Διαδικαστικός Προγραμματισμός

(ECE_Y215)

Επιπρόσθετες Βιβλιοθήκες στη C

Πρόγραμμα Διαλέξεων & Δραστηριοτήτων

Ημερομηνία	Περιεχόμενο
23 Απριλίου 2021 09:00 – 10:00	Εισαγωγή & GSL - GNU Scientific Library (1/2) • *εξειδικευμένος επιστημονικός υπολογισμός*
14 Μαΐου 2021 09:00 – 10:00	GSL - GNU Scientific Library (2/2) • *εξειδικευμένος επιστημονικός υπολογισμός*
21 Μαΐου 2021 09:00 – 10:00	**GMP - GNU MultiPrecision Library (1/2)** • ***αυθαίρετη αριθμητική ακρίβεια σε αριθμούς***
28 Μαΐου 2021 09:00 – 10:00	GMP - GNU MultiPrecision Library (2/2) • *αυθαίρετη αριθμητική ακρίβεια σε αριθμούς*

Δραστηριότητες	Παράδοση
1η Δραστηριότητα: 23 Απριλίου	10 Μαΐου 2021 (+ 0.25 μονάδες στον τελικό βαθμό)
2η Δραστηριότητα: 14 Μαΐου	28 Μαΐου 2021 (+ 0.25 μονάδες στον τελικό βαθμό)

Στόχοι

- Ο σκοπός της σημερινής διάλεξης είναι η παρουσίαση της **βιβλιοθήκης GMP**: *GNU MultiPrecision Library*
- Να αναφερθούμε σε **παραδείγματα** ώστε να εμπεδώσουμε καλύτερα τις θεωρητικές γνώσεις
- Να **ασχοληθούμε με προβλήματα** που απαιτούν αυθαίρετη αριθμητική ακρίβεια στους υπολογιστικούς υπολογισμούς:
  - **κρυπτογραφία**
  - ψηφιακές πιστοποιήσεις οικονομικών συναλλαγών
  - ασφάλεια κυβερνοχώρου

Γιατί χρειαζόμαστε Υπολογιστική Αριθμητική Αυθαίρετης Ακρίβειας

Πρόγραμμα το οποίο να βρίσκει τους πρώτους αριθμούς στο διάστημα [1,100].

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Πηγή της εικόνας: https://www.splashlearn.com

```
int start;
int end;
int k;
start=1;
end=100;
for (k = start; k <= end; k++) {
    if (k == 0 || k == 1) continue;

    for (c = 2; c <= k ; c++){
            if (k%c == 0) break;
     }
     if (c == k){
          printf("%d\n", k);
     }
}
```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

Θα λειτουργούσε σωστά το παραπάνω πρόγραμμα εύρεσης πρώτων αριθμών στο διάστημα [MAX_INT-10, MAX_INT] ;

Γιατί χρειαζόμαστε Υπολογιστική Αριθμητική Αυθαίρετης Ακρίβειας

Προσπαθήστε να γράψετε πρόγραμμα το οποίο να βρίσκει τους πρώτους αριθμούς στο διάστημα [MAX_INT-10, MAX_INT].

[32-bit:
start: 2147483637

32-bit:
end: 2147483647]

```
int start;
int end;
int k;
int c;
start=1;
end=100;
start=INT_MAX-10;
 end=INT_MAX;
 for (k = start; k <= end; k++) {
   if (k == 0 || k == 1) continue;

   for (c = 2; c <= k ; c++){
        if (k%c == 0) break;
    }
    if (c == k){
        printf("%d\n", k);
    }
}
```

```
2147483647
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
```

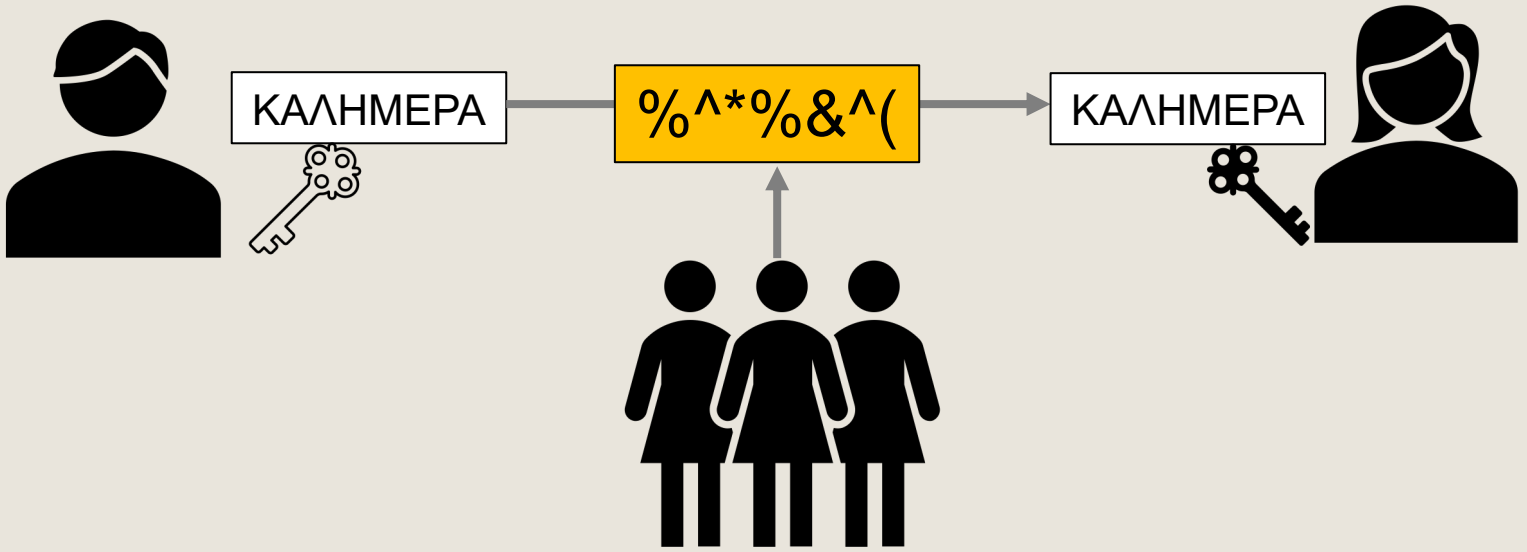
k: Integer overflow

```
k=-2147483648
k=-2147483647
k=-2147483646
k=-2147483645
```

Πως μπορώ να βρω *πρώτους αριθμούς* αυθαίρετης αριθμητικής ακρίβειας;

Γιατί χρειαζόμαστε μεγάλους πρώτους αριθμούς: Κρυπτογραφία

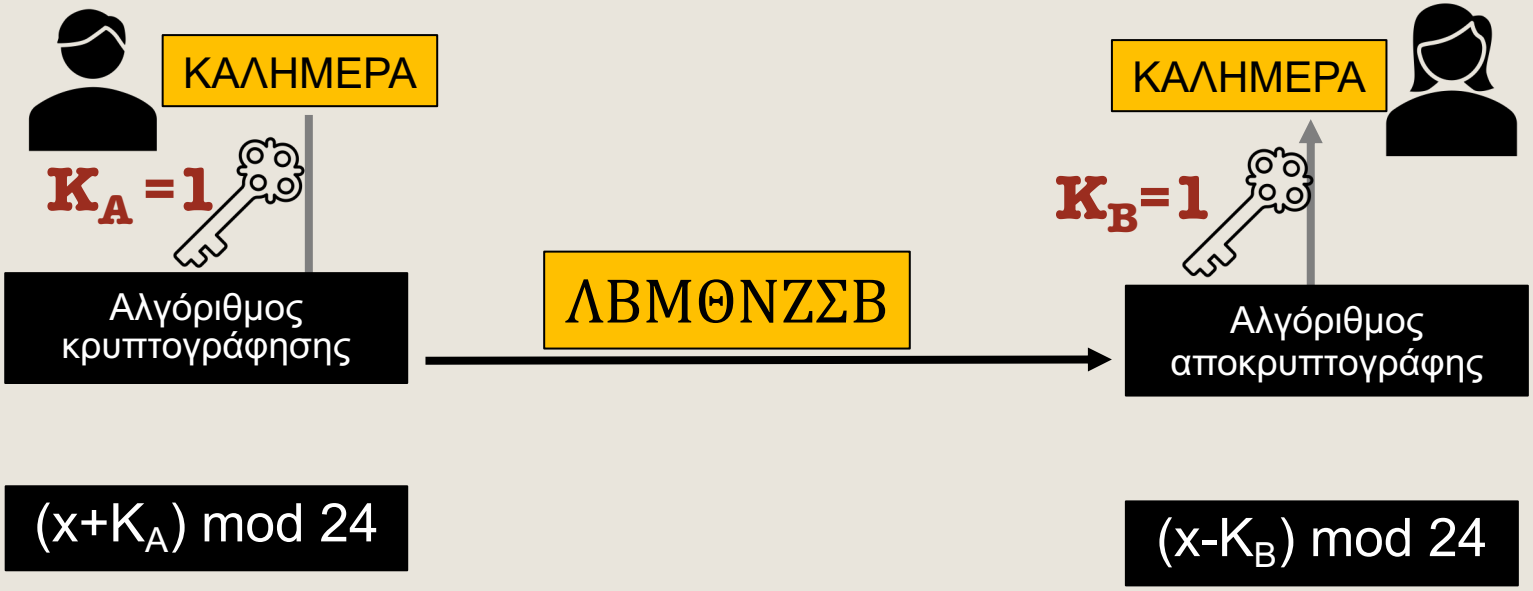
Το πρόβλημα της κρυπτογραφίας είναι ένα διαχρονικό πρόβλημα της ασφαλούς επικοινωνίας. Στόχος είναι αν το μήνυμα της επικοινωνίας υποκλαπεί να μην μπορεί να κατανοηθεί με εύκολο τρόπο.



https://en.wikipedia.org/wiki/Cryptography

Η Κρυπτογράφηση του Καίσαρα

- Σουητώνιος (Ιστορικός~120.π.Χ.): **Συμμετρικό ($K_A = K_B$)**



ΠΙΝΑΚΑΣ ΚΡΥΠΤΟΓΡΑΦΙΑΣ

Α 0	Β 1	Γ 2	...	Ω 23

https://en.wikipedia.org/wiki/Caesar_cipher

Γράψτε πρόγραμμα στη C που υλοποιεί τη μέθοδο κρυπτογράφησης του Καίσαρα βάση της κωδικοποίησης ASCII

Ο πηγαίος κώδικας του παραδείγματος βρίσκεται στο eclass: [lecture22] **CAESER-CIPHER**

ENCRYPT

```
for (i = 0; i < strlen(string); i++) {
        int ascii =(int) string[i];
        int new_ascii=(ascii+encryptionkey) % 128;
        char chiper_char = (char) new_ascii;
        enc_string[i] = chiper_char;
}
```

DECRYPT

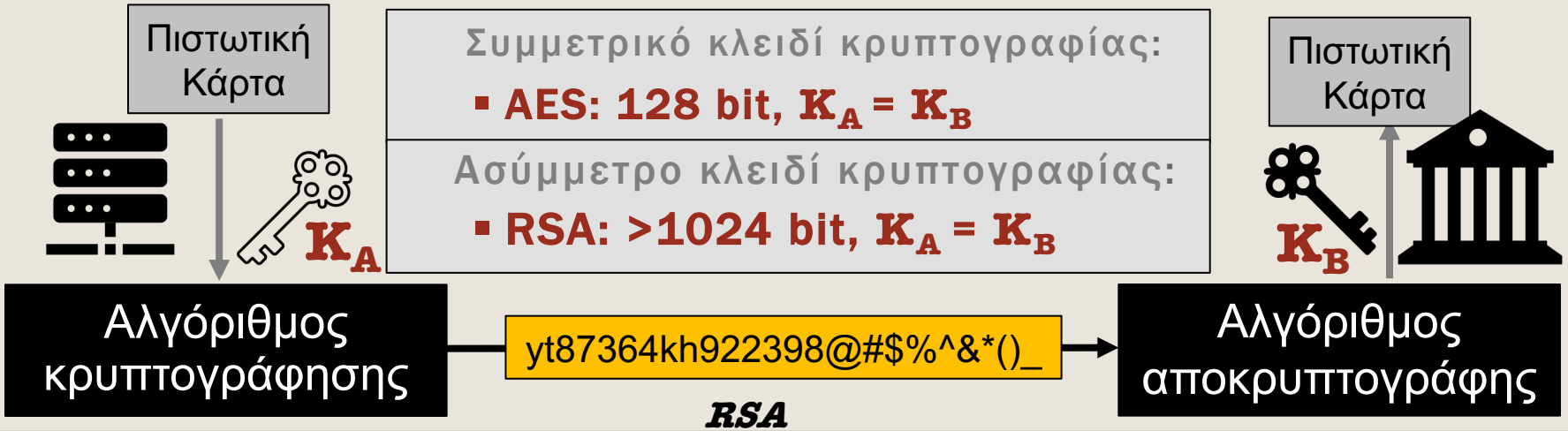
```
for ( i = 0; i < strlen(enc_string); i++) {
        int enc_ascii = (int) enc_string[i];
        int ascii=(enc_ascii-encryptionkey) % 128;
        char de_chiper_char = (char) ascii;
        dest_string[i] = de_chiper_char;
  }
```

```
MY DEAR ECE STUDENTS
end

ENTER ENCRYPTION KEY: 1

        CHIPERTEXT:
        NZ!EFBS!FDF!TUVEFOUT

DECRYPTION:
MY DEAR ECE STUDENTS
```

Θα μπορούσε ένας σύγχρονος υπολογιστής να βρει το κλειδί και να αποκωδικοποίηση το μήνυμά ;

https://en.wikipedia.org/wiki/Brute-force_attack

Κρυπτογραφία στο Διαδίκτυο: Συμμετρικό [AES] και Ασύμμετρο Κλειδί [RSA]



- Βρίσκουμε δυο μεγάλους (>1024 bit) πρώτους αριθμούς ***p, q.*** Υπολογίζουμε τον αριθμό ***N=p*q***
- Είναι αρκετά δύσκολο (*εκθετικός χρόνος ως προς το μήκος του N*) να βρεθούν οι **πρώτοι παράγοντες (p,q)** του αριθμού **N [The factoring problem]**
- Το πρώτο κλειδί το δίνουμε δημόσια για την κρυπτογράφηση του μηνύματος ενός το δεύτερο κλειδί το κρατάμε μυστικό για την αποκρυπτογράφηση [ασύμμετρη κρυπτογραφία]

https://en.wikipedia.org/wiki/RSA_(cryptosystem) *https://en.wikipedia.org/wiki/Advanced_Encryption_Standard*

Η Βιβλιοθήκη της GMP: Τύποι Μεταβλητών

Η διεπαφή για την πρόσβαση στις δομές δεδομένων και τις συναρτήσεις της GMP είναι η: <#include "gmp.h”>

- **Δήλωση μεταβλητών αυθαίρετης ακρίβειας:**

```
mpz_t x;
mpz_init (mpz_t x);
```

Δήλωση της ακέραιας μεταβλητής x τύπου mpz_t Αρχικοποίηση της μεταβλητής στην τιμή 0.

```
mpz_clear (mpz_t x);
```

Αποδέσμευση της μνήμης που σχετίζεται με τη μεταβλητή x

```
mpf_t fp;
mpq_t quotient;
```

Δήλωση μεταβλητής κινητής υποδιαστολής fp και Ρητού αριθμού quotient.

Η Βιβλιοθήκη της GMP: Ανάθεση Τιμών

Η διεπαφή για την πρόσβαση στις δομές δεδομένων και τις συναρτήσεις της GMP είναι η: <#include "gmp.h”>

- **Ανάθεση τιμών σε μεταβλητή αυθαίρετης ακρίβειας:**

void **mpz_set** *(MP_INT *dest_integer, MP_INT *src_integer)*

```
mpz_set_ui(x, z);
```

void **mpz_set_ui** *(MP_INT *integer, unsigned long int initial_value)*

```
mpz_set_ui(x, 1);
```

int **mpz_set_str** *(MP_INT *integer, char *initial_value, int base)*

```
mpz_init_set_str(
x, "98765432100123456789", 10);
```

https://gmplib.org/manual/

Η Βιβλιοθήκη της GMP: Αριθμητικές Πράξεις

Η διεπαφή για την πρόσβαση στις δομές δεδομένων και τις συναρτήσεις της GMP είναι η: <#include "gmp.h”>

```
mpz_t x;
mpz_init2(x,1024);
```

Δεσμεύονται 1024 bit για την αποθήκευση του ακέραιου αριθμού x

```
mpz_set_ui(x, 1);
```

Ανάθεση τιμής 1 στη μεταβλητή x

```
mpz_add_ui(x, 1);
mpz_sub_ui(x, 1);
mpz_mul_ui(x, 1);
```

Πρόσθεση, Αφαίρεση και Πολλαπλασιασμός της τιμής 1 στη μεταβλητή x

```
mpz_mul_exp(x, x, 19937);
```

Από τη 1η Δραστηριότητα:
Περίοδος Mersenne Twister
$x=x*2^{19937}$

https://gmplib.org/manual/Integer-Arithmetic

Η Βιβλιοθήκη της GMP: Σύγκριση Μεταβλητών

Η διεπαφή για την πρόσβαση στις δομές δεδομένων και τις συναρτήσεις της GMP είναι η: <#include "gmp.h”>

- **Σύγκριση μεταβλητών αυθαίρετης ακρίβειας:**

```
int mpz_cmp (const mpz_t op1, const mpz_t op2)
```

Η επιστρεφόμενη τιμή είναι:
- **>0 για op1>op2**
- **=0 για op1=op2**
- **<0 για op1<op2**

```
int mpz_cmp_si (const mpz_t op1, signed long int op2)
```

Παρομοίως με παραπάνω με τη διαφορά ότι συγκρίνει με προσημασμένο ακέραιο

```
int mpz_cmp_ui (const mpz_t op1, unsigned long int op2)
```

Παρομοίως με παραπάνω με τη διαφορά ότι συγκρίνει με μη-προσημασμένο ακέραιο

https://gmplib.org/manual/

1ο Παράδειγμα με τη GMP

Ο πηγαίος κώδικας του παραδείγματος βρίσκεται στο eclass: [lecture22] **GMP-ARBITRARY-INTEGER**

Πρόγραμμα το οποίο σας επιτρέπει να προσδιορίζεται αυθαίρετα τον αριθμό σε bits που θέλετε να δεσμεύεστε για την αποθήκευση ενός ακεραίου αριθμού και να εκτυπώνει τον μέγιστο αριθμό

```c
#include <stdio.h>
#include "gmp.h"
int main()
{
    int bits_for_int;
    printf("Enter an arbitrary number of bits
    scanf("%d",&bits_for_int);
    bits_for_int=bits_for_int-1;

    mpz_t res;
    mpz_init_set_ui(res,1);

    mpz_mul_2exp(res, res , bits_for_int);
    mpz_sub_ui (res, res, 1);

    mpz_out_str(stdout,10,res);
    printf("\n");
    mpz_clear(res);
    return 0;
}
```

```
Enter an arbitrary number of bits for a signed integer: 32
2147483647
```

```
Enter an arbitrary number of bits for a signed integer: 1024
89884656743115795386465259539451236680898848947115328636715040578866337902750481
56635423866120376801056005693993569667882939488440720831124642371531973706218888
39467124327426381511098006230470597265414760425028844190753411712314407369565552
70413618581675255342293149119973622969239858152417678164812112068607
```

```
Enter an arbitrary number of bits for a signed integer: 2054
10341441942819522336228760540374384627342112854308954890281710536807889644437725
08582310445168733230838038974749500736623811825892815709475042820876366345059860
18807575363187493600632801246886741747384442829016218485735099656383219717881901
88312374550586126672651617613724024285293141058754416767719657723367012586684693
86957461919193170066223835397633130442305858489781098592838166670730052808274971
65366773523019531899077943544314723687365266894091832362346519139926802191074624
67112050272552618023510297133625810143339214719125872959564228968053244619947373
95268333863347348650191315358468742193787315553907079380991
```

2ο Παράδειγμα με τη GMP

Ο πηγαίος κώδικας του παραδείγματος βρίσκεται στο eclass: [lecture22] **GMP-FIND-PRIME-NUMBERS**

Προσπαθήστε να γράψετε πρόγραμμα το οποίο να βρίσκει τους πρώτους αριθμούς στο διάστημα [MAX_INT-2, MAX_INT+13].

[32-bit:2147483645 32-bit: 2147483660]

```
#include <stdio.h>
#include "gmp.h"
int main()
{
   mpz_t mp_number,mp_end,mp_modulus,mp_c;
   mpz_init_set_ui(mp_number,2147483645);
   mpz_init_set_ui(mp_end,2147483660);
   while(mpz_cmp(mp_end,mp_number)>=0){
        mpz_init_set_ui(mp_c,2);
        while(mpz_cmp(mp_number,mp_c)>=0){
                mpz_mod (mp_modulus, mp_number, mp_c);
                if (mpz_cmp_si(mp_modulus,0)==0) break;
                mpz_add_ui (mp_c, mp_c, 1);
        }

        printf("\t  ");
       mpz_out_str(stdout,10,mp_number);
       if (mpz_cmp(mp_number,mp_c)==0){
           printf("\t PRIME! \n");
       }
       else printf("\t not prime \n");
       mpz_add_ui (mp_number, mp_number, 1);
   }
return 0;
}
```

```
2147483645      not prime
2147483646      not prime
2147483647      PRIME!
2147483648      not prime
2147483649      not prime
2147483650      not prime
2147483651      not prime
2147483652      not prime
2147483653      not prime
2147483654      not prime
2147483655      not prime
2147483656      not prime
2147483657      not prime
2147483658      not prime
2147483659      PRIME!
2147483660      not prime
```

Πως μπορώ να βρω *πρώτους αριθμούς με συγκεκριμένο μήκος bit*;

3ο Παράδειγμα με τη GMP – RSA v1

Ο πηγαίος κώδικας του παραδείγματος βρίσκεται στο eclass: [lecture22] **GMP-LARGE-PRIME-NUMBERS**

Προσπαθήστε να γράψετε πρόγραμμα το οποίο να βρίσκει δυο μεγάλους πρώτους αριθμούς (p , q) αυθαίρετου μήκους bits καθώς και το γινόμενο N=p*q .

```
printf("Enter RSA Encryption bits: ");
scanf("%d",&prime_bits);
gmp_randinit(stat, GMP_RAND_ALG_LC, 120);
mpz_set_ui(seed, (unsigned) time(NULL)%getpid());
gmp_randseed(stat, seed);

mpz_urandomb(p, stat, prime_bits);
primetest = mpz_probab_prime_p(p, 10);

 if (primetest != 0) {
    printf("p is prime\n");
} else {
    mpz_nextprime(p, p);
}
gmp_printf("p = %Zd\n",p);
printf("\n");
```

```
mpz_urandomb(q, stat, prime_bits);
primetest = mpz_probab_prime_p(q, 10);

 if (primetest != 0) {
    printf("q is prime\n");
} else {
    mpz_nextprime(q, q);
}
gmp_printf("q = %Zd\n",q);
printf("\n");

mpz_mul(N,p,q);
gmp_printf("N = %Zd\n",N);
printf("\n");
gmp_randclear(stat);
return 0;
```

Ο πηγαίος κώδικας των σημερινών ασκήσεων υπάρχει στο eclass [lecture 22]

3ο Παράδειγμα με τη GMP – RSA v1

Ο πηγαίος κώδικας του παραδείγματος βρίσκεται στο eclass: [lecture22] **GMP-LARGE-PRIME-NUMBERS**

Προσπαθήστε να γράψετε πρόγραμμα το οποίο να βρίσκει δυο μεγάλους πρώτους αριθμούς (p , q) αυθαίρετου μήκους bits καθώς και το γινόμενο N=p*q .

- Ο N είναι **ο συντελεστής αναδίπλωσης (modulo)** και για το δημόσιο και για το ιδιωτικό κλειδί

- Το μήκος του N σε **bits** ονομάζεται **μήκος κλειδιού** (key length)

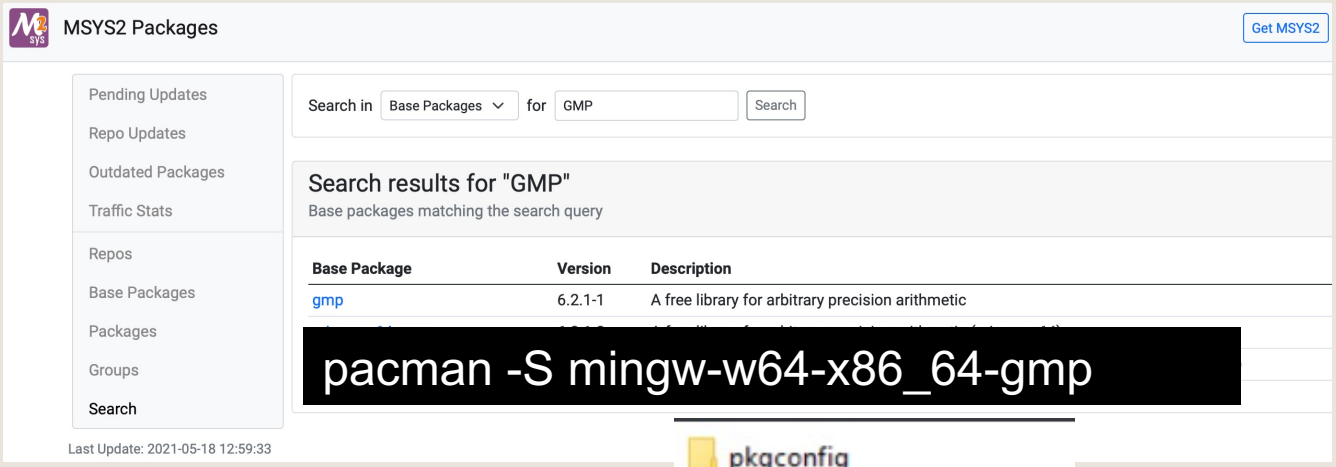
- Ο αριθμός N **δημοσιεύεται ως μέρος** του δημοσίου κλειδιού

```
RSA - ENCRYPTION
RSA - Calculate p,q, N
Enter RSA Encryption bits: 1024
p = 561576059790923652897206600000603730229600048207394681173581044563934915972761324100318915865432579823084491517789662521211872105495187144474661044446180316
63213955355002147328639982492353123173666234586426025810773526056164938007332437
80336280165492075582134609890161978662946605971284820495930753319181506 3

q = 681506758444778544084406877359958925526221910805826900936414376312235286216060143634955668904399907853023564555679351139307406538619083188474354778344958344
52242037146695574679416417562081807834641435666682252828710430742169862120737865
44416499471404902812091296499281839869898679626020810129267481668856716 3

N = 382717880128303518777957425580456633017015691972733479352566662218178900316555750049214871389120656590128524861771830136500918398365332954145564854723050274
43733058617774798330407402406979530375720369879347345139510142527739426254476660
74933522284975942097977846165451785592459310280748395254831774600487819301773199
14317339797679408097598111907614121670913041726264551490736529643956129898562619
36286898740787292960799611700832619898293296898354497130324252652417242456845367
21874856444225069546571384449502284200857042852300652148607260179336703526252347
91125627624090106687937196140491718922664784396984075057626 9
```

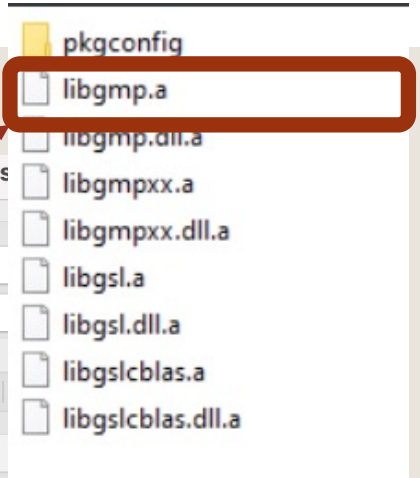
Ο πηγαίος κώδικας των σημερινών ασκήσεων υπάρχει στο eclass [lecture 22]

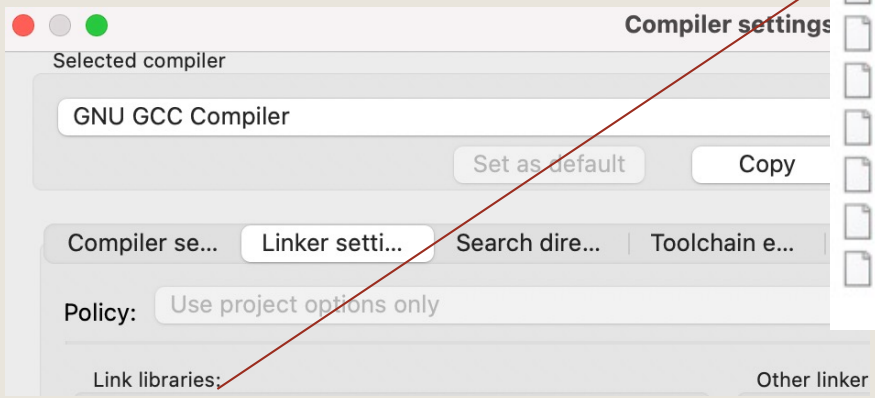
Οδηγίες Εγκατάστασης της GMP

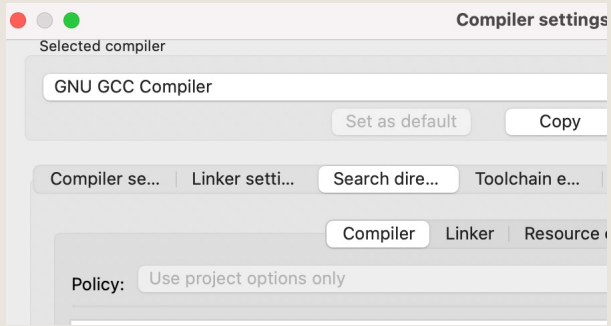
- **Windows: MSYS2 (δείτε και τις οδηγίες στο eclass::Lecture 20)**



pacman -S mingw-w64-x86_64-gmp

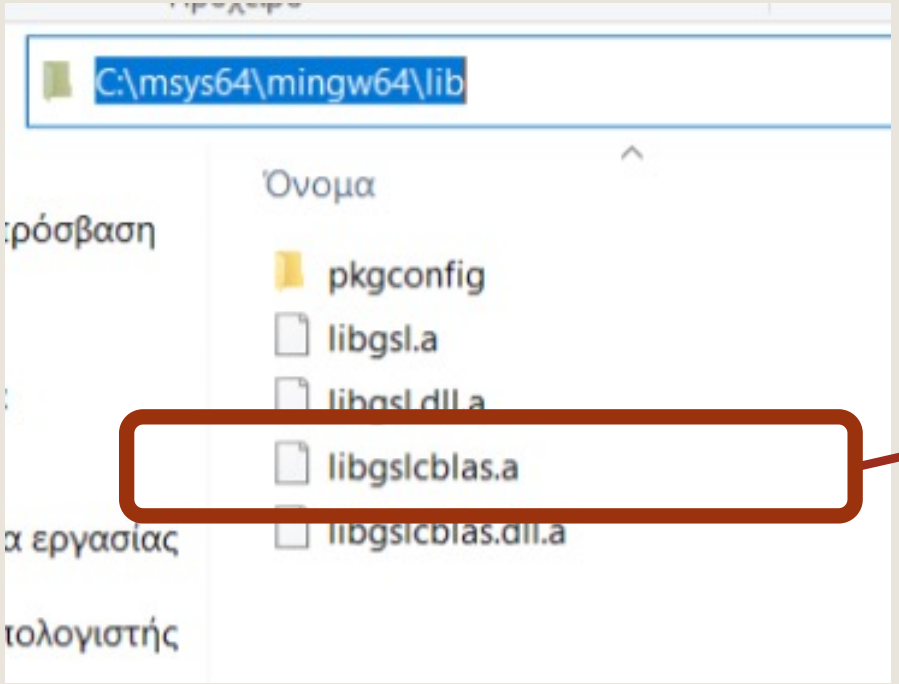


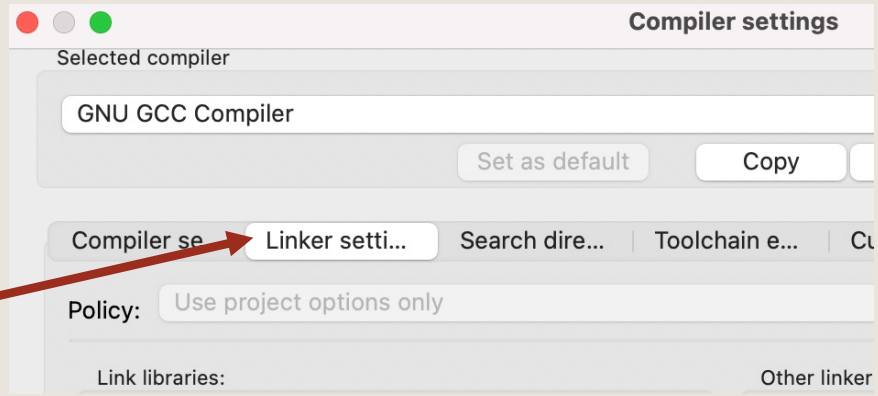




Σχετικά με τη 2η Δραστηριότητα με τη Βιβλιοθήκη GSL

- Ενημερώστε τον Linker για τη βιβλιοθήκη **libgslcblas.a**





Ευχαριστώ για την προσοχή σας

- **Επικοινωνία**
  - Skype: fidas.christos
  - Email: fidas@upatras.gr
  - Phone: 2610 – 996491
  - Web: http://cfidas.info

Το υλικό της διάλεξης είναι διαθέσιμο στο eclass