $x^w \mod n$, όπου το w κυμαίνετε μέσα σε αυτούς του δυνατούς πρώτους αριθμούς που έχουμε επιλέξει. Ελέγξτε σε κάθε στάδιο, ή μία φορά στο τέλος, αν προτιμάτε, κατά πόσον το gcd (x - 1, n) δεν είναι ίσο με 1.

Αλγόριθμος και χρόνος λειτουργίας

Ο βασικός αλγόριθμος μπορεί να γραφτεί ως εξής :

Inputs: *n*: ένας σύνθετος ακέραιος

Output: ένα σημαντικό παράγοντα του n, ή αποτυχία.

1. select a smoothnessbound *B*
2. $M \leftarrow \prod_{\text{primes } q \leq B} q^{\lfloor \log_q B \rfloor}$
3. randomly pick *a* coprimeto *n* (note: we can actually fix *a*, random selection here is not imperative)
4. $g \leftarrow \gcd(a^M - 1, n)$ (note: the powering can be done mod *n*)
5. if $1 < g < n$ then return *g*
6. if $g = 1$ then select a higher *B* and go to step 2 or return failure
7. if $g = n$ then go to step 2 or return failure

Αν g = 1 στο βήμα 6, αυτό δείχνει για κάθε p - 1, ότι κανένα δεν ήταν B-powersmooth. Αν g = n στο βήμα 7, αυτό συνήθως δείχνει ότι όλοι οι παράγοντες ήταν B-powersmooth, αλλά σε σπάνιες περιπτώσεις μπορεί να υποδηλώνουν ότι είχε μια μικρή σειρά modulo n.

Ο χρόνος εκτέλεσης του αλγορίθμου είναι $O(B \times \log B \times \log^2 n)$, για μεγαλύτερες τιμές του B ο αλγόριθμος τρέχει πιο αργά, αλλά είναι πιο πιθανό να παράγει ένα παράγοντα.

6Αλγόριθμοι κρυπτογράφησης

6.1 RSA

Ο αλγόριθμος RSA δημοσιεύτηκε το 1978 και οφείλει το όνομά του στα αρχικά των επιθέτων των δημιουργών του Ron Rivest, Adi Shamir και Leonard Adleman. Σήμερα, είναι ίσως ο ευρύτερα χρησιμοποιούμενος αλγόριθμος κρυπτογραφίας δημόσιου κλειδιού. Μια σειρά defacto προτύπων, διαμορφωμένων από τα RSA Laboratories, γνωστών ως PKCS (Public Key Cryptography Standards), περιγράφουν τόσο τον αλγόριθμο όσο και τη χρήση του στην πράξη, καθώς και άλλα θέματα σχετικά με την υλοποίησή του.

Δημιουργία κλειδιών

Κάθε χρήστης Α ακολουθεί τα παρακάτω βήματα για την δημιουργία του δημόσιου και ιδιωτικού του κλειδιού.

1. Επιλέγει δυο ισομήκεις στη δυαδική τους αναπαράσταση, διαφορετικούς, μεγάλους, τυχαίους πρώτους αριθμούς p και q, τέτοιους ώστε η διαφορά (p-q) να είναι επίσης μεγάλος αριθμός.
2. Υπολογίζει το γινόμενο n= pq.
3. Υπολογίζει την τιμή της συνάρτησης Euler, φ(n)=(p-1)(q-1).
4. Επιλέγει έναν αριθμό e, τέτοιον ώστε να είναι σχετικά πρώτος με το φ(n) και μεγαλύτερος του 1
5. Υπολογίζει τον αριθμό d, τέτοιον ώστε d*e =_ 1 modφ(n). Ο υπολογισμός αυτός συνήθως γίνεται με χρήση του εκτεταμένου αλγορίθμου του Ευκλείδη.
6. Το δημόσιο κλειδί του Α είναι το (n, e) και ιδιωτικό το d.

Υπενθυμίζεται ότι η συνάρτηση Euler είναι μια αριθμοθεωρητική συνάρτηση η οποία ορίζεται στους φυσικούς αριθμούς. Για κάθε φυσικό n, η φ(n) δίνει το πλήθος των φυσικών αριθμών οι οποίοι είναι μικρότεροι ή ίσοι με το n και οι οποίοι είναι σχετικά πρώτοι (έχουν δηλαδή μέγιστο κοινό διαιρέτη τη μονάδα) με το n. Αν ο n είναι πρώτος, τότε φ(n)=n -1 . Επειδή δε η συνάρτηση Euler έχει την πολλαπλασιαστική ιδιότητα, αν n = pq, τότε, φ(n)=φ(p)* φ(q).

Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης για το σύστημα RSA είναι οι εξής:

Κρυπτογράφηση

Ο χρήστης Α κρυπτογραφεί ένα μήνυμα m και το στέλνει στον Β

1. Λαμβάνει το δημόσιο κλειδί (n, e) του Β

2. Μετατρέπει το μήνυμα m σε έναν ακέραιο στο διάστημα {0,1,.....,n-1}. Μπορεί να χρησιμοποιηθεί οποιαδήποτε προτυποποιημένη διαδικασία μετατροπής (π.χ. τον κώδικα ASCII). Μεγάλου μήκους μηνύματα κατατέμνονται σε μικρότερα.
3. Υπολογίζει την τιμή $c= m^e \mod n$ και τη στέλνει στο Β

Αποκρυπτογράφηση

Ο χρήστης Β αποκρυπτογραφεί το μήνυμα c που έλαβε από τον Α.

Χρησιμοποιώντας το ιδιωτικό του κλειδί d υπολογίζει την ποσότητα $c^d \mod n$, που ισούται με το αρχικό μήνυμα m.

Είναι εύκολο να δούμε ότι η μέθοδος λειτουργεί σωστά, δηλαδή ανακτά το αρχικό μήνυμα από το κρυπτοκείμενο, με χρήση της παραπάνω αλγοριθμικής διαδικασίας. Πράγματι, έστω ότι $m' = c^d \mod n$. Αλλά από τη διαδικασία κρυπτογράφησης ισχύει ότι $c=m^e \mod n$. Επομένως, με αντικατάσταση παίρνουμε ότι $m'=c^d \mod n =_ (m^e)^d \mod n$. Εξ ορισμού όμως των d και e ισχύει ότι $de=_ 1+k\phi(n)$, οπότε $m^{ed} \mod n =_ m^{1+k\phi(n)} \mod n =_ m(m^k)^{\phi(n)} \mod n =_ m \mod n$. Η τελευταία ταυτότητα ισχύει λόγω του θεωρήματος Fermat-Euler, σύμφωνα με το οποίο, για m και n φυσικούς σχετικά πρώτους, ισχύει ότι $m^{\phi(n)}=_ 1 \mod n$. Άρα, $m' = m$. χρησιμοποιώντας δε το Κινέζικο Θεώρημα Υπολογισμού, μπορεί να αποδειχθεί ότι οι εξισώσεις ισχύουν για κάθε m.

Ασφάλεια του RSA

Αν και η παραπάνω περιγραφή του αλγορίθμου RSA φαίνεται απλή, στην πραγματική του εφαρμογή ανακύπτουν διάφορα προβλήματα:

- Για λόγους ασφάλειας, οι πρώτοι αριθμοί p και q πρέπει να επιλέγουν τυχαία και να είναι ισομήκεις στη δυαδική τους αναπαράσταση. Πρώτοι αριθμοί μπορούν να βρεθούν χρησιμοποιώντας κάποιον από τους αλγόριθμους που ελέγχουν αν ένας αριθμός είναι πρώτος (primality test).
- Οι αριθμοί p και q δεν πρέπει να είναι πολύ κοντινοί γιατί στην περίπτωση αυτή η παραγοντοποίηση Fermat του n μπορεί να πετύχει. Για παράδειγμα, αν η διαφορά (p-q) είναι μικρότερη από $2\, n^{1/4}$(αριθμός που ακόμη και για μικρές -1024 bits– τιμές του n είναι 3x1077),η εύρεση των και q είναι εύκολη. Επιπλέον, αν είτε ο (p-1) είτε ο (q-1) έχουν μόνο μικρούς πρώτους παράγοντες, ο n μπορεί να παραγοντοποιηθεί γρήγορα χρησιμοποιώντας τον αλγόριθμος του Pollard. Επομένως και αυτές ι τιμές για του p και q πρέπει να αποφεύγονται.
- Όταν το e είναι μικρός (π.χ. e=3)και για μικρές τιμές του m(δηλαδή τέτοιες ώστε $m<n^{1/e}$), ισχύει ότι $m^e<n$. Στην περίπτωση αυτή, το μήνυμα μπορεί εύκολα να ανακτηθεί από το κρυπτόγραμμα υπολογίζοντας την e-στή ρίζα του κρυπτογράμματος στο σύνολο των ακεραίων.

Προκειμένου να αποφευχθεί το πρόβλημα αυτό, καθώς και μερικά ακόμη, στην πράξη ο RSA μετασχηματίζει το μήνυμα m πριν το κρυπτογραφήσει, προσθέτοντας στο τέλος του έναν αριθμό από bits. Η τεχνική αυτή ονομάζεται παραγέμισμα (padding), υπάρχουν δε πολλοί τρόποι υλοποίησης της. Το πρότυπο PKCS#1 περιγράφει τέτοιες διαδικασίες.

- Είναι σημαντικό ο d να είναι αρκετά μεγάλος. Μπορεί να αποδειχθεί ότι, αν q<p<2q, πράγμα αρκετά συνηθισμένο και $d<n^{1/4}/3$, τότε ο d μπορεί να υπολογιστεί αποδοτικά αν είναι γνωστοί οι n και e. Αν ο e είναι σχετικά μικρός, αφού ο n είναι δημόσια γνωστός, τότε, με την προϋπόθεση ότι δεν χρησιμοποιείται (ή δε χρησιμοποιείται σωστά) παραγέμισμα, ο αλγόριθμος είναι ευάλωτος σε κρυπταναλυτικές επιθέσεις. Η συνήθως χρησιμοποιούμενη τιμή του e είναι e=65537,ενώ μικρότερες τιμές πρέπει να αποφεύγονται.

Η ασφάλεια του RSA βασίζεται στη δυσκολία επίλυσης του «προβλήματος RSA», που ορίζεται ως το πρόβλημα της εύρεσης των e-στών ριζών ενός φυσικού modulo n, όπου n σύνθετος φυσικός, δηλαδή της εύρεσης της τιμής m, δοθείσης της τιμής c, έτσι ώστε να ισχύει $c=m^e \bmod n$, όπου (n,e) είναι ένα δημόσιο κλειδί RSA και c είναι ένα κρυπτόγραμμα RSA. Η καλύτερη σήμερα γνωστή μέθοδος για την επίλυση του προβλήματος αυτού είναι η παραγοντοποίηση του n. Αν ένας επιτιθέμενος κατορθώσει να βρει τους πρώτους παράγοντες του n, μπορεί να υπολογίσει τον d και στη συνέχεια να αποκρυπτογραφήσει το c χρησιμοποιώντας τον αλγόριθμο RSA. Αν και δεν έχει ακόμη βρεθεί μέθοδος που να επιλύει το πρόβλημα αυτό σε πολυωνυμικό χρόνο σε ένα κλασικό υπολογιστή, δεν έχει αποδειχθεί ότι τέτοια μέθοδος δεν υπάρχει.

6.2 Rabin

Το 1979, ο Michael Rabin πρότεινε ένα κρυπτογραφικό σύστημα το οποίο βασίζει την ασφάλειά του στο πρόβλημα της εύρεσης τετραγωνικών ριζών modulo ενός σύνθετου αριθμού. Το συγκεκριμένο πρόβλημα είναι ισοδύναμο με το πρόβλημα της παραγοντοποίησης ακεραίων. Το γεγονός αυτό αποτελεί σημαντικό πλεονέκτημα του κρυπτοσυστήματος Rabin έναντι του RSA, αφού για το πρόβλημα RSA μια τέτοια ισοδυναμία δεν έχει αποδειχθεί. Ωστόσο, το μειονέκτημα που έχει το σύστημα Rabin είναι ότι κάθε κρυπτοκείμενο μπορεί να δημιουργηθεί από τέσσερα πιθανά απλά κείμενα και απαιτείτε επιπλέον πολυπλοκότητα για να βρεθεί το σωστό αρχικό κείμενο.

Αναλυτικότερα, κάθε χρήστης ακολουθεί τον παρακάτω αλγόριθμο για τη δημιουργία των κλειδιών του.

Δημιουργία κλειδιών

Κάθε χρήστης Α ακολουθεί τα παρακάτω βήματα για τη δημιουργία του δημόσιου και ιδιωτικού του κλειδιού

1. Δημιουργεί δύο μεγάλους, τυχαίους πρώτους αριθμούς p και q.
2. Υπολογίζει το γινόμενο n= pq.
3. Το δημόσιο κλειδί του Α είναι το n και ιδιωτικό είναι το ζευγάρι (p,q).

Όπως και στο κρυπτοσύστημα RSA, η μυστικότητα του ιδιωτικού κλειδιού βασίζεται στη δυσκολία επίλυσης του προβλήματος της παραγοντοποίησης ακεραίων. Επομένως, οι ιδιότητες που πρέπει να έχουν οι πρώτοι p και q στο κρυπτοσύστημα Rabin είναι οι ίδιες με αυτές που αναφέρθηκαν για τον RSA. Συγκεκριμένα, οι p και q πρέπει να έχουν το ίδιο μέγεθος σε bits, να είναι ισχυροί πρώτοι και η διαφορά (p-q) θα πρέπει να είναι ένας μεγάλος αριθμός. Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης για το σύστημα Rabin είναι οι εξής:

Κρυπτογράφηση

Ο χρήστης Α κρυπτογραφεί ένα μήνυμα m και το στέλνει στον Β.

1. Λαμβάνει το δημόσιο κλειδί n του Β
2. Μετατρέπει το μήνυμα m σε έναν ακέραιο στο διάστημα {0,1,.....n-1}.
3. Υπολογίζει την τιμή $c= m^2 \mod n$ και την στέλνει στον Β.

Αποκρυπτογράφηση

Ο χρήστης Β αποκρυπτογραφεί το μήνυμα c που έλαβε από τον Α

1. Υπολογίζει τις 4 τετραγωνικές ρίζες του c modulo n.
2. Αποφασίζει με κάποιο τρόπο ποια από τις 4 ρίζες αντιστοιχεί στο αρχικό μήνυμα m.

Υπολογισμός ριζών

Όπως προαναφέρθηκε, το πρόβλημα της εύρεσης τετραγωνικών ριζών modulo έναν σύνθετο αριθμό n είναι ισοδύναμο με το πρόβλημα της παραγοντοποίησης ακεραίων και λύνεται σε εκθετικό χρόνο. Ωστόσο, αν κανείς γνωρίζει τους πρώτους παράγοντες του n, μπορεί να λύσει το πρόβλημα αυτό αποδοτικά . Συγκεκριμένα, αν n=pq και οι πρώτοι p και q είναι γνωστοί, υπολογίζονται οι τετραγωνικές ρίζες modulo p και q, και στην συνέχεια με την εφαρμογή του Κινέζικου Θεωρήματος Υπολοίπου μπορούν να βρεθούν οι ρίζες modulo n. Στην περίπτωση που p=_ q =_ 3 mod 4 οι τετραγωνικές ρίζες

του cmod p και cmod q υπολογίζονται πολύ πιο εύκολα και αποδοτικά σε σχέση με την περίπτωση p=_ q =_ 1mod 4.

Αναλυτικότερα, οι τετραγωνικές ρίζες του cmod p και p=_ 3 mod 4 είναι οι

$$r1= c^{(p+1)/4} mod\ p$$

και

$$r2= -c^{(p+1)/4} mod\ p$$

ενώ αντίστοιχα οι τετραγωνικές ρίζες του cmod q είναι οι

$$r3= c^{(p+1)/4} mod\ q$$

και

$$r4= -c^{(p+1)/4} mod\ q$$

Κάθε συνδυασμός των ριζών modulo p και q θα δώσει και μία ρίζα modulo n. Οι τέσσερις τετραγωνικές ρίζες m_1,m_2,m_3,m_4modulon = pq προκύπτουν από τα ζευγάρια (r_1, r_3), (r_1, r_4),$(r_2,r_3) = (-r_1,-r_4)$ και $(r_2,r_4) = (-r_1, -r_3)$. Χρησιμοποιώντας τον επεκταμένο αλγόριθμο του Ευκλείδη μπορούμε να βρούμε ακεραίους a,b που ικανοποιούν τη σχέση

$$ap+bq=1$$

Τότε, οι τέσσερις τετραγωνικές ρίζες modulo n= pq είναι ίσες με

$$m_1= (apr_3 + bqr_1) mod\ n,$$

$$m_2=(apr_3 - bqr_1) mod\ n,$$

$$m_3= -m_1\ mod\ n$$

και

$$m_4=-m_2\ mod\ n.$$

Ασφάλεια αλγορίθμου

Αν το αρχικό μήνυμα m αντιστοιχεί σε κείμενο, τότε η εύρεση της σωστής ρίζας είναι εύκολη διαδικασία. Στην περίπτωση όμως που το αρχικό μήνυμα m αντιστοιχεί σε κάποιον τυχαίο αριθμό, τότε δεν υπάρχει δυνατότητα να βρεθεί σε ποια από τις τέσσερις ρίζες αντιστοιχεί το m . Το πρόβλημα αυτό

μπορεί να αντιμετωπιστεί με την προσθήκη κάποιων bits στην αρχή του μηνύματος ή με τη χρήση οποιασδήποτε συνάρτησης πλεονασμού.

Η κρυπτογράφηση στο σύστημα Rabin είναι πιο αποδοτική σε σχέση με την κρυπτογράφηση στο RSA, αφού απαιτείται μόνο ο υπολογισμός ενός τετραγώνου modulon. Στο σύστημα RSA το αρχικό μήνυμα υψώνεται στ δύναμη e που είναι τουλάχιστον ίση με 3. Οι αλγόριθμοι αποκρυπτογράφησης έχουν περίπου την ίδια πολυπλοκότητα. Το μειονέκτημα του κρυπτοσυστήματος Rabin είναι το επιπρόσθετο υπολογιστικό κόστος που προστίθεται για την εύρεση της σωστής από τις 4 τετραγωνικές ρίζες. Η απαραίτητη προσθήκη κάποια συνάρτησης πλεονασμού αποτελεί το βασικό λόγο που το σύστημα Rabin δεν χρησιμοποιείται ευρέως σε πρακτικές εφαρμογές.

Το κρυπτοσύστημα Rabin είναι ασφαλές έναντι σε παθητικού τύπου επιθέσεις. Αυτό προκύπτει άμεσα από την ισοδυναμία του προβλήματος εύρεσης τετραγωνικών ριζών modulo ένα σύνθετο αριθμό με το πρόβλημα της παραγοντοποίησης ακεραίων. Επομένως, το κρυπτοσύστημα Rabin είναι ευπαθές σε παρόμοιες επιθέσεις με αυτές που περιγράφηκαν για το σύστημα RSA. Ωστόσο το σύστημα στην αρχική του μορφή, χωρίς την προσθήκη κάποιας συνάρτησης πλεονασμού είναι ευάλωτο σε επιθέσεις επιλεγμένου κρυπτοκειμένου. Η εισαγωγή ορισμένων bits στο αρχικό μήνυμα και γενικότερα η επεξεργασία του από μια συνάρτηση πλεονασμού, λειτουργεί αποτρεπτικά στην επίθεση που αναφέρθηκε προηγουμένως . Αν ο επιτιθέμενος επιλέξει ένα μήνυμα m που έχει τη συγκεκριμένη ιδιότητα, τότε η μηχανή αποκρυπτογράφησης του Β θα επιστρέψει με πολύ μεγάλη πιθανότητα το ίδιο μήνυμα m και επομένως η επίθεση δε θα έχει αποτέλεσμα. Στην περίπτωση που επιλέξει ένα τυχαίο μήνυμα, τότε με πολύ μεγάλη πιθανότητα καμία από τις τέσσερις ρίζες δε θα έχει την ιδιότητα του πλεονασμού και επομένως η μηχανή αποκρυπτογράφησης του Β θα αποτύχει να αποκρυπτογραφήσει το κείμενο και δεν θα επιστρέψει κάποια τιμή.

7Υλοποίηση

7.1Γνωριμία με την βιβλιοθήκη GNUMP

GNU MP είναι μια φορητή βιβλιοθήκη γραμμένη σε C για αυθαίρετη αριθμητική ακρίβεια σε ακέραιους αριθμούς, ορθολογικούς αριθμούς και αριθμούς κινητής υποδιαστολής. Στόχος της είναι να παρέχει την ταχύτερη δυνατή αριθμητική για όλους, εφαρμογές που απαιτούν μεγαλύτερη ακρίβεια από ό, τι υποστηρίζεται άμεσα από τους βασικούς τύπους C. Πολλές εφαρμογές χρησιμοποιούν μόνο μερικές εκατοντάδες bits. Αλλά ορισμένες εφαρμογές μπορεί να χρειαστούν χιλιάδες ή ακόμα και εκατομμύρια. Η GMP έχει σχεδιαστεί για να δίνει καλή απόδοση και για τα δύο, με την επιλογή αλγορίθμων με βάση τα μεγέθη των τελεστών, και κρατώντας την επιβάρυνση στο ελάχιστο. Η ταχύτητα της GMP επιτυγχάνεται χρησιμοποιώντας full words ως το βασικό τύπο αριθμητικής, με τη χρήση εξελιγμένων αλγόριθμων, με τη συμπερίληψη προσεκτικά βελτιστοποιημένου κώδικα assembly για τους πιο κοινούς εσωτερικούς βρόχους για πολλές διαφορετικές CPUs, και δίνει έμφαση στην ταχύτητα (σε αντίθεση με την απλότητα ή κομψότητα).

7.1.1 Εγκατάσταση της GMP

Η GMP έχει ένα autocnf / automake / libtool ανάλογα με την διαμόρφωση του συστήματος. Σε ένα σύστημα Unix η ρύθμιση της βιβλιοθήκης μπορεί να γίνει με το

```
./configure
```

```
Make
```

Μερικά test μπορούμε να τα τρέξουμε ως εξής

```
Makecheck
```

Και μπορείτε να κάνετε εγκατάσταση με την εντολή

```
Make install
```

7.1.2 Δημιουργία επιλογών

Όλες οι συνηθισμένες και προκαθορισμένες επιλογές είναι διαθέσιμες στον « ./configure –help». Το αρχείο "Install .autoconf" περιέχει και αυτό μερικέ πληροφορίες.

Δημιουργία διαδρομής

Για να κάνετε compile σε διαφορετική διαδρομή, κάντε "cd" σε αυτή την διαδρομή, και αλλάξτε τις ρυθμίσεις στην βιβλιοθήκη GMP έτσι ώστε να δείχνουν εκεί. Για παράδειγμα

```
Cd/my/build/dir

/my/sources/gmp-5.0.2/configure
```

Δεν είναι απαραίτητο όλα λειτουργικά συστήματα να υποστηρίζουν κάτι τέτοιο (VPATH). Σε συγκεκριμένα όπως το SunOS και το Slowaris η εντολή make δημιουργεί πρόβλημα και καθιστά αδύνατη την δημιουργία μιας ξεχωριστής διαδρομής.

7.1.3 Βασικά της GMP

Αν χρησιμοποιήσετε functions, macros, datatypes, κ.α που δεν υπάρχουν μέσα στο manual της GMP είναι σίγουρο ότι η εφαρμογή σας δεν θα είναι συμβατή σε επόμενες εκδόσεις της βιβλιοθήκης!

Κεφαλίδες και Βιβλιοθήκες

Όλες οι δηλώσεις που απαιτούνται για τη χρήση της GMP περιλαμβάνονται στο «gmp.h». Είναι σχεδιασμένη για να λειτουργεί τόσο με C όσο και με C + + μεταγλωττιστές.

```
#include<gmp.h>
```

Σημειώστε, ωστόσο, ότι τα πρωτότυπα για GMP λειτουργίες με τις παραμέτρους FILE * παρέχονται μόνο εάν η <stdio.h> περιλαμβάνεται.

```
#include <stdio.h>

#include <gmp.h>
```

7.2 Αλγόριθμοι Πρώτων Παραγόντων

7.2.1 Trial Division (Ανάλυση εντολών)

mpz_t j, v, z, l;

mpz_t seed;

με την εντολή **mpz_t** ορίζουμε τι είδους μεταβλητές θα εισαχθούν στο πρόγραμμα

```
mpz_init(j);
mpz_init(z);
mpz_init(l);
mpz_init(v);
mpz_init(seed);
```

Η εντολή **mpz_init()** δηλώνει μια μεταβλητή και την αρχικοποιεί με τιμή 0.

```
int((unsigned)int());   /* Δημιουργία ενός random αριθμού και ορισμός του
μεγέθους του σε bit  */
sd=rand();
mpz_set_ui(seed, sd);
gmp_randseed(stat, seed);
mpz_urandomb(j, stat, 15);
```

void gmp_randseed (gmp randstate t state, mpz t seed) : Το μέγεθος του seed καθορίζει το πόσο πολλές διαφορετικές ακολουθίες τυχαίων αριθμών είναι δυνατό να δημιουργηθούν.

void mpz_urandomb (mpz t rop, gmp randstate t state, mp bitcnt t n) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n − 1

```
isprime1 = mpz_probab_prime_p(j, 10); /* Εξετάζουμε αν ο j είναι πρώτος
αριθμός */
```

int mpz_probab_prime_p (mpz t n, int reps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

```
mpz_init_set_ui(v,2)
```

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

```
if (isprime1!=0)          /* Εάν ο j είναι πρώτος αριθμός τυπώνει το
αποτέλεσμα */
      {
            printf("THE COMPOSITE NUMBER j IS PRIME:");
            mpz_out_str(stdout, 10, j);
            printf("\n");
      }

      if (isprime1==0)  /*  Εάν δεν είναι πρώτος ο αριθμός τυπώνει ότι είναι
σύνθετος και συνεχίζει να ψάχνει για παράγοντες */
      {
            printf("THE COMPOSITE NUMBER j IS:");
            mpz_out_str(stdout, 10, j);
            printf("\n");

            while (mpz_cmp(v,j)==-1)
            {
                isprime2 = mpz_probab_prime_p(v, 10);  /*Εξετάζουμε μέσα στο
loop αν ο αριθμός v είναι πρώτος */
            mpz_div(z,j,v);
                mpz_mod(l,j,v);
            if(isprime2!=0) //Εάν ο v είναι πρώτος τον τυπώνει
```

```
        {
                if(mpz_cmp_ui(1,0)==0)
                {
                    printf("THE FIRST FACTORS ARE:");
                    mpz_out_str(stdout, 10, v);
                    printf("\n");
                }
            }
mpz_add_ui(v,v,1);  /*Εάν ο v δεν είναι πρώτος επαναλαμβάνουμε την διαδικασία
αυξάνοντας τον v κατά ένα */
        }
    }
```

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

void mpz_div (MP_INT *quotient, MP_INT *dividend, MP_INT *divisor): Με αυτή την εντολή δίνουμε στον χρήστη την δυνατότητα να στον χρήστη να χειριστεί αριθμητικές εξαιρέσεις του προγράμματος σαν να χειριζόταν οποιαδήποτε άλλη αριθμητική εξαίρεση

void mpz_mod (mpz t r, mpz t n, mpz t d) : Θέτει το n mod d στην μεταβλητή r

void mpz_add_ui (MP_INT *sum, MP_INT *addend1, unsigned long int addend2): Με αυτήν την εντολή μπορούμε να δώσουμε το βήμα που θέλουμε να αυξάνετε μια μεταβλητή για παράδειγμα v= v+1

7.2.2 Pollard Rho (Ανάλυση εντολών)

```
    mpz_init (z);          //
    mpz_init (r);          //
    mpz_init (p);          //
    mpz_init (q);          //
```

```
      mpz_init (total1);       // Γίνεται η εισαγωγή των μεταβλητών που θα
χρησιμοποιηθούν

      mpz_init (total2);       //

      mpz_init (total3);       //

      mpz_init (seed);  //

      mpz_init (t);            //
```

Η εντολή mpz_init() δηλώνει μια μεταβλητή και την αρχικοποιεί με τιμή 0.

```
srand ((unsigned) getpid()); // Σε αυτό το σημείο γίνεται η εύρεση του
τυχαίου αριθμού

      sd=rand();               //

mpz_urandomb (r, stat, 15);     //Ορίζουμε το μήκος του τυχαίου αριθμού
```

voidmpz_urandomb (mpztrop, gmprandstatetstate, mpbitcnttn) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n– 1

```
mpz_init_set_si (a, 2);          //Καταχωρούμε την τιμή 2 στο a

mpz_init_set_si (b, 2);          //Καταχωρούμε την τιμή 2 στο b
```

voidmpz_init_set_si (mpztrop, signedlongintop) : Με την συγκεκριμένη εντολή μπορούμε να καταχωρίσουμε μια ακέραια τιμή σε μια μεταβλητή που θέλουμε

```
mpz_init_set_si (z, 10000000); //θέτουμε πόσες επαναλήψεις θα κάνει για βρει
τους πρώτους παράγοντες ( Στην συγκεκριμένο 1 εκατομμύριοloops)

isprime=mpz_probab_prime_p(r, 10);// Εξετάζουμε αν ο αριθμός r είναι πρώτος
```

intmpz_probab_prime_p (mpztn, intreps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

```
if (isprime!=0)     // Ελέγχουμε αν ο r είναι πρώτος και τον τυπώνει
```

```
    {
        printf("THE RANDOM NUMBER IS PRIME:");
        mpz_out_str (stdout, 15, r);
        printf("\n");
    }
```

size_tmpz_out_str (FILE *stream, intbase, mpztop):Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
if (isprime==0)  // Αν ο r δεν είναι πρώτος τότε ψάχνει για παράγοντες
    {
        printf("THE RANDOM COMPOSITE NUMBER IS PRIME:");
        mpz_out_str (stdout, 15, r);
        printf("\n");
sequence1: for (v=1; mpz_cmp_ui(z,v); v++)  // Δηλώνουμε τις περιπτώσεις που
έχει το πρόγραμμα
```

intmpz_cmp_ui (mpztop1, unsignedlongintop2): Η συγκεκριμένη μακρό εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1<op2

```
mpz_pow_ui (a, a, 2);       //
                mpz_add_ui (total1, a, 1);  // Υπολογίζουμε το a^2 + 1 modr
                mpz_mod (a, total1, r);     //

                mpz_pow_ui (b, b, 2);       //
                mpz_add_ui (total2, p, 1);  // Υπολογίζουμε το b^2 +1 modr
                mpz_mod (b, total2, r);     //
                mpz_pow_ui (b, b, 2);       //
                mpz_add_ui (total3, p, 1);  // Υπολογίζουμε το b^2 +1 modr
                mpz_mod (b, total3, r);     //
```

```
mpz_sub (t, a, b);          //

mpz_gcd (p, t, r);          // Υπολογίζουμε το p=gcd(a-b,r)
```

Αναλυτικότερα :

void mpz_ui_pow_ui (mpz t rop, unsigned long int base, unsigned long int exp) : Θέτουμε στο rop το base ^ exp

void mpz_add_ui (mpz t rop, mpz t op1, unsigned long int op2) : Θέτουμε στο rop το op1+ op2

void mpz_mod (mpz t r, mpz t n, mpz t d) : Θέτουμε το n mod d στην μεταβλητή r

void mpz_sub (mpz t rop, mpz t op1, mpz t op2) : Θέτουμε στο rop το op1-op2

void mpz_gcd (mpz t rop, mpz t op1, mpz t op2) : Ορίζουμε στο rop τον μέγιστο κοινό διαιρέτη του op1 και op2

```
if ((mpz_cmp_si(p,1)) && (mpz_cmp (r,p))) //Εξετάζουμε αν το p είναι μεταξύ
του 1 και του r
```

intmpz_cmp_si (mpztop1, unsignedlongintop2): Η συγκεκριμένη μάκρο εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1<op2

Αντίστοιχη με την εντολή mpz_cmp_si είναι και η εντολή mpz_cmp.

```
printf("THIS IS A PRIME FACTOR OF THE COMPOSITE NUMBER:");

                    mpz_out_str (stdout, 10, p);

                    printf("\n");

                    mpz_div (q, r, p);   //Υπολογίζει το πηλίκο του
τυχαίου αριθμού με τον παράγοντα που βρήκε

                    isprime=mpz_probab_prime_p  (q, 10);  //Εξετάζει αν
το q είναι πρώτος
```

size_tmpz_out_str (FILE *stream, intbase, mpztop): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

voidmpz_div (MP_INT *quotient, MP_INT *dividend, MP_INT *divisor): Με αυτή την εντολή δίνουμε στον χρήστη την δυνατότητα να στον χρήστη να χειριστεί αριθμητικές εξαιρέσεις του προγράμματος σαν να χειριζόταν οποιαδήποτε άλλη αριθμητική εξαίρεση

intmpz_probab_prime_p (mpztn, intreps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

```
if (isprime!=0) //Αν είναι πρώτος τυπώνει τον αριθμό και τερματίζει το
πρόγραμμα

                        {

                        printf("THIS IS A PRIME FACTOR OF THE COMPOSITE
NUMBER:");

                              mpz_out_str (stdout, 10, q);

                              printf("\n");

                        goto sequence2;  // Μας στέλνει στην περίπτωση 2 όπου
είναι το return 0 και τερματίζει το πρόγραμμα

                        }

                        else   // Αν δεν είναι πρώτος επαναλαμβάνει την ίδια
διαδικασία θέτοντας στον r την τιμή του q

                        {

                              mpz_set(r,q);

                              goto sequence1; // Μας στέλνει στην περίπτωση 1
όπου επαναλαμβάνεται η διαδικασία

                        }

                        goto sequence2;

                  }


                  if (mpz_cmp_si(p,1))  //Αν ο p είναι μεγαλύτερος του 1
επαναλαμβάνεται η διαδικασία

                  {

                        goto sequence1;

                  }
```

```
                if (p==r) //Αν ο p είναι ίσος με τον r τότε αποτυγχάνει μας
εμφανίζει ότι απέτυχε και τερματίζει το πρόγραμμα
                {
                        printf("FAIL");
                        goto sequence2;
                }
        }
}
sequence2:
return 0;
}
```

7.2.3 PollardRhop-1 (Ανάλυση εντολών)

```
int main(void)
{
      mpf_t w1; // Ο αλγόριθμος του ln2 μου δόθηκε

      #define computeln2(w1)
      {

            long v;

            mpz_t z4;
            mpf_t s5, s1, t2, t1, half1;
            mpf_set_default_prec(10);
            mpz_init(z4);
```

```
        mpf_init(s5);
        mpf_init(s1);
        mpf_init(t2);
        mpf_init(t1);
        mpf_init(half1);
        mpz_init_set_si (z4, 5);
        mpf_set_ui(s5, 0);
        mpf_set_str(t2, "0.5e0", 10);
        mpf_set_str(t1, "0.5e0", 10);
        mpf_set_str(half1, "0.5e0", 10);

        for(v = 2; mpz_cmp_ui(z4, v); v++)
        {
              mpf_add(s1, s5, t2);
              if(mpf_cmp(s5, s1) == 0)
                    break;
              mpf_set(s5, s1);
              mpf_mul(t1, t1, half1);
              mpf_div_ui(t2, t1, v);
        }

        mpf_set(w1, s5);

  }
```

Αναλυτικότερα :

mpz_t : Δηλώνει τι μεταβλητές θα χρησιμοποιηθούν

mpf_t : Δηλώνει τι μεταβλητές θα χρησιμοποιηθούν τύπου **float**

mpz_init() : δηλώνει μια μεταβλητή και την αρχικοποιεί με τιμή 0.

int mpf_set_str (mpf t rop, char *str, int base): Ορίζει την αξία του rop από το αλφαριθμητικό. Το αλφαριθμητικό είναι της μορφής "M@N"" ή , αν η τιμή του base είναι λιγότερο από 10, εναλλακτικά "MeN" το M είναι το δεκαδικό και το N ο εκθέτης

int mpz_cmp_ui (mpz t op1, unsigned long int op2): Η συγκεκριμένη μακρό εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1< op2

void mpf_add (mpf t rop, mpf t op1, mpf t op2): Ορίζει την τιμή op1+ op2 στην μεταβλητή rop

void mpf_set (mpf t rop, mpf t op): Ορίζει καινούρια τιμή σε μια αρχικοποιημένη float μεταβλητή

void mpf_mul (mpf t rop, mpf t op1, mpf t op2): Ορίζουμε στο rop την τιμή op1 X op2

void mpf_div_ui (mpf t rop, mpf t op1, unsigned long int op2) : Ορίζουμε στο rop την τιμή op1/op2

```
mpf_tw, x;  // Ο αλγόριθμος του lnx μου δόθηκε
      #define myln(w,x);
      {
            long h;
            mpz_t z3;                          /* Δήλωσημεταβλητών
            mpf_t y, n1, help, help1, t3;
            mpf_t s3, s4, t4;
            mpf_set_default_prec(10);
            mpf_init(x);
            mpz_init(z3);
            mpf_init(y);
            mpf_init(n1);
            mpf_init(help);
            mpf_init(help1);
            mpf_init(t3);
```

```
mpf_init(s3);
mpf_init(s4);
mpf_init(t4);
mpz_init_set_si (z3, 5);
mpf_set_str(help, "0.5e0", 10);
mpf_set_str(help1, "0.15e1", 10);   */
if (mpf_cmp_ui(x, 1) == -1)
{
      mpf_set_ui(n1, 0);
      mpf_ui_sub(y, 1, x);
}
If (mpf_cmp_ui(x, 1) == 0)
{
      mpf_set_ui(w, 0);
}

if(mpf_cmp_ui(x, 1) == 1)
{
      mpf_set_ui(n1, 0);
      mpf_set_str(t3, "0.5e0", 10);
      mpf_mul(t3, t3, x);
while (mpf_cmp_ui(t3, 1) != -1)
      {
            mpf_div_ui(t3, t3, 2);
            mpf_add_ui(n1, n1, 1);
      }
      mpf_mul_ui(t3, t3, 2);
      mpf_ui_sub(y, 1, t3);
```

```
        }
If (mpf_cmp(x, help) == -1)
        {
                mpf_set_ui(n1, 0);
                mpf_set_str(t3, "0.2e1", 10);
                mpf_mul(t3, t3, x);
                while (mpf_cmp_ui(t3, 1) == -1)
                {
                        mpf_mul_ui(t3, t3, 2);
                        mpf_sub_ui(n1, n1, 1);
                }
                mpf_div_ui(t3, t3, 2);
                mpf_ui_sub(y, 1, t3);
        }
        mpf_set_ui(s3, 0);
        mpf_set(t3, y);
        mpf_set(t4, y);

        for (h = 2; mpz_cmp_ui(z3, h); h++)
        {
                mpf_add(s4, s3, t3);
                if(mpf_cmp(s3, s4) == 0)
                        break;
                mpf_set(s3, s4);
                mpf_mul(t4, t4, y);
                mpf_div_ui(t3, t4, h);
        }
 if(mpf_sgn(n1) == 0)
```

```
        {
                mpf_set_ui(t3, 0);
        }
        else
        {
                computeln2(&t3);
                mpf_mul(t3, t3, n1);
        }
        mpf_sub(w, t3, s3);

}
```

Αναλυτικότερα:

int mpf_cmp_ui (mpz t op1, unsigned long int op2): Η συγκεκριμένη μακρό εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1< op2

void mpf_set_ui (mpf t rop, mpf t op): Ορίζει καινούρια τιμή σε μια αρχικοποιημένη float μεταβλητή

void mpf_ui_sub (mpf t rop, unsigned long int op1, mpf t op2):Θέτουμε στο rop το op1-op2

int mpf_set_str (mpf t rop, char *str, int base):Ορίζει την αξία του rop από το αλφαριθμητικό. Το αλφαριθμητικό είναι της μορφής "M@N"" ή , αν η τιμή του base είναι λιγότερο από 10, εναλλακτικά "MeN" το M είναι το δεκαδικό και το N ο εκθέτης

void mpf_mul (mpf t rop, mpf t op1, mpf t op2): Ορίζουμε στο rop την τιμή op1 X op2

void mpf_div_ui (mpf t rop, mpf t op1, unsigned long int op2) : Ορίζουμε στο rop την τιμή op1/op2

void mpf_add_ui (mpf t rop, mpf t op1, unsigned long int op2) : Ορίζουμε στο rop την τιμή op1 + op2

void mpf_mul_ui (mpf t rop, mpf t op1, unsigned long int op2) : Έχει την ίδια λειτουργία με την mpf_mul

void mpf_sub_ui (mpf t rop, unsigned long int op1, mpf t op2) : Έχει την ίδια λειτουργία με την mpf_ui_sub

void mpf_set (mpf t rop, mpf t op): Ορίζει καινούρια τιμή σε μια αρχικοποιημένη float μεταβλητή

void mpf_add (mpf t rop, mpf t op1, mpf t op2): Έχει την ίδια λειτουργία με την mpf_add_ui

==========Στο κύριο πρόγραμμα==========

```
mpz_t B, c, d, i, j, jv, m, r, u;   // Ορίζουμε τι είδους μεταβλητές θα
εισαχθούν
      mpf_t l1, l2, z1, z2;              //
gmp_randinit(stat,GMP_RAND_ALG_LC,120);
gmp_randinit:  Επιστρέφει μια ένδειξη σφάλματος μέσω μιας κεντρικής
μεταβλητής
mpf_init (l1);          //
mpf_init (l2);    //Εισάγονται οι float
mpf_init (z1);    //μεταβλητές
mpf_init (z2);    //

mpz_init (B);           //
mpz_init (c);           //
mpz_init (d);           //
mpz_init (i);           //
mpz_init (j);           //Εισάγονται οι integer μεταβλητές
mpz_init (jv);          //
mpz_init (m);           //
mpz_init (r);           //
mpz_init (u);           //
mpz_init (seed);    //

srand((unsigned)getpid());     //
sd=rand();                       //
```

```
mpz_set_ui(seed,sd);          // Δημιουργείται ο τυχαίος αριθμός ορισμένου
μήκους bits

gmp_randseed(stat,seed);      //

mpz_urandomb(j,stat,15);       //
```

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void gmp_randseed (gmp randstate t state, mpz t seed) : Το μέγεθος του seed καθορίζει το πόσο πολλές διαφορετικές ακολουθίες τυχαίων αριθμών είναι δυνατό να δημιουργηθούν.

void mpz_urandomb (mpz t rop, gmp randstate t state, mp bitcnt t n) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n − 1

isprime1=mpz_probab_prime_p(j,10); //Εξετάζει αν ο τυχαίος αριθμός j είναι πρώτος

int mpz_probab_prime_p (mpz t n, int reps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

```
mpz_init_set_ui(B,1000);      //

mpz_init_set_ui(u,2);         // Γίνεται αρχικοποίηση συγκεκριμένων τιμών
στις μεταβλητές

mpz_init_set_ui(i,0);         //
```

void mpz_init_set_ui(mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή του rop.

```
mpz_sub_ui(jv,j,1);           //Υπολογίζει το jv=j-1
```

void mpz_sub (mpz t rop, mpz t op1, mpz t op2) : Θέτουμε στο rop το op1-op2

```
mpz_urandomb(c,stat,15);      //Ορίζεται το μήκος του c

if (isprime1==0)  // Εάν ο j δεν είναι πρώτος τυπώνει ότι το j είναι σύνθετος

      {
```

```
            printf("THE COMPOSITE j IS:");

            mpz_out_str(stdout,10,j);

            printf("\n");
```

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
if ((mpz_cmp_ui(c,2)==1) || (mpz_cmp_ui(c,2)==0)) //Εάν το c είναι μεγαλύτερο
ή ίσο με 2

            {

                  if ((mpz_cmp(c,jv)==-1) || (mpz_cmp(c,jv)==0)) // και εάν
το c είναι μικρότερο ή ίσο του jv κάνει τα εξής

                  {
```

int mpz_cmp_ui (mpz t op1, unsigned long int op2): Η συγκεκριμένη μακρό εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1< op2

int mpz_cmp(mpz t op1, unsigned long int op2): Έχει ίδια λειτουργία με την εντολή mpz_cmp_ui

```
mpz_gcd(d,c,j); //Υπολογίζει το ΜΚΔ των c και j

isprime3=mpz_probab_prime_p(d,10); //Εξετάζει αν το d είναι πρώτος

if ((mpz_cmp_ui(d,2)==1) || (mpz_cmp_ui(d,2)==0)) // Ελέγχουμε αν το d είναι
μεγαλύτερο ή ίσο με το 2

{

if (isprime3!=0)  //Αν το d είναι μεγαλύτερο ή ίσο με το 2 εξετάζουμε αν
είναι πρώτος και το τυπώνουμε

{

printf("THE d IS:");

mpz_out_str(stdout,10,d);

printf("\n");

}
```

```
else //αλλιώς τερματίζουμε το πρόγραμμα

{

goto sequence2;

}

}
```

void mpz_gcd (mpz t rop, mpz t op1, mpz t op2) : Ορίζουμε στο rop τον μέγιστο κοινό διαιρέτη του op1 και op2

```
else //Εάν το d είναι μικρότερο του 2 κάνει τα εξής

{

while ((mpz_cmp(u,B)==-1)||(mpz_cmp(u,B)==0)) //Όσο το u είναι μικρότερο ή
ίσο του B κάνει τα εξής

{

isprime2=mpz_probab_prime_p(u,10); // Εξετάζει αν το u είναι πρώτος

if (isprime2!=0)// Εάν το u είναι πρώτος υπολογίζουμε

{

myln(z1, n); // το ln(n)

myln(z2, u); // το ln(u)

mpf_div (l2, z1, z2); //και κάνουμε την διαίρεση των 2 παραπάνω

 mpf_trunc(l1,l2); //Υπολογίζεται το infimum

mpz_set_f (r, l1); //Μετατρέπει σε integer από float

 while ((mpz_cmp(i,r)==-1)||(mpz_cmp(i,r)==0)) // Υπολογίζει αν το ι είναι
μικρότερο από το r ή ίσο

{

mpz_mul(u,u,u); //Υπολογίζει το u=u*u

mpz_add_ui(i,i,1); //Υπολογίζει i=i+1

}

 mpz_powm(c,c,u,j); //Υπολογίζει το c=c^u modj

}
```

```
 mpz_add_ui(u,u,1); //Υπολογίζει u=u+1
}
```

void mpf_trunc (mpf t rop, mpf t op): Ορίζεται η στρογγυλοποίηση ενός ακέραιου ως τιμή του rop

void mpz_set_f (mpzt rop, mpft op): Ορίζει την τιμή του rop από την op

void mpz_mul (mpzt rop, mpzt op1, mpzt op2):Ορίζει στο rop την τιμήop1 Xop2

void mpz_add_ui (MP_INT *sum, MP_INT *addend1, unsigned long int addend2): Με αυτήν την εντολή μπορούμε να δώσουμε το βήμα που θέλουμε να αυξάνετε μια μεταβλητή για παράδειγμα v= v+1

void mpz_powm (mpz t rop, mpz t base, mpz t exp, mpz t mod): Ορίζει στην rop την τιμή base^exp mod mod.

```
mpz_sub_ui(m,c,1); //Υπολογίζει το m=c-1
 mpz_gcd(d,m,j); //Υπολογίζει το MKD των m και j
 isprime3=mpz_probab_prime_p(d,10); //εξετάζει αν το d είναι πρώτος
if (isprime3!=0) //εάν είναι πρώτος
{
if ((mpz_cmp_ui(d,1)==0) || (mpz_cmp(d,j)==0)) // και αν το d είναι ίσο με 1
ή το d είναι ίσο με  j
{
      printf("FAIL\n");  // Μας τυπώνει ότι απέτυχε και τερματίζει το
πρόγραμμα
      goto sequence2;
}
      else
{ //Αλλιώς τυπώνει το d
      printf("THE d IS:");
            mpz_out_str(stdout,10,d);
printf("\n");
```

```
}
 }
else
{ //Εάν δεν είναι πρώτος τερματίζει το πρόγραμμα
goto sequence2;
}
```

void mpz_sub_ui (mpz t rop, mpz t op1, mpz t op2) : Θέτουμε στο rop το op1-op2

7.2.4 Συγκριτική μελέτη αλγορίθμων παραγωγής πρώτων παραγόντων

7.2.4.1 TrialDivision

Για τον αλγόριθμο Trial Division ύστερα από μελέτη που έγινε καταλήξαμε στα εξής αποτελέσματα :

Σημειώστε επίσης ότι ο χρόνος που χρειάζεται η Trial Division για να βρει τους παράγοντες εξαρτάται από το μέγεθος του μικρότερου συντελεστή (αποτέλεσμα), καθώς και τον αριθμό που πρέπει να υπολογίσει.

Input	Result	sqrt	squaring	odd	primes
2	PRIME	20 ns	3 ns	20 ns	22 ns
23	PRIME	57 ns	41 ns	32 ns	51 ns
281	PRIME	220 ns	208 ns	107 ns	99 ns
2741	PRIME	648 ns	661 ns	354 ns	233 ns
27737	PRIME	2.0 µs	2.1 µs	1.1 µs	0.52 µs
241333	PRIME	5.9 µs	6.2 µs	3.1 µs	1.2 µs
2327911	PRIME	18.2 µs	19.1 µs	9.6 µs	3.1 µs
24988597	PRIME	59 µs	63 µs	31 µs	8.5 µs
290045981	PRIME	200 µs	215 µs	106 µs	25 µs
2144892011	PRIME	550 µs	583 µs	289 µs	61 µs
2144892013	11	144 ns	150 ns	83 ns	87 ns
2144892019	29287	350 µs	368 µs	183 µs	41 µs

Όπως ήταν αναμενόμενο το function που χρησιμοποιεί μόνοprime divisors είναι αρκετά γρήγορο για μεγάλες εισόδους, παρά την αρχικοποίηση τους. Άλλες μέθοδοι όπως του τετραγωνισμού και των πιθανοτήτων διακρίθηκαν μόνο για μικρές εισόδους / εξόδους. Γενικότερα η Trial Divisioneίναι πολύ γρήγορη για αριθμούς με μικρούς παράγοντες άσχετα με το μέγεθος του αριθμού

7.2.4.2Pollard Rho

Ο αλγόριθμος του Pollard Rhoείναι πολύ γρήγορος για αριθμούς με μικρούς παράγοντες. Για παράδειγμα, σε ένα σταθμό εργασίας 3 GHz, ο αρχικός αλγόριθμος rho βρήκε τον παράγοντα 274177 του έκτου αριθμού Fermat (18446744073709551617) σε 26 milliseconds.

Για f, επιλέγουμε ένα πολυώνυμο με ακέραιους συντελεστές. Οι πιο κοινές από αυτές είναι της μορφής:

$$f(x) = (x^2 + c) \bmod n,\ c \neq 0, -2.$$

7.2.4.3Pollard Rhop-1

Ο χρόνος εκτέλεσης του αλγορίθμου είναι $O(B \times \log B \times \log^2 n)$, για μεγαλύτερες τιμές του B ο αλγόριθμος τρέχει πιο αργά, αλλά είναι πιο πιθανό να παράγει ένα παράγοντα.

Ας δούμε μια σύγκριση των αλγορίθμων του Pollard Rhoκαι Pollard Rho p-1.

Table 1: Products of two nearby primes				
	number 1	number 2	number 3	number 4
Pollard's P-1	0.12	0.72	1.9	0.39
Pollard's Rho	0.02	0.33	0.59	0.61

Όπου:
number 1 = 3980021 = 1993 x 1997
number 2 = 16831170221 = 129733 x 129737
number 3 = 431589872009 = 656951 x 656959
number 4 = 1469322167111 = 1212121 x 1212191

Table 2: Products of three nearby primes				
	number 1	number 2	number 3	number 4
Pollard's P-1	0.26	0.68	0.28	1.74
Pollard's Rho	0.05	0.05	0.07	0.29

Όπου:
number 1 = 7956061979 = 1993 x 1997 x 1999
number 2 = 110154695923 = 4789 x 4793 x 4799
number 3 = 1019829472003 = 10061 x 10067 x 10069

number 4 = 1005660644975291 = 100183 x 100189 x 100193

Table 3: Products of three arbitrary primes				
	number 1	number 2	number 3	number 4
Pollard's P-1	0.23	0.90	5.46	-
Pollard's Rho	0.07	0.06	0.14	1.31

Όπου:
number 1 = 14960418503 = 179 x 8467 x 9871
number 2 = 8355211084777 = 1163 x 12347 x 581857
number 3 = 416531649825896503 = 12983 x 987533 x 32487877
number 4 = 153674304751986405509 = 762479 x 1276237 x 157921783

Σημείωση: Οι χρόνοι είναι σε second

7.3 Αλγόριθμοι Κρυπτογραφίας

7.3.1 RSA (Ανάλυση εντολών)

```
mpz_t p, q, p1, q1, n, p_1, q_1, f, e, gcd, d, c, m, A, B, C, D, i, ed, ed_m,
phl1, phl2, dia1, dia2, message_int, a, b, temp_p, temp_q; // Δήλωση των
μεταβλητών

mpz_t seed; // που θα χρησιμοποιηθούν από το πρόγραμμα

mpz_t : Δηλώνει τι μεταβλητές θα χρησιμοποιηθούν

gmp_randinit(stat, GMP_RAND_ALG_LC, 120);

gmp_randinit:  Επιστρέφει μια ένδειξη σφάλματος μέσω μιας κεντρικής
μεταβλητής

mpz_init(p);    /*    Δήλωση ακεραίων μεταβλητών

mpz_init(q);

mpz_init(p1);

mpz_init(q1);

mpz_init(n);

mpz_init(p_1);

mpz_init(q_1);

mpz_init(f);
```

```
mpz_init(e);
mpz_init(gcd);
mpz_init(d);
mpz_init(c);
mpz_init(A);
mpz_init(m);
mpz_init(B);
mpz_init(C);
mpz_init(D);
mpz_init(i);
mpz_init(ed);
mpz_init(ed_m);
mpz_init(phl1);
mpz_init(phl2);
mpz_init(dia1);
mpz_init(dia2);
mpz_init(message_int);
mpz_init(seed);
unsigned char message[MAXLEN], out_message[MAXLEN];
mpz_init(a);
mpz_init(b);  */
```

mpz_init : Δηλώνει ότι μια μεταβλητή θα έχει ακέραια τιμή

```
srand( (unsigned) getpid());
sd=rand();
mpz_set_ui(seed, sd);
```

```
gmp_randseed(stat, seed);

srand( (unsigned) getpid());                                    //

mpz_set_ui(seed, sd);                                           //

mpz_urandomb(p1, stat, 512);                          // Δημιουργία τυχαίου
αριθμού p
```

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void gmp_randseed (gmp randstate t state, mpz t seed) : Το μέγεθος του seed καθορίζει το πόσο πολλές διαφορετικές ακολουθίες τυχαίων αριθμών είναι δυνατό να δημιουργηθούν.

void mpz_urandomb (mpz t rop, gmp randstate t state, mp bitcnt t n) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n – 1

```
srand( (unsigned) getpid());

sd=rand();

mpz_set_ui(seed, sd);

gmp_randseed(stat, seed);

srand( (unsigned) getpid());                                    //

mpz_set_ui(seed, sd);                                           // Δημιουργία
τυχαίου αριθμού q

mpz_urandomb(q1, stat, 512);                          //

int primetest;

primetest = mpz_probab_prime_p(p1, 5);

if (primetest != 0)

{

      mpz_set(p, p1);

      printf("p= "); mpz_out_str(stdout, 10, p); //Εκτύπωση p

      printf("\n");

      printf("\n");

}
```

```
else
{
	mpz_nextprime(p, p1);
	printf("p= "); mpz_out_str(stdout, 10, p); //Εκτύπωση p
	printf("\n");
	printf("\n");
}
```

int mpz_probab_prime_p (mpz t n, int reps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

void mpz_nextprime (mpz t rop, mpz t op): Θέτει στην rop την τιμή του επόμενου παράγοντα μεγαλύτερου από το op

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
primetest = mpz_probab_prime_p(q1, 5);
if (primetest != 0)
{
	mpz_set(q, q1);
	printf("q= "); mpz_out_str(stdout, 10, q); //Εκτύπωση q
	printf("\n");
	printf("\n");
}
else
{
	mpz_nextprime(q, p);
	printf("q= "); mpz_out_str(stdout, 10, q); //Εκτύπωση q
	printf("\n");
```

```
	printf("\n");
}
```

int mpz_probab_prime_p (mpz t n, int reps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

void mpz_nextprime (mpz t rop, mpz t op): Θέτει στην rop την τιμή του επόμενου παράγοντα μεγαλύτερου από το op

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
//------(Υπολογισμός των n, p-1, q-1, f)-----------------------------------
-----------------------------------------------

	mpz_mul(n, p, q);
	printf("n= "); mpz_out_str(stdout, 10, n);		//Εκτύπωση n=p*q
	printf("\n");
	printf("\n");
	mpz_sub_ui(p_1, p, 1);
	printf("p-1= "); mpz_out_str(stdout, 10, p_1);		//Εκτύπωση p-1
	printf("\n");
	printf("\n");
	mpz_sub_ui(q_1, q, 1);
	printf("q-1= "); mpz_out_str(stdout, 10, q_1);		//Εκτύπωση q-1
	printf("\n");
	printf("\n");
	mpz_mul(f, p_1, q_1);
	printf("f= "); mpz_out_str(stdout, 10, f);		//Εκτύπωση f=(p-
1)*(q-1)
	printf("\n");
	printf("\n");
```

void mpz_mul (mpz t rop, mpz t op1, mpz t op2): Ορίζει στο rop την τιμή op1 X op2

void mpz_sub_ui (mpz t rop, mpz t op1, mpz t op2) :Θέτουμε στο rop το op1-op2

void mpz_mul (mpz t rop, mpz t op1, mpz t op2):Ορίζει στο rop την τιμή op1 X op2

```
//------(Επιλέγει τυχαίο e τέτοιο ώστε 1<e<f)-------------------------------
-----------------------------------------------
      sequence1:
      mpz_urandomb(e, stat, 512);
               if ((mpz_cmp_si(e,1)) && (mpz_cmp (e,f)))
      {
      mpz_gcd(gcd, e, f);                  // Υπολογίζει το MKD των e και f
              if (mpz_cmp_si(gcd,1))            //Εξετάζει αν το MKD είναι 1
και αν ναι τον εκτυπώνει αλλιώς δημιουργεί νέο e
            {
                  printf("To e einai:\n");
                  printf("e= ");
                  mpz_out_str(stdout, 10, e);
                  printf("\n");
                  printf("\n");
            }
            else
            {
                  goto sequence1;
            }
      }
      else
      {
```

```
        goto sequence1;
    }
```

void mpz_urandomb (mpz t rop, gmp randstate t state, mp bitcnt t n) :Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n – 1

void mpz_gcd (mpz t rop, mpz t op1, mpz t op2) :Ορίζουμε στο rop τον μέγιστο κοινό διαιρέτη του op1 και op2

int mpz_cmp_si (mpz t op1, unsigned long int op2): Η συγκεκριμένη μακρό εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1< op2

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
//------(Ypologismos tou d tetoio wste 1<d<f kai e*d= 1 mod f)---------------
---------------------------------------- mpz_powm_ui(d, e, -1, f);
```

void mpz_powm_ui (mpz t rop, mpz t base, mpz t exp, mpz t mod):Ορίζει στην rop την τιμή base^exp mod mod.

```
printf("To d einai:\n");   // Εκτύπωση του d
printf("d= ");
mpz_out_str(stdout, 10, d);
printf("\n");
printf("\n");
printf("To Dhmosio kleidi einai:\n");          //Εκτύπωση του δημόσιου
κλειδιού
printf("(n,e)= ");
printf(" (");
mpz_out_str(stdout, 10, n);
printf("  ,  ");
```

```
mpz_out_str(stdout, 10, e);

printf(") ");

printf("\n");

printf("\n");


printf("To Idiwtiko Kleidi einai:\n");          //Εκτύπωση του ιδιωτικού
κλειδιού

printf("d= ");

mpz_out_str(stdout, 10, d);

printf("\n");

printf("\n");

printf("Input a string(Max length = %d): ", MAXLEN);  //Ζητάμε από τον χρήστη
να εισάγει το κείμενο προς κρυπτογράφηση

fgets(message, MAXLEN - 1, stdin);

//-----(Κλήση συνάρτησης μετατροπής κειμένου σε ακέραιο)--------------------
-------------------------------------

str2int(m, message);

#ifdef HEX

gmp_printf("MHNYMA(%d)  -> %Zx\n\n", strlen(message), m);   //Εκτύπωση της
μετατροπής του μηνύματος σε αριθμό

#else

gmp_printf("MHNYMA(%d)  -> %Zd\n\n", strlen(message), m);

#endif

//-----------(Κρυπτογράφηση: Υπολογισμός του c = m^e mod n)-----------------
-------------------------------------

mpz_powm(c, m, e, n);

exit1:

printf("To ciphertext einai:\n");          //Εκτύπωση του ciphertext

printf("c= ");

mpz_out_str(stdout, 10, c);
```

```
printf("\n");

printf("\n");
```

void mpz_powm (mpz t rop, mpz t base, mpz t exp, mpz t mod):Ορίζει στην rop την τιμή base^exp mod mod.

size_t mpz_out_str (FILE *stream, int base, mpz t op): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
mpz_set_ui(m,0);        //Εκκαθάριση των μεταβλητών
```

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

```
//------(Αποκρυπτογράφηση: Υπολογισμός του m = c^d mod n)--------------------
-----------------------------------

mpz_powm(m, c, d, n);

exit2:

printf("To plaintext einai:\n");          //Εκτύπωσητου ciphertext

printf("m= ");

mpz_out_str(stdout, 10, m);

printf("\n");

printf("\n");
```

void mpz_powm (mpz t rop, mpz t base, mpz t exp, mpz t mod):Ορίζει στην rop την τιμή base^exp mod mod.

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

```
//-----(Κλήση συνάρτησης μετατροπής ακεραίου σε κείμενο)--------------------
-----------------------------------

int2str(out_message, m);
```

```
#ifdef HEX

gmp_printf("Αποκρυπτογραφημένο μήνυμα: %Zd\n                -> %s(%d)\n", m,
out_message, strlen(out_message));

#else

gmp_printf("Αποκρυπτογραφημένο μήνυμα: %Zd\n                -> %s(%d)\n", m,
out_message, strlen(out_message));

#endif

//------(Συνάρτηση μετατροπής string σε integer)-----------------------------
-------------------------------------------------------

void str2int(mpz_t ret, char *str)

{

long int str_len, j;

unsigned char C;


str_len = strlen(str);

if(str[str_len - 1] == '\n')

str[str_len - 1] = '\0';

str_len = strlen(str);


mpz_set_ui(ret, 0UL);                //Αρχικοποίηση του ret σε 0


for(j = str_len - 1; j >= 0; j--)           // ret = str[str_len - 1] *
BASE^(str_len - 1) + ... + str[1] * BASE + str[0]

{

C = str[j];


mpz_mul_ui(ret, ret, (unsigned long)BASE);            //Me8odos Horner

mpz_add_ui(ret, ret, (unsigned long)C);

}
```

```
}
```

void mpz_mul_ui (mpz t rop, mpz t op1, mpz t op2):Ορίζει στο rop την τιμή op1 X op2

void mpz_set_ui (mpzt rop, unsignedlongintop) : Με αυτήν την εντολή μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void mpz_add_ui (MP_INT *sum, MP_INT *addend1, unsignedlongintaddend2): Με αυτήν την εντολή μπορούμε να δώσουμε το βήμα που θέλουμε να αυξάνετε μια μεταβλητή για παράδειγμα v= v+1

```
//------(Συνάρτηση μετατροπής integer σε string)-----------------------------
---------------------------------------- void int2str(char *str, mpz_t
org_str_int)
{
long int str_len, i;            //
mpz_t max_int, c_int, str_int; // Δήλωση μεταβλητών που θα χρησιμοποιηθούν
mpz_init(max_int);                  //Αρχικοποίηση
mpz_init(c_int);
mpz_init(str_int);
mpz_set(str_int, org_str_int);
mpz_set_ui(max_int, 1UL);           //Παίρνει το μήκος της εγγραφής
for(i = 0; i < MAXLEN; i++)
{
if(mpz_cmp(str_int, max_int) <= 0)
{
      str_len = i;
      break;
}
mpz_mul_ui(max_int, max_int, (unsigned long)BASE);
}
for(i = 0; i < str_len; i++)        //Μετατροπή σε γράμμα
{
```

```
mpz_mod_ui(c_int, str_int, (unsigned long)BASE);

mpz_sub(str_int, str_int, c_int);

mpz_tdiv_q_ui(str_int, str_int, (unsigned long)BASE);


str[i] = mpz_get_ui(c_int);

}

str[str_len] = '\0';


mpz_clear(max_int);

mpz_clear(c_int);

mpz_clear(str_int);

}
```

void mpz_set_ui (mpz t rop, unsigned long int op) : Με αυτήν την εντολή μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

int mpz_cmp_ui (mpz t op1, unsigned long int op2): Η συγκεκριμένη μάκρο εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1< op2

void mpf_mul_ui (mpf t rop, mpf t op1, mpf t op2): Ορίζουμε στο rop την τιμή op1 X op2

unsigned long int mpz_mod_ui (mpz t r, mpz t n, unsigned long int d):Θέτει το n mod d στην μεταβλητή r

void mpz_sub (mpz t rop, mpz t op1, mpz t op2) :Θέτουμε στο rop το op1-op2

void mpz_clear (mpz t x): Αδειάζει τις μεταβλητές έτσι ώστε να μην τρώει μνήμη από τον υπολογιστή όταν δεν τις χρειαζόμαστε.

7.3.2Rabin(Ανάλυση εντολών)

```
mpz_ttemp_p,temp_q, message_int;   /* Δήλωση μεταβλητών που θα
χρησιμοποιηθούν από το πρόγραμμα και δήλωση του τύπου

int root_no;                              της μεταβλητής

mpz_t c;

mpz_init(c);

mpz_t a;

mpz_init(a);

mpz_t b;

mpz_init(b);

mpz_t m;

mpz_init(m);

mpz_t(p_ek);

mpz_init(p_ek);

mpz_t(q_ek);

mpz_init(q_ek);

mpz_t (r);

mpz_init(r);

mpz_t (s);

mpz_init(s);

mpz_t root1;

mpz_init(root1);

mpz_t root2;

mpz_init(root2);

mpz_t root3;

mpz_init(root3);

mpz_t root4;

mpz_init(root4);
```

```
mpz_t aps;
mpz_init(aps);
mpz_t bqr;
mpz_init(bqr);              */

gmp_randstate_t rand_state;
gmp_randinit_default(rand_state);
```

voidgmp_randinit_default (gmprandstatetstate): αρχικοποιεί τον επιλεγμένο αλγόριθμο συμβιβάζοντας τον μεταξύ ταχύτητας και τυχαιότητας.

```
gmp_randinit(stat, GMP_RAND_ALG_LC, 120);
```

gmp_randinit: Επιστρέφει μια ένδειξη σφάλματος μέσω μιας κεντρικής μεταβλητής

```
//------(Δημιουργία p)-----------------------------------------------------
-------------------------
srand( (unsigned) getpid());
sd=rand();
mpz_set_ui(seed, sd);
gmp_randseed(stat, seed);

srand( (unsigned) getpid());                                    //
mpz_set_ui(seed, sd);                                           //
sequence1:        //Δημιουργία τυχαίου αριθμού p
mpz_urandomb(p1, stat, 200);                                    //
//
```

void mpz_set_ui (mpzt rop, unsigned long int op) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void gmp_randseed (gmprandstatetstate, mpzt seed) : Το μέγεθος του seed καθορίζει το πόσο πολλές διαφορετικές ακολουθίες τυχαίων αριθμών είναι δυνατό να δημιουργηθούν.

void mpz_set_ui (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void mpz_urandomb (mpzt rop, gmprandstatetstate, mpbitcnttn) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n - 1

```
//------(Εξετάζει αν είναι πρώτος ο p και αν είναι 3 mod 4, αλλιώς βρίσκουμε
τον επόμενο πρώτο)---------

int primetest;

primetest = mpz_probab_prime_p(p1, 5);

if (primetest != 0)

{

mpz_set(p, p1);

mpz_sub_ui(pmod,p,3);

mpz_mod_ui(pmod, pmod, 4);

if (mpz_cmp_ui(pmod,0)!=0)

{

      goto sequence1;

}

else

{

      printf("p= "); mpz_out_str(stdout, 10, p); //Ektupwsh p

      printf("\n");

      printf("\n");

}

}

else
```

```
{
mpz_nextprime(p, p1);
mpz_sub_ui(pmod,p,3);
mpz_mod_ui(pmod, pmod, 4);
if (mpz_cmp_ui(pmod,0)!=0)
{
      goto sequence1;
}
else
{
      printf("p= "); mpz_out_str(stdout, 10, p); //Ektupwsh p
      printf("\n");
      printf("\n");
}
}
```

intmpz_probab_prime_p (mpztn, intreps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

void mpz_set (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

unsigned long int mpz_mod_ui (mpz t r, mpz t n, unsigned long int d): Θέτει το n mod d στην μεταβλητή r

intmpz_cmp_ui (mpztop1, unsignedlongintop2): Η συγκεκριμένη μάκρο εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1<op2

size_tmpz_out_str (FILE *stream, intbase, mpztop): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
//------(Δημιουργία q)-------------------------------------------------------
------------------------

srand( (unsigned) getpid());

sd=rand();

mpz_set_ui(seed, sd);

gmp_randseed(stat, seed);


srand( (unsigned) getpid());                                    //

mpz_set_ui(seed, sd);                                           //

sequence2:                                                      //Δημιουργία τυχαίου
αριθμού q

mpz_urandomb(q1, stat, 200);
```

void mpz_set_ui (mpzt *rop*, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void gmp_randseed (gmprandstatetstate, mpztseed) : Το μέγεθος του seed καθορίζει το πόσο πολλές διαφορετικές ακολουθίες τυχαίων αριθμών είναι δυνατό να δημιουργηθούν.

void mpz_set_ui (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

void mpz_urandomb (mpzt rop, gmprandstatetstate, mpbitcnttn) : Δημιουργεί έναν τυχαίο ακέραιο αριθμό της τάξεως του 0 ως το 2n − 1

```
//------(Εξετάζει αν είναι πρώτος ο q και αν είναι 3 mod 4, αλλιώς βρίσκω τον
επόμενο πρώτο που να μην είναι ίδιος με τον p)---------

primetest = mpz_probab_prime_p(q1, 5);

if (primetest != 0)

{

mpz_set(q, q1);

mpz_sub_ui(qmod,q,3);

mpz_mod_ui(qmod, qmod, 4);
```

```
if (mpz_cmp_ui(qmod,0)!=0)
{
goto sequence2;
}
else
{
printf("q= "); mpz_out_str(stdout, 10, q); //Ektupwsh p
printf("\n");
printf("\n");
}
}
else
{
mpz_sub_ui(qmod,q,3);
mpz_mod_ui(qmod, qmod, 4);
if (mpz_cmp_ui(qmod,0)!=0)
{
goto sequence2;
}
else
{
printf("q= "); mpz_out_str(stdout, 10, q); //Ektupwsh q
printf("\n");
printf("\n");
}
}
```

intmpz_probab_prime_p (mpztn, intreps) : Με την συγκεκριμένη εντολή μπορούμε να ελέγξουμε αν ένας αριθμός είναι πρώτος

void mpz_set (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

unsigned long int mpz_mod_ui (mpz t r, mpz t n, unsigned long int d): Θέτει το n mod d στην μεταβλητή r

intmpz_cmp_ui (mpztop1, unsignedlongintop2): Η συγκεκριμένη μάκρο εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1<op2

size_tmpz_out_str (FILE *stream, intbase, mpztop): Με αυτήν την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
//------(Ypologismos tou n=p*q)-----------------------------------------------
-------------------------------------------------

mpz_mul(n, p, q);

printf("n= "); mpz_out_str(stdout, 10, n);            //Εκτύπωση n=p*q

printf("\n");

printf("\n");
```

void mpz_mul (mpz t rop, mpz t op1, mpz t op2): Ορίζει στο rop την τιμή op1 X op2

```
//------(Εκτυπώσεις κλειδιών)--------------------------------------------------
-------------------------------------------------

printf("To Idiwtiko kleidi einai:\n");          // Εκτύπωση του Ιδιωτικού
κλειδιού

printf("(p,q)= ");

printf(" (");

mpz_out_str(stdout, 10, p);

printf("  ,  ");

mpz_out_str(stdout, 10, q);
```

```
printf(") ");

printf("\n");

printf("\n");

printf("To Dhmosio Kleidi einai:\n");          // Εκτύπωση του Δημόσιου
κλειδιού

printf("n= ");

mpz_out_str(stdout, 10, n);

printf("\n");

printf("\n");
```

size_tmpz_out_str (FILE *stream, intbase, mpztop):Με αυτή την εντολή μπορούμε να τυπώσουμε τα string του προγράμματος

```
//-----(Κλήση συνάρτησης μετατροπής κειμένου σε ακέραιο)--------------------
-------------------------------------

/* Input string */

printf("Input a string(Max length = %d): ", MAXLEN);

fgets(message, MAXLEN - 1, stdin);

str2int(m, message);

/* string -> integer*/

#ifdef HEX

gmp_printf("MHNYMA(%d)  -> %Zx\n\n", strlen(message), m);

#else

gmp_printf("MHNYMA(%d)  -> %Zd\n\n", strlen(message), m);

#endif


//------(Κρυπτογράφηση c=m^2modn)-----------------------------------------
-------------------------------------------------

mpz_powm_ui(c,m,2,n);
```

void mpz_powm_ui (mpz t rop, mpz t base, mpz t exp, mpz t mod): Ορίζει στην rop την τιμή base^exp mod mod.

```
//------(Αποκρυπτογράφηση)-----------------------------------------------------
-----------------------------------------------
      mpz_set(p_ek,p);
      mpz_add_ui(p_ek,p_ek,1);
      mpz_div_ui(p_ek,p_ek,4);
      mpz_set(q_ek,q);
      mpz_add_ui(q_ek,q_ek,1);
      mpz_div_ui(q_ek,q_ek,4);
      mpz_powm(r,c,p_ek,p);
      mpz_powm(s,c,q_ek,q);
      mpz_init(temp_p);
      mpz_init(temp_q);
      mpz_set(temp_p,p);
      mpz_set(temp_q,q);
      mpz_set(aps,a);
      mpz_mul(aps,aps,p);
      mpz_mul(aps,aps,s);
      mpz_set(bqr,b);
      mpz_mul(bqr,bqr,q);
      mpz_mul(bqr,bqr,r);
      mpz_set(root1,aps);
      mpz_add(root1,root1,bqr);
      mpz_mod(root1,root1,n);

      mpz_set(root2,n);
```

```
mpz_sub(root2,root2,root1);

mpz_set(root3,aps);

mpz_sub(root3,root3,bqr);

mpz_mod(root3,root3,n);

mpz_set(root4,n);

mpz_sub(root4,root4,root2);
```

void mpz_set (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

voidmpf_div_ui (mpftrop, mpftop1, unsignedlongintop2) : Ορίζουμε στο rop την τιμή op1/op2

void mpz_powm_ui (mpzt rop, mpztbase, mpztexp, mpztmod): Ορίζει στην rop την τιμή base^expmodmod.

voidmpf_mul (mpftrop, mpftop1, unsignedlongintop2) : Έχει την ίδια λειτουργία με την mpf_mul

void mpz_sub (mpzt rop, mpztop1, mpztop2) : Θέτουμε στο rop το op1-op2

```
//------(Συνάρτηση μετατροπής string σε integer)----------------------------
-----------------------------------------

void str2int(mpz_t ret, char *str)

{

long int str_len, j;

unsigned char C;


str_len = strlen(str);

if(str[str_len - 1] == '\n')

str[str_len - 1] = '\0';

str_len = strlen(str);

mpz_set_ui(ret, 0UL);                  //Αρχικοποίηση του ret σε 0
```

```
for(j = str_len - 1; j >= 0; j--)      // ret = str[str_len - 1] *
BASE^(str_len - 1) + ... + str[1] * BASE + str[0]

{

C = str[j];


mpz_mul_ui(ret, ret, (unsigned long)BASE);            //Me8odos Horner

mpz_add_ui(ret, ret, (unsigned long)C);

}

}
```

void mpz_mul_ui (mpz t rop, mpz t op1, mpz t op2): Ορίζει στο rop την τιμή op1 X op2

voidmpz_set_ui (mpztrop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

voidmpz_add_ui (MP_INT *sum, MP_INT *addend1, unsignedlongintaddend2): Με αυτήν την εντολή μπορούμε να δώσουμε το βήμα που θέλουμε να αυξάνεται μια μεταβλητή για παράδειγμα v= v+1

```
//------(Συνάρτηση μετατροπής integer σε string)----------------------------
-------------------------------------------

void int2str(char *str, mpz_t org_str_int)

{

long int str_len, i;           //

mpz_tmax_int, c_int, str_int; // Δήλωση μεταβλητών που θα χρησιμοποιηθούν


mpz_init(max_int);                  //Αρχικοποίηση

mpz_init(c_int);

mpz_init(str_int);


mpz_set(str_int, org_str_int);


mpz_set_ui(max_int, 1UL);           //Παίρνει το μήκος της εγγραφής

for(i = 0; i < MAXLEN; i++)
```

```
{
if(mpz_cmp(str_int, max_int) <= 0)
{
      str_len = i;
      break;
}
mpz_mul_ui(max_int, max_int, (unsigned long)BASE);
}

for(i = 0; i < str_len; i++)        //Μετατροπή σε γράμμα
{
mpz_mod_ui(c_int, str_int, (unsigned long)BASE);
mpz_sub(str_int, str_int, c_int);
mpz_tdiv_q_ui(str_int, str_int, (unsigned long)BASE);
str[i] = mpz_get_ui(c_int);
}
str[str_len] = '\0';
mpz_clear(max_int);
mpz_clear(c_int);
mpz_clear(str_int);
}
```

void mpz_set_ui (mpzt rop, unsignedlongintop) : Με αυτήν την εντολής μπορούμε να θέσουμε την τιμή της δεύτερης μεταβλητής στην πρώτη.

int mpz_cmp_ui (mpzt op1, unsigned long int op2): Η συγκεκριμένη μάκρο εντολή συγκρίνει τους 2 αριθμούς op1 και op2 και επιστρέφει θετική τιμή αν το op1>op2 , μηδενική τιμή αν το op1=op2 και αρνητική αν το op1<op2

voidmpf_mul_ui (mpftrop, mpftop1, mpftop2): Ορίζουμε στο rop την τιμή op1 Χop2

unsigned long int mpz_mod_ui (mpz t r, mpz t n, unsigned long int d): Θέτει το n mod d στην μεταβλητή r

void mpz_sub (mpz t rop, mpz t op1, mpz t op2) : Θέτουμε στο rop το op1-op2

void mpz_clear (mpztx): Αδειάζει τις μεταβλητές έτσι ώστε να μην τρώει μνήμη από τον υπολογιστή όταν δεν τις χρειαζόμαστε.

ΠΑΡΑΡΤΗΜΑΑ

Trial Division

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "gmp.h"

gmp_randstate_t stat;

int main(void)

{

int isprime1;

int isprime2;

long sd = 0;

mpz_t j, v, z, l;

mpz_t seed;

gmp_randinit(stat, GMP_RAND_ALG_LC, 120);

mpz_init(j);

mpz_init(z);

mpz_init(l);

mpz_init(v);

mpz_init(seed);

srand((unsigned)getpid());

sd=rand();

mpz_set_ui(seed, sd);

gmp_randseed(stat, seed);

mpz_urandomb(j, stat, 15);

isprime1 = mpz_probab_prime_p(j, 10);

mpz_init_set_ui(v,2);
```

```
        if (isprime1!=0)        {
                printf("THE COMPOSITE NUMBER j IS PRIME:");
                mpz_out_str(stdout, 10, j);
                printf("\n");
        }
        if (isprime1==0)        {
                printf("THE COMPOSITE NUMBER j IS:");
                mpz_out_str(stdout, 10, j);
                printf("\n");
                while (mpz_cmp(v,j)==-1)
                {
                    isprime2 = mpz_probab_prime_p(v, 10);
                    mpz_div(z,j,v);
                    mpz_mod(l,j,v);

                    if(isprime2!=0)                 {
                            if(mpz_cmp_ui(l,0)==0)
                            {
                                    printf("THE FIRST FACTORS ARE:");
                                    mpz_out_str(stdout, 10, v);
                                    printf("\n");
                            }
                      }
                mpz_add_ui(v,v,1);
                }
        }
return 0;
}
```

Pollard Rho

```
#include <stdio.h>

#include <stdlib.h>

#include "gmp.h"

gmp_randstate_t stat;

int main (int argc, char *argv[])

{

      int v;

      long sd = 0;

      int isprime;

      mpz_t z, r, a, b, p, q, total1, total2, total3, t;

      mpz_t seed;

      v=1;

      gmp_randinit (stat, GMP_RAND_ALG_LC, 120);

      mpz_init (z);

      mpz_init (r);

      mpz_init (p);

      mpz_init (q);

      mpz_init (total1);

      mpz_init (total2);

      mpz_init (total3);

      mpz_init (seed);

      mpz_init (t);

      srand ((unsigned) getpid());

      sd=rand();

      mpz_set_ui (seed,sd);

      gmp_randseed (stat, seed);

      mpz_urandomb (r, stat, 15);
```

```
mpz_init_set_si (a, 2);
mpz_init_set_si (b, 2);                mpz_init_set_si (z, 10000000);
isprime=mpz_probab_prime_p(r, 10);
if (isprime!=0)
{
      printf("THE RANDOM NUMBER IS PRIME:");
      mpz_out_str (stdout, 15, r);
      printf("\n");
}
if (isprime==0)
{
      printf("THE RANDOM COMPOSITE NUMBER IS PRIME:");
      mpz_out_str (stdout, 15, r);
      printf("\n");
      sequence1: for (v=1; mpz_cmp_ui(z,v); v++)
      {
            mpz_pow_ui (a, a, 2);
            mpz_add_ui (total1, a, 1);
            mpz_mod (a, total1, r);
            mpz_pow_ui (b, b, 2);
            mpz_add_ui (total2, p, 1);
            mpz_mod (b, total2, r);
            mpz_pow_ui (b, b, 2);
            mpz_add_ui (total3, p, 1);
            mpz_mod (b, total3, r);
            mpz_sub (t, a, b);
            mpz_gcd (p, t, r);
```

```
                if ((mpz_cmp_si(p,1)) && (mpz_cmp (r,p)))
                {
                    printf("THIS IS A PRIME FACTOR OF THE COMPOSITE
NUMBER:");
                    mpz_out_str (stdout, 10, p);
                    printf("\n");
                    mpz_div (q, r, p);
                    isprime=mpz_probab_prime_p  (q, 10);
                    if (isprime!=0)
                    {
                        printf("THIS IS A PRIME FACTOR OF THE COMPOSITE
NUMBER:");
                        mpz_out_str (stdout, 10, q);
                        printf("\n");
                        goto sequence2;
                    }
                    else
                    {
                        mpz_set(r,q);
                        goto sequence1;
                    }
                    goto sequence2;
                }
                if (mpz_cmp_si(p,1))
                {
                    goto sequence1;
                }
                if (p==r)
                {
```

```
                        printf("FAIL");
                        goto sequence2;
                }
        }
}
sequence2:
return 0;
}
```

Pollard Rho p-1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gmp.h"
gmp_randstate_t stat;
int main(void)
{
      mpf_t w1;
      #define computeln2(w1)
      {
            long v;
            mpz_t z4;
            mpf_t s5, s1, t2, t1, half1;
            mpf_set_default_prec(10);
            mpz_init(z4);
            mpf_init(s5);
            mpf_init(s1);
```

```
        mpf_init(t2);
        mpf_init(t1);
        mpf_init(half1);
        mpz_init_set_si (z4, 5);
        mpf_set_ui(s5, 0);
        mpf_set_str(t2, "0.5e0", 10);
        mpf_set_str(t1, "0.5e0", 10);
        mpf_set_str(half1, "0.5e0", 10);

        for(v = 2; mpz_cmp_ui(z4, v); v++)
        {
              mpf_add(s1, s5, t2);
              if(mpf_cmp(s5, s1) == 0)
                    break;
              mpf_set(s5, s1);
              mpf_mul(t1, t1, half1);
              mpf_div_ui(t2, t1, v);
        }
        mpf_set(w1, s5);
}
mpf_t w, x;
#define myln(w,x);
{
        long h;
        mpz_t z3;
        mpf_t y, n1, help, help1, t3;
        mpf_t s3, s4, t4;
        mpf_set_default_prec(10);
```

```
mpf_init(x);
mpz_init(z3);
mpf_init(y);
mpf_init(n1);
mpf_init(help);
mpf_init(help1);
mpf_init(t3);
mpf_init(s3);
mpf_init(s4);
mpf_init(t4);
mpz_init_set_si (z3, 5);
mpf_set_str(help, "0.5e0", 10);
mpf_set_str(help1, "0.15e1", 10);
if(mpf_cmp_ui(x, 1) == -1)
{
      mpf_set_ui(n1, 0);
      mpf_ui_sub(y, 1, x);
}
if(mpf_cmp_ui(x, 1) == 0)
{
      mpf_set_ui(w, 0);
}
if(mpf_cmp_ui(x, 1) == 1)
{
      mpf_set_ui(n1, 0);
      mpf_set_str(t3, "0.5e0", 10);
      mpf_mul(t3, t3, x);
      while (mpf_cmp_ui(t3, 1) != -1)
```

```
        {
                mpf_div_ui(t3, t3, 2);
                mpf_add_ui(n1, n1, 1);
        }
        mpf_mul_ui(t3, t3, 2);
        mpf_ui_sub(y, 1, t3);
}
if(mpf_cmp(x, help) == -1)
{
        mpf_set_ui(n1, 0);
        mpf_set_str(t3, "0.2e1", 10);
        mpf_mul(t3, t3, x);
        while (mpf_cmp_ui(t3, 1) == -1)
        {
                mpf_mul_ui(t3, t3, 2);
                mpf_sub_ui(n1, n1, 1);
        }
        mpf_div_ui(t3, t3, 2);
        mpf_ui_sub(y, 1, t3);
}
mpf_set_ui(s3, 0);
mpf_set(t3, y);
mpf_set(t4, y);
for(h = 2; mpz_cmp_ui(z3, h); h++)
{
        mpf_add(s4, s3, t3);
        if(mpf_cmp(s3, s4) == 0)
                break;
```

```
            mpf_set(s3, s4);
            mpf_mul(t4, t4, y);
            mpf_div_ui(t3, t4, h);
      }
      if(mpf_sgn(n1) == 0)
      {
            mpf_set_ui(t3, 0);
      }
      else
      {
            computeln2(&t3);
            mpf_mul(t3, t3, n1);
      }
      mpf_sub(w, t3, s3);
}
int isprime1;
int isprime2;
int isprime3;
long sd=0;
mpz_t B, c, d, i, j, jv, m, r, u;
mpf_t l1, l2, z1, z2;
mpz_t seed;
gmp_randinit(stat,GMP_RAND_ALG_LC,120);
mpf_init (l1);
mpf_init (l2);
mpf_init (z1);
mpf_init (z2);
mpz_init (B);
```

```
mpz_init (c);
mpz_init (d);
mpz_init (i);
mpz_init (j);
mpz_init (jv);
mpz_init (m);
mpz_init (r);
mpz_init (u);
mpz_init (seed);
srand((unsigned)getpid());
sd=rand();
mpz_set_ui(seed,sd);
gmp_randseed(stat,seed);
mpz_urandomb(j,stat,15);
isprime1=mpz_probab_prime_p(j,10);
mpz_init_set_ui(B,1000);
mpz_init_set_ui(u,2);
mpz_init_set_ui(i,0);
mpz_sub_ui(jv,j,1);
mpz_urandomb(c,stat,15);
if (isprime1==0)
{
      printf("THE COMPOSITE j IS:");
      mpz_out_str(stdout,10,j);
      printf("\n");

      if ((mpz_cmp_ui(c,2)==1) || (mpz_cmp_ui(c,2)==0))
      {
```

```
if ((mpz_cmp(c,jv)==-1) || (mpz_cmp(c,jv)==0))
{
      mpz_gcd(d,c,j);
      isprime3=mpz_probab_prime_p(d,10);
      if ((mpz_cmp_ui(d,2)==1) || (mpz_cmp_ui(d,2)==0))
      {
            if (isprime3!=0)

                  printf("THE d IS:");
                  mpz_out_str(stdout,10,d);
                  printf("\n");
            }
            else
            {
                  goto sequence2;
            }
      }
      else                        {
            while ((mpz_cmp(u,B)==-1)||(mpz_cmp(u,B)==0))
            {
                  isprime2=mpz_probab_prime_p(u,10);
                  if (isprime2!=0)
                  {
                     myln(z1, n);
                     myln(z2, u);
                     mpf_div (l2, z1, z2);
                     mpf_trunc(l1,l2);
                     mpz_set_f (r, l1);
```

```
                                        while ((mpz_cmp(i,r)==-
1)||(mpz_cmp(i,r)==0))

                                                {

                                           mpz_mul(u,u,u);

                                                    mpz_add_ui(i,i,1);

                                                }

                                           mpz_powm(c,c,u,j);

                                       }

                                                mpz_add_ui(u,u,1);

                                   }

                                   mpz_sub_ui(m,c,1);

                                     mpz_gcd(d,m,j);

                                     isprime3=mpz_probab_prime_p(d,10);

                                     if (isprime3!=0)

                                     {

                                   if ((mpz_cmp_ui(d,1)==0) ||
(mpz_cmp(d,j)==0))

                                   {

                                         printf("FAIL\n");

                                         goto sequence2;

}

                                   else

                                   {

                                         printf("THE d IS:");

                                                  mpz_out_str(stdout,10,d);

printf("\n");

                                           }

                                     }

                                   else
```

```
                                {
goto sequence2;
                                }
                        }
                }
                else
                {
goto sequence2;
                }
        }
        else
        {
goto sequence2;
        }
    }
    else
    {
        printf("THE PRIME NUMBER j IS:");
        mpz_out_str(stdout,10,j);
printf("\n");
    }
sequence2:
return 0;
}
```

ΠΑΡΑΡΤΗΜΑΒ

RSA

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "gmp.h"

#define BASE 256

#ifndef MAXLEN

#define MAXLEN 256

#endif

void str2int(mpz_t ret, char *str);

void int2str(char *str, mpz_t org_str_int);

void euclides(mpz_t a,mpz_t b,mpz_t xx,mpz_t yy);

 gmp_randstate_t stat;

 int main()

{

       gmp_randstate_t rand_state;

      gmp_randinit_default(rand_state);

      long sd = 0;

       mpz_t p, q, p1, q1, n, p_1, q_1, f, e, gcd, d, c, m, A, B, C, D, i,
ed, ed_m, ph11, ph12, dia1, dia2, message_int, a, b, temp_p, temp_q;

      mpz_t seed;

gmp_randinit(stat, GMP_RAND_ALG_LC, 120);

      mpz_init(p);

      mpz_init(q);

      mpz_init(p1);

      mpz_init(q1);

      mpz_init(n);
```

```
mpz_init(p_1);
mpz_init(q_1);
mpz_init(f);
mpz_init(e);
mpz_init(gcd);
mpz_init(d);
mpz_init(c);
mpz_init(A);
mpz_init(m);
mpz_init(B);
mpz_init(C);
mpz_init(D);
mpz_init(i);
mpz_init(ed);
mpz_init(ed_m);
mpz_init(phl1);
mpz_init(phl2);
mpz_init(dia1);
mpz_init(dia2);
 mpz_init(message_int);
 mpz_init(seed);
 unsigned char message[MAXLEN], out_message[MAXLEN];
mpz_init(a);
mpz_init(b);
mpz_init(temp_p);
mpz_init(temp_q);
 srand( (unsigned) getpid());
sd=rand();
```

```
 mpz_set_ui(seed, sd);
 gmp_randseed(stat, seed);
  srand( (unsigned) getpid());
 mpz_set_ui(seed, sd);
 mpz_urandomb(p1, stat, 512);
 srand( (unsigned) getpid());
 sd=rand();
 mpz_set_ui(seed, sd);
 gmp_randseed(stat, seed);
srand( (unsigned) getpid());
 mpz_set_ui(seed, sd);
 mpz_urandomb(q1, stat, 512);
int primetest;
 primetest = mpz_probab_prime_p(p1, 5);
 if (primetest != 0)
 {
       mpz_set(p, p1);
       printf("p= "); mpz_out_str(stdout, 10, p);
       printf("\n");
       printf("\n");
 }
 else
 {
       mpz_nextprime(p, p1);
       printf("p= "); mpz_out_str(stdout, 10, p);
       printf("\n");
       printf("\n");
```

```
}
primetest = mpz_probab_prime_p(q1, 5);
if (primetest != 0)
{
      mpz_set(q, q1);
      printf("q= "); mpz_out_str(stdout, 10, q);
      printf("\n");
      printf("\n");
}
else
{
      mpz_nextprime(q, p);
      printf("q= "); mpz_out_str(stdout, 10, q);
      printf("\n");
      printf("\n");

}
mpz_mul(n, p, q);
printf("n= "); mpz_out_str(stdout, 10, n);
printf("\n");
printf("\n");
mpz_sub_ui(p_1, p, 1);
printf("p-1= "); mpz_out_str(stdout, 10, p_1);
printf("\n");
printf("\n");
mpz_sub_ui(q_1, q, 1);
printf("q-1= "); mpz_out_str(stdout, 10, q_1);
printf("\n");
```

```
	printf("\n");
	mpz_mul(f, p_1, q_1);
	printf("f= "); mpz_out_str(stdout, 10, f);
	printf("\n");
	printf("\n");
	sequence1:
	mpz_urandomb(e, stat, 512);
	if ((mpz_cmp_si(e,1)) && (mpz_cmp (e,f)))
	{
		mpz_gcd(gcd, e, f);
		if (mpz_cmp_si(gcd,1))
		{
			printf("To e einai:\n");
			printf("e= ");
			mpz_out_str(stdout, 10, e);
			printf("\n");
			printf("\n");
		}
		else
		{
			goto sequence1;
		}
	}
	else
	{
		goto sequence1;
	}
mpz_powm_ui(d, e, -1, f);
```

```
printf("To d einai:\n");
printf("d= ");
mpz_out_str(stdout, 10, d);
printf("\n");
printf("\n");
printf("To Dhmosio kleidi einai:\n");
printf("(n,e)= ");
printf(" (");
mpz_out_str(stdout, 10, n);
printf("  ,  ");
mpz_out_str(stdout, 10, e);
printf(") ");
printf("\n");
printf("\n");
printf("To Idiwtiko Kleidi einai:\n");
printf("d= ");
mpz_out_str(stdout, 10, d);
printf("\n");
printf("\n");
  printf("Input a string(Max length = %d): ", MAXLEN);
  fgets(message, MAXLEN - 1, stdin);
str2int(m, message);
#ifdef HEX
  gmp_printf("MHNYMA(%d)  -> %Zx\n\n", strlen(message), m);
#else
  gmp_printf("MHNYMA(%d)  -> %Zd\n\n", strlen(message), m);
#endif
```

```
	mpz_powm(c, m, e, n);

	exit1:

	printf("To ciphertext einai:\n");

	printf("c= ");

	mpz_out_str(stdout, 10, c);

	printf("\n");

	printf("\n");

	mpz_powm(m, c, d, n);

	exit2:

	printf("To plaintext einai:\n");

	printf("m= ");

	mpz_out_str(stdout, 10, m);

	printf("\n");

	printf("\n");

	int2str(out_message, m);

	#ifdef HEX

	  gmp_printf("Apokuptografhmeno mhnuma: %Zd\n            -> %s(%d)\n",
m, out_message, strlen(out_message));

	#else

	  gmp_printf("Apokuptografhmeno mhnuma: %Zd\n            -> %s(%d)\n",
m, out_message, strlen(out_message));

	#endif

return 0;

}


void str2int(mpz_t ret, char *str)

{

	long int str_len, j;

	unsigned char C;
```

```
	str_len = strlen(str);
	if(str[str_len - 1] == '\n')
		str[str_len - 1] = '\0';
	str_len = strlen(str);
	mpz_set_ui(ret, 0UL);
	for(j = str_len - 1; j >= 0; j--)
	{
	C = str[j];
		mpz_mul_ui(ret, ret, (unsigned long)BASE);
		mpz_add_ui(ret, ret, (unsigned long)C);
	}
}

void int2str(char *str, mpz_t org_str_int)
{
	long int str_len, i;
	mpz_t max_int, c_int, str_int;
	mpz_init(max_int);
	mpz_init(c_int);
	mpz_init(str_int);
	mpz_set(str_int, org_str_int);
	mpz_set_ui(max_int, 1UL);
	for(i = 0; i < MAXLEN; i++)
	{
		if(mpz_cmp(str_int, max_int) <= 0)
		{
			str_len = i;
			break;
```

```
            }
            mpz_mul_ui(max_int, max_int, (unsigned long)BASE);
      }
      for(i = 0; i < str_len; i++)
      {
            mpz_mod_ui(c_int, str_int, (unsigned long)BASE);
            mpz_sub(str_int, str_int, c_int);
            mpz_tdiv_q_ui(str_int, str_int, (unsigned long)BASE);

            str[i] = mpz_get_ui(c_int);
      }
      str[str_len] = '\0';
      mpz_clear(max_int);
      mpz_clear(c_int);
      mpz_clear(str_int);
}

void euclides(mpz_t a,mpz_t b,mpz_t xx,mpz_t yy)
{
      int flag=0;
      mpz_t d,x,y,x1,x2,y1,y2,q,r;
      mpz_t qb,a_qb,temp;
      mpz_t qx1,x2_qx1;
      mpz_t qy1,y2_qy1;
      mpz_init(x);
      mpz_init(y);
      mpz_init(d);
```

```
if(mpz_cmp(b,a)>0) {
        flag=1;
        mpz_init(temp);
        mpz_set(temp,b);
        mpz_set(b,a);
        mpz_set(a,temp);
        mpz_clear(temp);
}
if(mpz_cmp_ui(b,0)==0) {
        mpz_set(d,a);
        mpz_set_ui(x,1);
        mpz_set_ui(y,0);
}
else {
        mpz_init(q);
        mpz_init(r);
        mpz_init(x1);
        mpz_init(x2);
        mpz_init(y1);
        mpz_init(y2);

        mpz_init(qb);
        mpz_init(a_qb);
        mpz_init(qx1);
        mpz_init(qy1);
        mpz_init(x2_qx1);
        mpz_init(y2_qy1);
```

```
		mpz_set_ui(x2,1);
		mpz_set_ui(x1,0);
		mpz_set_ui(y1,1);
		mpz_set_ui(y2,0);
		while(mpz_cmp_ui(b,0)>0) {
			mpz_tdiv_q(q,a,b);
			mpz_mul(qb,q,b);
			mpz_sub(a_qb,a,qb);
			mpz_set(r,a_qb);
			mpz_mul(qx1,q,x1);
			mpz_sub(x2_qx1,x2,qx1);
			mpz_set(x,x2_qx1);
			mpz_mul(qy1,q,y1);
			mpz_sub(y2_qy1,y2,qy1);
			mpz_set(y,y2_qy1);
			mpz_set(a,b);
			mpz_set(b,r);
			mpz_set(x2,x1);
			mpz_set(x1,x);
			mpz_set(y2,y1);
			mpz_set(y1,y);
		}
	}
	mpz_set(d,a);
	mpz_set(x,x2);
	mpz_set(y,y2);
	if(flag==1){
		mpz_set(yy,x);
```

```
            mpz_set(xx,y);
        }
        else {
            mpz_set(yy,y);
            mpz_set(xx,x);
        }
        mpz_clear(d);
        mpz_clear(x);
        mpz_clear(y);
        mpz_clear(x1);
        mpz_clear(x2);
        mpz_clear(y1);
        mpz_clear(y2);
        mpz_clear(q);
        mpz_clear(r);
        mpz_clear(qb);
        mpz_clear(a_qb);
        mpz_clear(qx1);
        mpz_clear(x2_qx1);
        mpz_clear(qy1);
        mpz_clear(y2_qy1);
}
```

Rabin

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "gmp.h"
#define BASE 256
#ifndef MAXLEN
#define MAXLEN 256
#endif
void str2int(mpz_t ret, char *str);
void int2str(char *str, mpz_t org_str_int);
void euclides(mpz_t a,mpz_t b,mpz_t xx,mpz_t yy);
gmp_randstate_t stat;
int main()
{
	mpz_t temp_p,temp_q, message_int;
	int root_no;
	mpz_t c;
	mpz_init(c);
	mpz_t a;
	mpz_init(a);
	mpz_t b;
	mpz_init(b);
	mpz_t m;
	mpz_init(m);
	mpz_t(p_ek);
	mpz_init(p_ek);
	mpz_t(q_ek);
	mpz_init(q_ek);
	mpz_t (r);
	mpz_init(r);
	mpz_t (s);
```

```
mpz_init(s);

mpz_t root1;

mpz_init(root1);

mpz_t root2;

mpz_init(root2);

mpz_t root3;

mpz_init(root3);

mpz_t root4;

mpz_init(root4);

mpz_t aps;

mpz_init(aps);

mpz_t bqr;

mpz_init(bqr);

gmp_randstate_t rand_state;

gmp_randinit_default(rand_state);

long sd = 0;

mpz_t p, q, p1, q1, n, pmod, qmod;

mpz_t seed;

gmp_randinit(stat, GMP_RAND_ALG_LC, 120);

mpz_init(p);

mpz_init(q);

mpz_init(p1);

mpz_init(q1);

mpz_init(n);

mpz_init(pmod);

mpz_init(qmod);

mpz_init(seed);
```

```
	mpz_init(message_int);
	mpz_init(seed);
unsigned char message[MAXLEN], out_message[MAXLEN];
	srand( (unsigned) getpid());
	sd=rand();
	mpz_set_ui(seed, sd);
	gmp_randseed(stat, seed);
	srand( (unsigned) getpid());
	mpz_set_ui(seed, sd);
	sequence1:
	mpz_urandomb(p1, stat, 200);
	int primetest;
	primetest = mpz_probab_prime_p(p1, 5);
	if (primetest != 0)
	{
		mpz_set(p, p1);
		mpz_sub_ui(pmod,p,3);
		mpz_mod_ui(pmod, pmod, 4);
		if (mpz_cmp_ui(pmod,0)!=0)
		{
			goto sequence1;
		}
		else
		{
			printf("p= "); mpz_out_str(stdout, 10, p); //Ektupwsh p
			printf("\n");
			printf("\n");
		}
```

```
}
else
{
      mpz_nextprime(p, p1);
      mpz_sub_ui(pmod,p,3);
      mpz_mod_ui(pmod, pmod, 4);
      if (mpz_cmp_ui(pmod,0)!=0)
      {
            goto sequence1;
      }
      else
      {
            printf("p= "); mpz_out_str(stdout, 10, p); //Ektupwsh p
            printf("\n");
            printf("\n");
      }
}
srand( (unsigned) getpid());
sd=rand();
mpz_set_ui(seed, sd);
gmp_randseed(stat, seed);
srand( (unsigned) getpid());
mpz_set_ui(seed, sd);
sequence2:
mpz_urandomb(q1, stat, 200);
primetest = mpz_probab_prime_p(q1, 5);
if (primetest != 0)
{
```

```
        mpz_set(q, q1);
        mpz_sub_ui(qmod,q,3);
        mpz_mod_ui(qmod, qmod, 4);
        if (mpz_cmp_ui(qmod,0)!=0)
        {
              goto sequence2;
        }
        else
        {
              printf("q= "); mpz_out_str(stdout, 10, q);
              printf("\n");
              printf("\n");
        }
}
else
{
        mpz_sub_ui(qmod,q,3);
        mpz_mod_ui(qmod, qmod, 4);
        if (mpz_cmp_ui(qmod,0)!=0)
        {
              goto sequence2;
        }
        else
        {
              printf("q= "); mpz_out_str(stdout, 10, q);
              printf("\n");
              printf("\n");
        }
```

```
}
mpz_mul(n, p, q);
printf("n= "); mpz_out_str(stdout, 10, n);
printf("\n");
printf("\n");
printf("To Idiwtiko kleidi einai:\n");
printf("(p,q)= ");
printf(" (");
mpz_out_str(stdout, 10, p);
printf("  ,  ");
mpz_out_str(stdout, 10, q);
printf(") ");
printf("\n");
printf("\n");
printf("To Dhmosio Kleidi einai:\n");
printf("n= ");
mpz_out_str(stdout, 10, n);
printf("\n");
printf("\n");
  printf("Input a string(Max length = %d): ", MAXLEN);
  fgets(message, MAXLEN - 1, stdin);
str2int(m, message);
#ifdef HEX
  gmp_printf("MHNYMA(%d)  -> %Zx\n\n", strlen(message), m);
#else
  gmp_printf("MHNYMA(%d)  -> %Zd\n\n", strlen(message), m);
#endif
```

```
mpz_set(p_ek,p);

mpz_add_ui(p_ek,p_ek,1);

mpz_div_ui(p_ek,p_ek,4);

mpz_set(q_ek,q);

mpz_add_ui(q_ek,q_ek,1);

mpz_div_ui(q_ek,q_ek,4);

mpz_powm(r,c,p_ek,p);

mpz_powm(s,c,q_ek,q);

mpz_init(temp_p);

mpz_init(temp_q);

mpz_set(temp_p,p);

mpz_set(temp_q,q);

euclides(temp_p,temp_q,a,b);

mpz_set(aps,a);

mpz_mul(aps,aps,p);

mpz_mul(aps,aps,s);

mpz_set(bqr,b);

mpz_mul(bqr,bqr,q);

mpz_mul(bqr,bqr,r);

mpz_set(root1,aps);

mpz_add(root1,root1,bqr);

mpz_mod(root1,root1,n);

mpz_set(root2,n);

mpz_sub(root2,root2,root1);

mpz_set(root3,aps);

mpz_sub(root3,root3,bqr);

mpz_mod(root3,root3,n);
```

```
        mpz_set(root4,n);

        mpz_sub(root4,root4,root2);

        printf("\n\nMhnuma kruptografhshs: \n");

        printf("m = ");

        mpz_out_str(stdout, 10, m);

        printf("\n");

        printf("\n");

        printf("\nc = ");

        mpz_out_str(stdout, 10, c);

        printf("\n");

        printf("\n");

        printf("\n\nYpologismos twn a, b tetoia wste a*p + b*q = 1, r =
c^(p+1)/4 mod p, s = c^(q+1)/4 mod q\n");

        printf("a = ");

        mpz_out_str(stdout, 10, a);

        printf("\n");

        printf("\n");

        printf("\nb = ");

        mpz_out_str(stdout, 10, b);

        printf("\n");

        printf("\n");

        printf("\nr = ");

        mpz_out_str(stdout, 10, r);

        printf("\n");

        printf("\n");

        printf("\ns = ");

        mpz_out_str(stdout, 10, s);

        printf("\n");
```

```
	printf("\n");
	printf("\n\nYpologismos twn rizwn: x, -x, y, -y \n");
	printf("root 1: x = ");
	mpz_out_str(stdout, 10, root1);
	printf("\n");
	printf("\n");
	printf("\nroot 2: -x = ");
	mpz_out_str(stdout, 10, root2);
	printf("\n");
	printf("\n");
	printf("\nroot 3: y = ");
	mpz_out_str(stdout, 10, root3);
	printf("\n");
	printf("\n");
	printf("\nroot 4: -y = ");
	mpz_out_str(stdout, 10, root4);
	printf("\n");
	printf("\n");
	if (mpz_cmp(m, root1)==0)
	{
		root_no=1;
		printf("\n\nH swsth riza einai\n");
		printf("\nSugkrinontas me to mhnuma, h riza einai h root
%d\n",root_no);
		printf("\n");
		printf("\n");
	}
	if (mpz_cmp(m, root3)==0)
```

```
	{
		root_no=3;
		printf("\n\nH swsth riza einai\n");
		printf("\nSugkrinontas me to mhnuma, h riza einai h root
%d\n",root_no);
		printf("\n");
		printf("\n");
	}
	if (mpz_cmp(m, root2)==0)
	{
		root_no=2;
		printf("\n\nH swsth riza einai\n");
		printf("\nSugkrinontas me to mhnuma, h riza einai h root
%d\n",root_no);
		printf("\n");
		printf("\n");
	}
	if (mpz_cmp(m, root4)==0)
	{
		root_no=4;
		printf("\n\nH swsth riza einai\n");
		printf("\nSugkrinontas me to mhnuma, h riza einai h root
%d\n",root_no);
		printf("\n");
		printf("\n");
	}

	int2str(out_message, m);
	#ifdef HEX
```

```
        gmp_printf("Apokuptografhmeno mhnuma: %Zd\n                -> %s(%d)\n",
m, out_message, strlen(out_message));

      #else

        gmp_printf("Apokuptografhmeno mhnuma: %Zd\n                -> %s(%d)\n",
m, out_message, strlen(out_message));

      #endif

return 0;

}


void str2int(mpz_t ret, char *str)

{

      long int str_len, j;

      unsigned char C;

      str_len = strlen(str);

      if(str[str_len - 1] == '\n')

            str[str_len - 1] = '\0';

      str_len = strlen(str);

      mpz_set_ui(ret, 0UL);

      for(j = str_len - 1; j >= 0; j--)

      {

      C = str[j];

            mpz_mul_ui(ret, ret, (unsigned long)BASE);

            mpz_add_ui(ret, ret, (unsigned long)C);

      }

}


void int2str(char *str, mpz_t org_str_int)

{

      long int str_len, i;
```

```
	mpz_t max_int, c_int, str_int;
	mpz_init(max_int);
	mpz_init(c_int);
	mpz_init(str_int);
	mpz_set(str_int, org_str_int);
	mpz_set_ui(max_int, 1UL);
	for(i = 0; i < MAXLEN; i++)
	{
		if(mpz_cmp(str_int, max_int) <= 0)
		{
			str_len = i;
			break;
		}
		mpz_mul_ui(max_int, max_int, (unsigned long)BASE);
	}
	for(i = 0; i < str_len; i++)
	{
		mpz_mod_ui(c_int, str_int, (unsigned long)BASE);
		mpz_sub(str_int, str_int, c_int);
		mpz_tdiv_q_ui(str_int, str_int, (unsigned long)BASE);

		str[i] = mpz_get_ui(c_int);
	}
	str[str_len] = '\0';
	mpz_clear(max_int);
	mpz_clear(c_int);
	mpz_clear(str_int);
}
```

```
void euclides(mpz_t a,mpz_t b,mpz_t xx,mpz_t yy){
	int flag=0;
	mpz_t d,x,y,x1,x2,y1,y2,q,r;
	mpz_t qb,a_qb,temp;
	mpz_t qx1,x2_qx1;
	mpz_t qy1,y2_qy1;
	mpz_init(x);
	mpz_init(y);
	mpz_init(d);
	if(mpz_cmp(b,a)>0) {
		flag=1;
		mpz_init(temp);
		mpz_set(temp,b);
		mpz_set(b,a);
		mpz_set(a,temp);
		mpz_clear(temp);
	}
	if(mpz_cmp_ui(b,0)==0) {
		mpz_set(d,a);
		mpz_set_ui(x,1);
		mpz_set_ui(y,0);
	}
	else {
		mpz_init(q);
		mpz_init(r);
		mpz_init(x1);
		mpz_init(x2);
		mpz_init(y1);
```

```
mpz_init(y2);
mpz_init(qb);
mpz_init(a_qb);
mpz_init(qx1);
mpz_init(qy1);
mpz_init(x2_qx1);
mpz_init(y2_qy1);
mpz_set_ui(x2,1);
mpz_set_ui(x1,0);
mpz_set_ui(y1,1);
mpz_set_ui(y2,0);
while(mpz_cmp_ui(b,0)>0) {
      mpz_tdiv_q(q,a,b);
      mpz_mul(qb,q,b);
      mpz_sub(a_qb,a,qb);
      mpz_set(r,a_qb);
      mpz_mul(qx1,q,x1);
      mpz_sub(x2_qx1,x2,qx1);
      mpz_set(x,x2_qx1);
      mpz_mul(qy1,q,y1);
      mpz_sub(y2_qy1,y2,qy1);
      mpz_set(y,y2_qy1);
      mpz_set(a,b);
      mpz_set(b,r);
      mpz_set(x2,x1);
      mpz_set(x1,x);
      mpz_set(y2,y1);
      mpz_set(y1,y);
```

```
        }
    }
    mpz_set(d,a);
    mpz_set(x,x2);
    mpz_set(y,y2);
    if(flag==1){
        mpz_set(yy,x);
        mpz_set(xx,y);
    }
    else {
        mpz_set(yy,y);
        mpz_set(xx,x);
    }
    mpz_clear(d);
    mpz_clear(x);
    mpz_clear(y);
    mpz_clear(x1);
    mpz_clear(x2);
    mpz_clear(y1);
    mpz_clear(y2);
    mpz_clear(q);
    mpz_clear(r);
    mpz_clear(qb);
    mpz_clear(a_qb);
    mpz_clear(qx1);
    mpz_clear(x2_qx1);
    mpz_clear(qy1);
    mpz_clear(y2_qy1);
```

```
}
```

Βιβλιογραφία

Βιβλία

1. Σύγχρονη Κρυπτογραφία : Θεωρία και Εφαρμογές Εκδόσεις Παπασωτηρίου
2. Τεχνικές κρυπτογραφία &κρυπτανάλυσης Εκδόσεις Ζυγός
3. HandbookofappliedCryptographyby A. Menezes, P. vanOorschotand S. Vanstone
4. Gmp-man-5.0.2 manual
5. Βασικές αρχές θεωρίας κωδικοποίησης και κρυπτογραφίας Εκδόσεις Κλειδάριθμος,
6. Ασφάλεια της πληροφορίας Εκδόσεις Νέων Τεχνολογιών
7. Κρυπτογραφία Εκδόσεις Τραυλός
8. Σύγχρονη κρυπτογραφία Εκδόσεις Ελληνικά Γράμματα

Ηλεκτρονικές Πηγές

9. http://en.wikipedia.org/wiki/Trial_division
10. http://en.wikipedia.org/wiki/Pollard_rho
11. http://en.wikipedia.org/wiki/Pollard%27s_p_-_1_algorithm
12. http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html