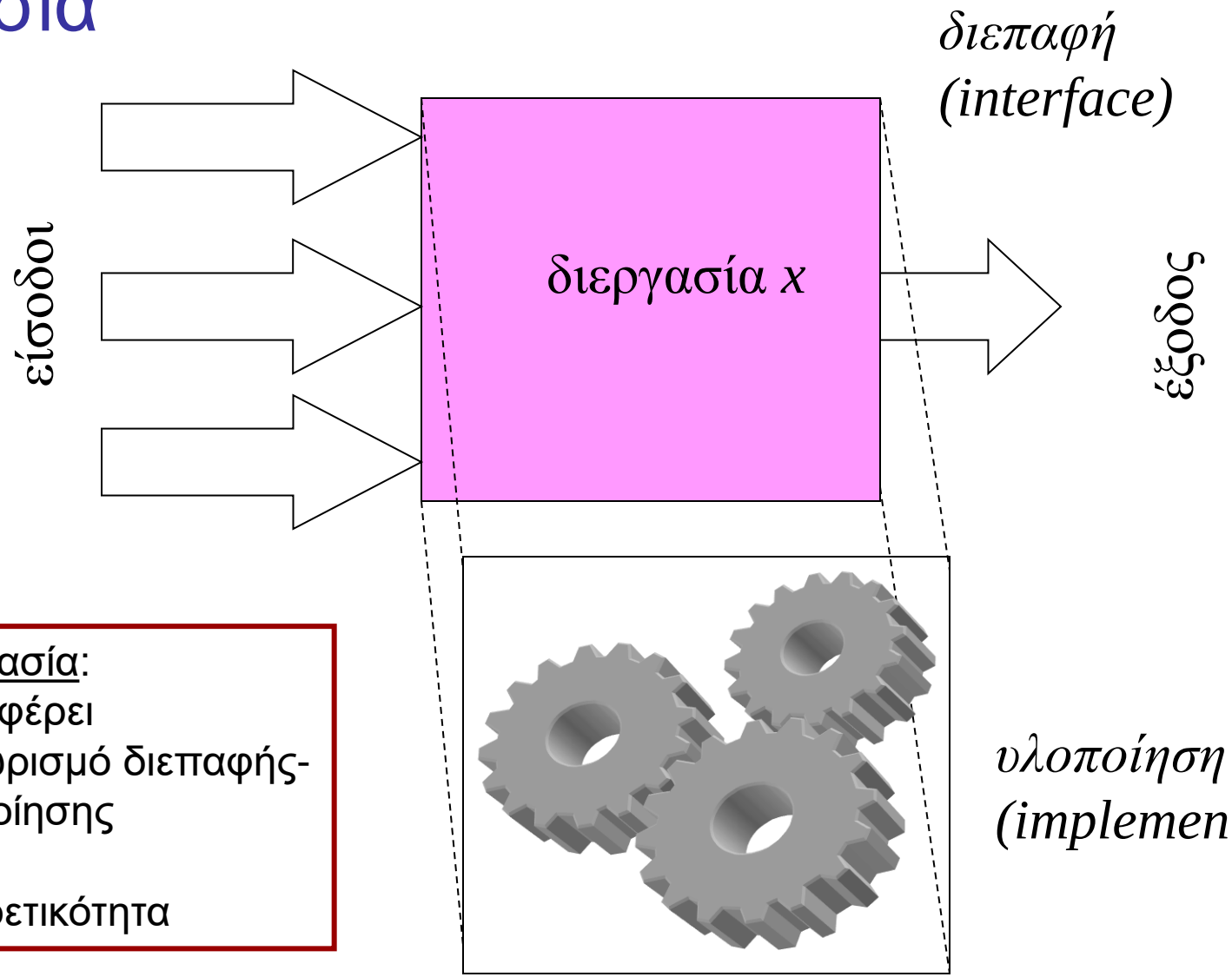
Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

Διαδικαστικός προγραμματισμός: Η έννοια της διεργασίας

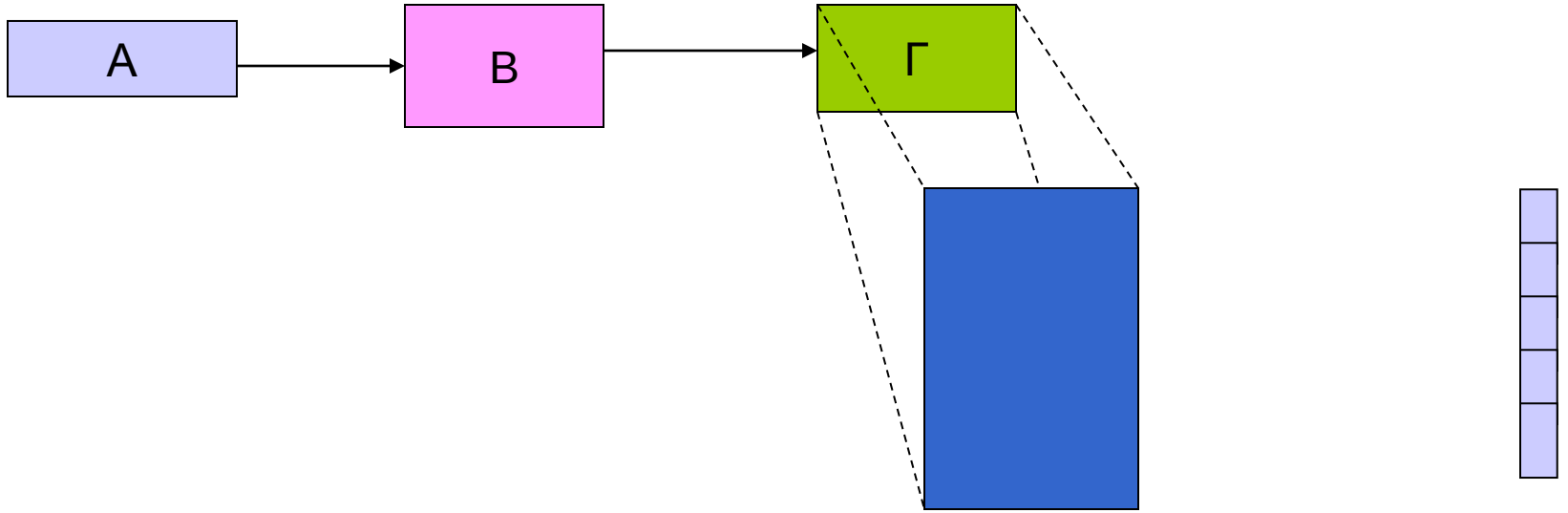
- top down και bottom up
- Τμήμα μιας ευρύτερης λύσης, με σαφώς καθορισμένη σχέση εισόδων εξόδων
  - όταν είναι δυνατόν να εκτελεστεί από υπολογιστή ⇒ *υπολογιστική διεργασία*
- Βασικό εργαλείο στο χειρισμό της πολυπλοκότητας ⇒ αφαιρετικότητα

Διεργασία



Διεργασία:
προσφέρει διαχωρισμό διεπαφής-υλοποίησης
⇒
αφαιρετικότητα

Συναρτήσεις και διεργασίες - Οργάνωση λύσης

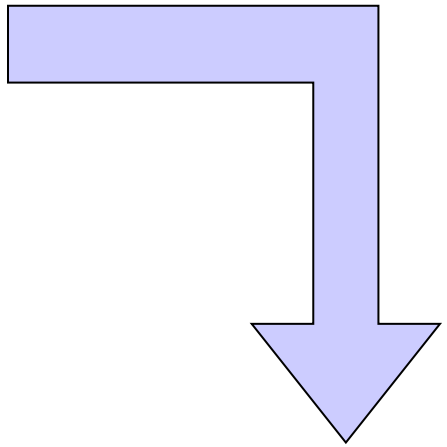


Στη γλώσσα C

συναρτήσεις ↔ υπολογιστικές διεργασίες

Απεικόνιση σε οδηγίες προς τον υπολογιστη: «προστακτικός» προγραμματισμός

- Εκτέλεσε τη διεργασία Α
- Εκτέλεσε τη διεργασία Β
- Εκτέλεσε τη διεργασία Γ
- Εκτέλεσε τη διεργασία Δ



Πώς γνωρίζει το σύστημα τι πρέπει
να κάνει για να εκτελέσει τη διεργασία Α;

Διεργασία Α:
  Εκτέλεσε τη διεργασία Α1
  Εκτέλεσε τη διεργασία Α2

Στα πλαίσια του μαθήματος

- Τεχνικές για την ανάπτυξη προγραμμάτων
  - Διαδικαστικό στυλ προγραμματισμού
  - «Σωστά προγράμματα», «καλύτερα προγράμματα», «ποιότητα κώδικα»,…
  - «Δουλεύει» άρα «εντάξει»

- Γλώσσα C
  - C17 ISO/IEC 9899:2018
  - C11 ISO/IEC 9899:2011
  - C99 ISO/IEC 9899:1999
  - C90 ISO/IEC 9899:1990

Η φιλοσοφία της C (όχι τυπικά)

- Θεωρεί ότι ο προγραμματιστής ξέρει τι κάνει => εμπιστοσύνη
- Υποστηρίζει κατάλληλα επίπεδα αφαίρεσης αλλά δεν κρύβει το υλικό (hardware) που εκτελεί τα προγράμματα.
- Ο προγραμματιστής έχει πρόσβαση στο υλικό όταν είναι αναγκαίο.
- Υποστηρίζεται η ανάπτυξη κώδικα ευκολονόητου και απλού στη συντήρηση.
- Παρέχει ταχύτητα χωρίς πάρα πολύ προσπάθεια.
- Για προγραμματισμό σε χαμηλό επίπεδο μπορείς να γράψεις assembly συμβατή με C, αλλά δεν το χρειάζεσαι πολύ συχνά.
- Για κατασκευή compilers (μεταγλωττιστών): από καλογραμμένο κώδικα C μπορεί να προκύψει assembly υψηλής ποιότητας
- Για τον προγραμματιστή, η κατανόηση της οργάνωσης του υπολογιστή παραμένει σημαντική.

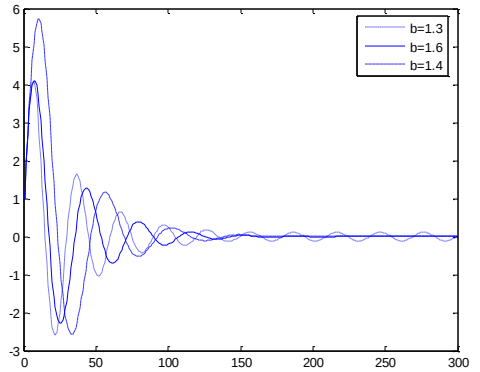
Μερικές διαφορές python και C

- Η C είναι ***απλούστερη*** γλώσσα
  - Ελάχιστα keywords
  - Βασικές δομές διαθέσιμες, οι λοιπές κατασκευάζονται
  - Διαχείριση μνήμης από τον προγραμματιστή
  - Κοντά στο υλικό
- **Στατικό** σύστημα τύπων αντί δυναμικού
  - **Τύποι** διαθέσιμοι ή κατασκευασμένοι
  - C: Έλεγχος τύπων γίνεται την ώρα της **μεταγλώττισης**, όχι την ώρα της εκτέλεσης του προγράμματος.
  - Κάθε μεταβλητή έχει καθορισμένο τύπο.
- Χρήση δεικτών (pointers)

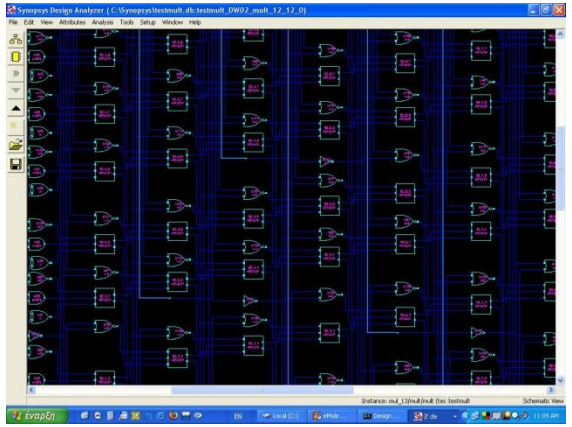
C και Python –
Οργάνωση προγράμματος

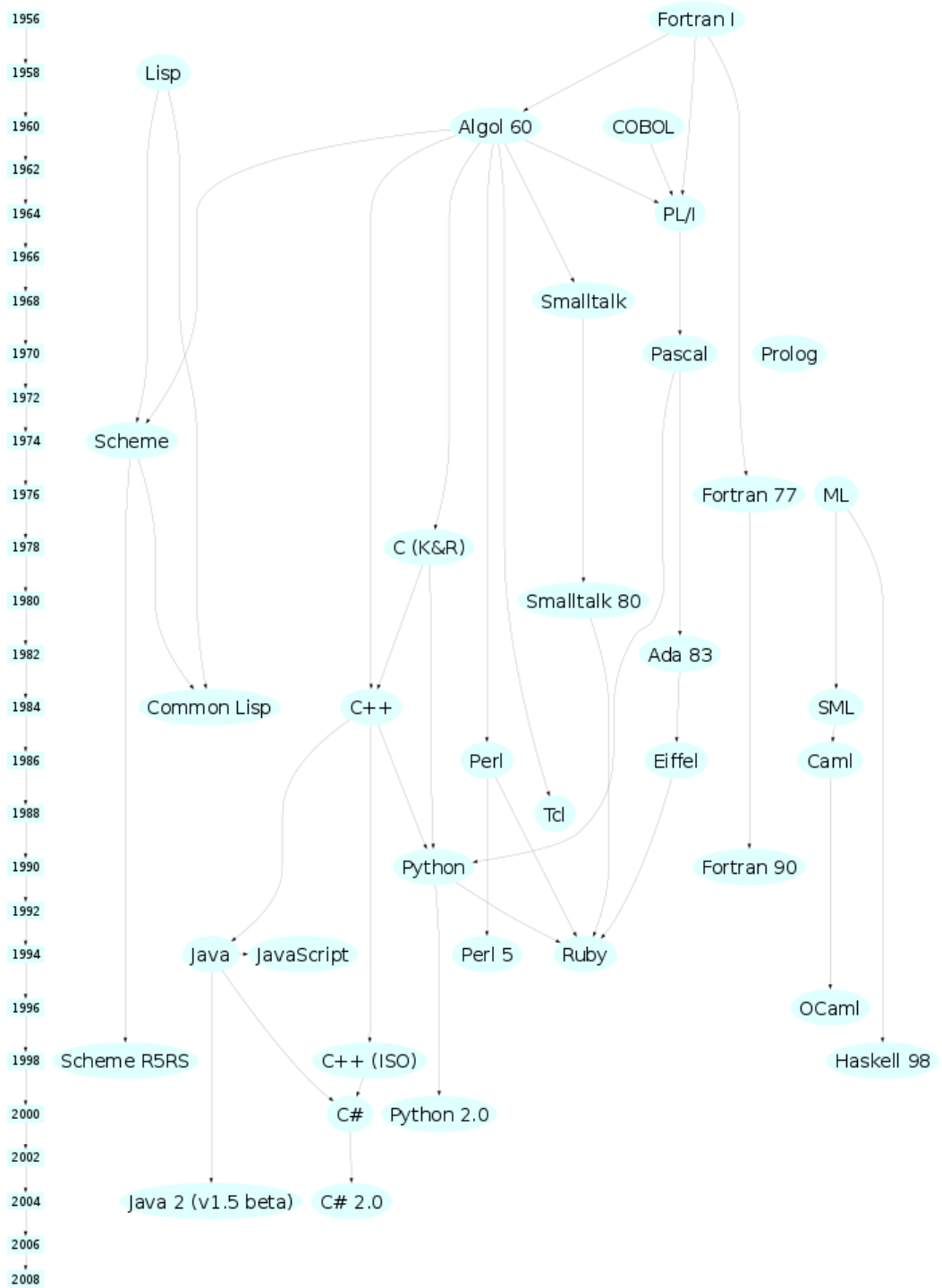
- C: Κώδικας ***μόνο*** σε συναρτήσεις
  - Επιτρέπονται μόνο ***δηλώσεις*** εκτός συναρτήσεων
- C: Υποχρεωτικά δηλώσεις **ονομάτων**
  - Τύπων, μεταβλητών, …
  - Κατάλληλες δηλώσεις αξιοποιούν το **type checking**
- C: Μπλοκ κώδικα με { } αντί στοίχιση ανά γραμμή
- C: Προτάσεις ολοκληρώνονται με **;** αντί τέλους γραμμής
- Θα δούμε και άλλες διαφορές.

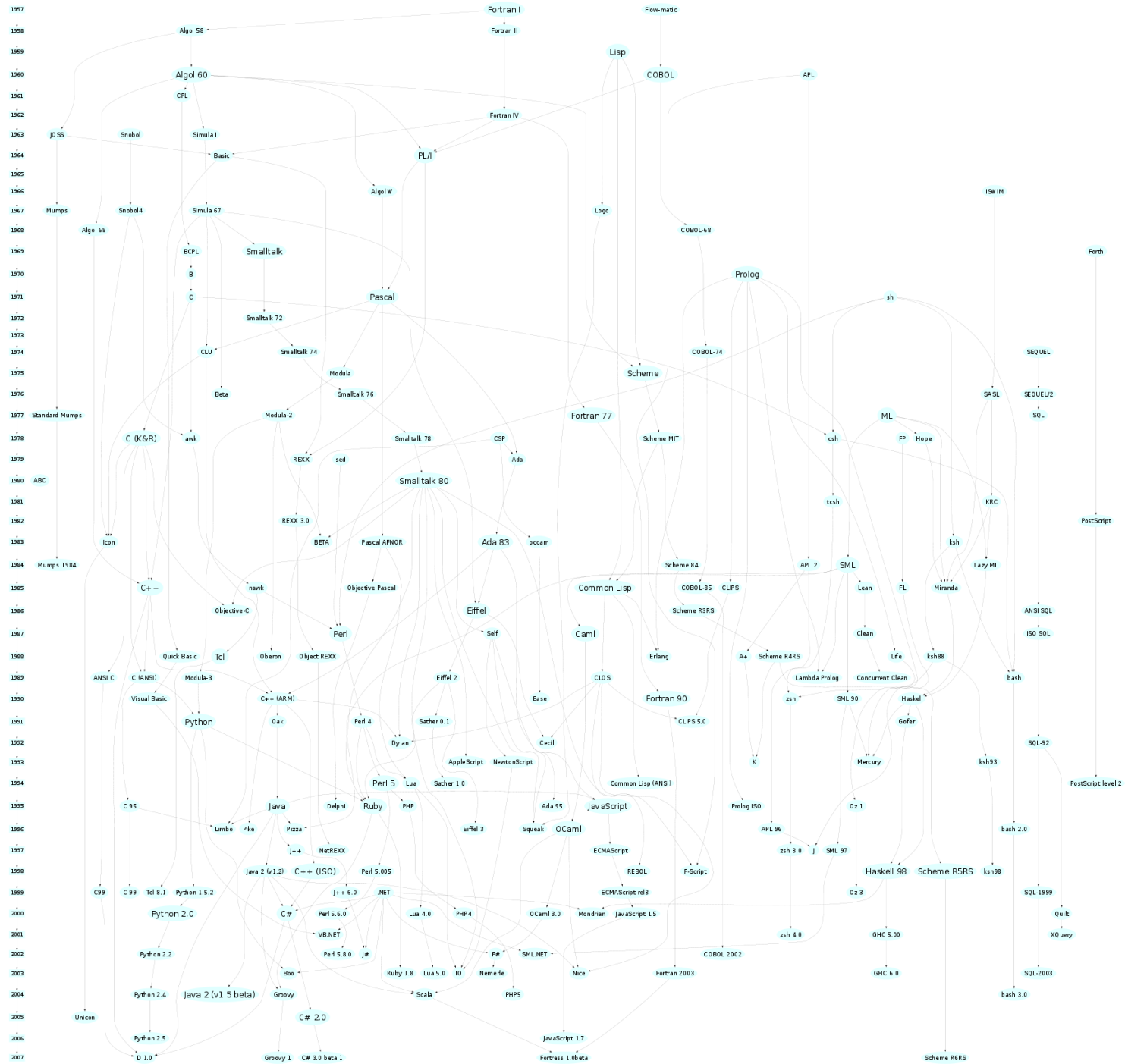
Καθημερινότητα...



- Προδιαγραφές και εξομοιώσεις
  σε C/C++, matlab, SystemC,...
- Περιγραφή υλικού σε VHDL, verilog,...
  - γράφουμε απευθείας μοντέλα ή μέσω scripts (perl,...) αλλά και matlab ή C, mathematica
  - παράγουμε οδηγίες προς τον εξομοιωτή με script
- Επεξεργασία μοντέλων
  με EDA CAD,
  - οδηγίες με γλώσσα
    χαρακτηριστική του εργαλείου
- κτλ.




1956
1958
1960
1962
1964
1966
1968
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
Fortran I
Lisp
Algol 60
COBOL
PL/I
Smalltalk
Pascal
Prolog
Scheme
Fortran 77
ML
C (K&R)
Smalltalk 80
Ada 83
Common Lisp
C++
SML
Perl
Eiffel
Caml
Tcl
Python
Fortran 90
Java
JavaScript
Perl 5
Ruby
OCaml
Scheme R5RS
C++ (ISO)
Haskell 98
C#
Python 2.0
Java 2 (v1.5 beta)
C# 2.0


1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
Fortran I
Flow-matic
Algol 58
Fortran II
Lisp
Algol 60
COBOL
APL
CPL
Fortran IV
JOSS
Snobol
Simula I
Basic
PL/I
Algol W
ISWIM
Mumps
Snobol4
Simula 67
Logo
Algol 68
COBOL-68
BCPL
Smalltalk
Forth
B
Prolog
C
Pascal
sh
Smalltalk 72
CLU
Smalltalk 74
COBOL-74
SEQUEL
Modula
Scheme
Beta
Smalltalk 76
SASL
SEQUEL/2
Standard Mumps
Modula-2
Fortran 77
ML
SQL
C (K&R)
awk
Smalltalk 78
CSP
Scheme MIT
csh
FP
Hope
REXX
sed
Ada
ABC
Smalltalk 80
tcsh
KRC
REXX 3.0
PostScript
Icon
BETA
Pascal AFNOR
Ada 83
occam
ksh
Mumps 1984
Scheme 84
APL 2
SML
Lazy ML
C++
nawk
Objective Pascal
Common Lisp
COBOL-85
CLIPS
Lean
FL
Miranda
Objective-C
Scheme R3RS
ANSI SQL
Eiffel
Perl
Self
Caml
Clean
ISO SQL
Quick Basic
Tcl
Oberon
Object REXX
Erlang
A+
Scheme R4RS
Life
ksh88
ANSI C
C (ANSI)
Modula-3
Eiffel 2
CLOS
Lambda Prolog
Concurrent Clean
bash
Visual Basic
C++ (ARM)
Ease
Fortran 90
zsh
SML 90
Haskell
Python
Oak
Perl 4
Sather 0.1
CLIPS 5.0
Gofer
SQL-92
Dylan
Cecil
AppleScript
NewtonScript
K
Mercury
ksh93
Perl 5
Lua
Sather 1.0
Common Lisp (ANSI)
PostScript level 2
C 95
Java
Delphi
Ruby
PHP
Ada 95
JavaScript
Prolog ISO
Oz 1
Limbo
Pike
Pizza
Eiffel 3
Squeak
OCaml
APL 96
bash 2.0
J++
NetREXX
ECMAScript
zsh 3.0
J
SML 97
Java 2 (v1.2)
C++ (ISO)
Perl 5.005
REBOL
F-Script
Haskell 98
Scheme R5RS
ksh98
C99
C 99
Tcl 8.1
Python 1.5.2
J++ 6.0
.NET
ECMAScript rel3
Oz 3
SQL-1999
Python 2.0
C#
Perl 5.6.0
Lua 4.0
PHP4
OCaml 3.0
Mondrian
JavaScript 1.5
Quilt
VB.NET
zsh 4.0
GHC 5.00
XQuery
Python 2.2
Perl 5.8.0
J#
F#
SML.NET
COBOL 2002
Boo
Ruby 1.8
Lua 5.0
IO
Nemerle
Nice
Fortran 2003
GHC 6.0
SQL-2003
Python 2.4
Java 2 (v1.5 beta)
Groovy
Scala
PHP5
bash 3.0
Unicon
C# 2.0
Python 2.5
JavaScript 1.7
D 1.0
Groovy 1
C# 3.0 beta 1
Fortress 1.0beta
Scheme R6RS

Γιατί C;

- Υποστηρίζεται από πάρα πολλές πλατφόρμες
- Είναι φορητή
- Στενή σχέση με το unix
  - Κυκλοφορεί αρκετό καιρό
  - Πάντα υποστηρίζεται σε κάθε Unix και παραλλαγές (linux)
- Είναι απλή.
- Είναι ευέλικτη
- Είναι γρήγορη
- Είναι δωρεάν
- Μεγάλη εγκαταστημένη βάση.
  - Δημοφιλής για δεκαετίες
- Ώριμη: Δεν αναμένονται μεγάλες αλλαγές

Πηγή: http://programmers.stackexchange.com/

Περιβάλλοντα προγραμματισμού

- dev-c++ (http://www.bloodshed.net)
- Orwell Dev-C++ (http://orwelldevcpp.blogspot.gr/)
- gcc σε Cygwin ή linux
- Eclipse IDE for C/C++ developers
- Visual Studio
- Code::Blocks https://www.codeblocks.org/
- …

Εργαστήριο

- Πέντε εργαστηριακές ασκήσεις
- Εργαστήριο κάθε δεύτερη εβδομάδα γενικά.
- Code::Blocks

Θεμελιώδης διαφορά από φέτος

- Θα πρέπει να ολοκληρώσουμε επιτυχώς το εργαστήριο για να προσέλθουμε στην εξέταση
- Επιτυχής ολοκλήρωση
  - Μέσος όρος βαθμών εργαστηριακών ασκήσεων >=5
  - Παρουσίες στο εργαστήριο.

Δώστε έμφαση στο εργαστήριο!

- Προσοχή στη διαχείριση του χρόνου
- Εξοικειωνόμαστε με το προγραμματιστικό περιβάλλον
- Μελέτη!
- Λύνουμε απλές ασκήσεις, δοκιμάζουμε τα παραδείγματα
- Χρησιμοποιούμε εργαλεία όπως cppcheck και προτεινόμενα compiler options
- Επιδιώκουμε να καταλάβουμε γιατί δεν δουλεύει μια λύση

Ένα πρόγραμμα σε C

Πάντα υπάρχει μια και μόνο μια συνάρτηση `main`

```
#include <stdio.h>

int main (void) {

  printf ("Hello world!\n");

  return 0;
}
```

Ο κώδικας οργανώνεται με άγκιστρα

Γράφουμε κώδικα μόνο μέσα σε συναρτήσεις!!!

```c
#include <stdio.h>
int main() {
   int i, j;
   int sum ;

   i = 5 ;
   j = 6 ;
   sum = sum + j ;

   printf ("sum: %d\n", sum);

   return 0;
}
```

Δηλώνουμε τα ονόματα πριν τα χρησιμοποιήσουμε !!!

Διαφορετικές γλώσσες – διαφορετική συμπεριφορά **C** *vs.* **Python**

```c
#include <stdio.h>
int main(void) {
 int i = 1<<30;
 printf("%d\n", i);
 system("pause");
}
```

Σημαίνει 2^{30}

1073741824

```
>>> a = 1 <<30
>>> a
1073741824
```

```c
#include <stdio.h>
int main(void) {
 int i = 1<<31;
 printf("%d\n", i);
 system("pause");
}
```

-2147483648

```
>>> a = 1<<31
>>> a
```

Γιατί;

Πόσο κοστίζει;

2147483648

Ένα **πείραμα σε Python**

```
global a
a = 1<<30
def test():
    for i in range(0, 50):
        b = a + 1
if __name__ == '__main__':
        from timeit import Timer
        t = Timer('test()','from __main__ import test')
        print (t.timeit(1000000))
```

Χρόνος: 10.7764198383 sec

```
global a
a = 1<<31
def test():
    for i in range(0, 50):
        b = a + 1
if __name__ == '__main__':
    from timeit import Timer
    t = Timer('test()','from __main__ import test')
    print (t.timeit(1000000))
```

Χρόνος: 21.0824803152 sec

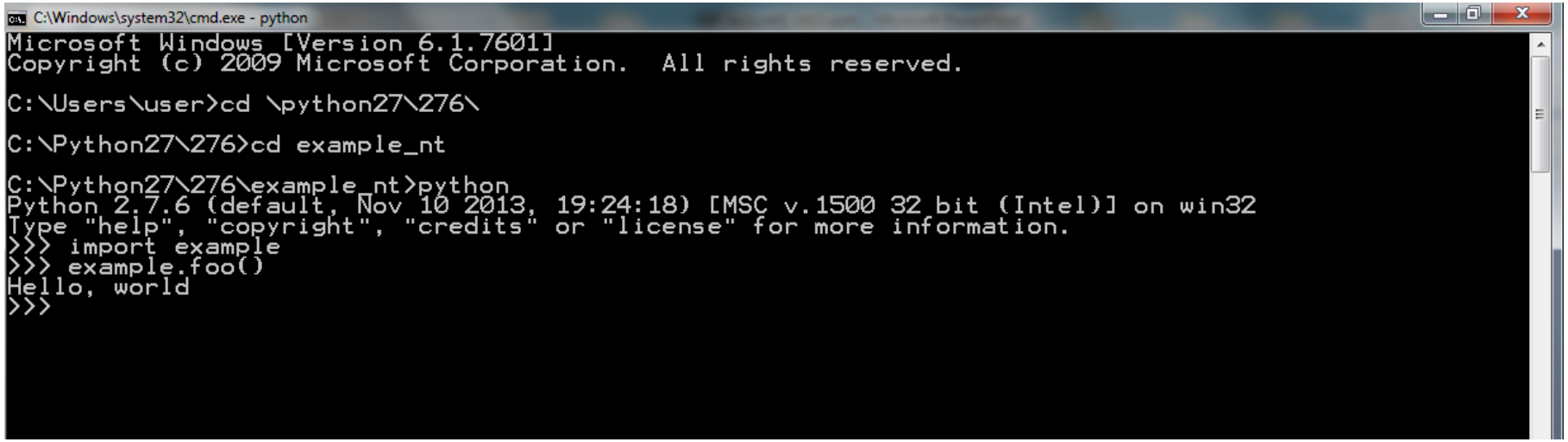
Διπλάσιος χρόνος!

```c
#include <stdio.h>
#include <time.h>
int a = 1<<31;

int test() {
 int j;
 int b ;
 for (j=0;j<=50;j++)
        b = a + 1;
}
int main( ) {
 int j;
 clock_t t1, t2;
 float ratio ;
 ratio = 1./CLOCKS_PER_SEC;
 t1 = clock();
 for (j=0;j<1000000;j++)
     test();
 t2 = clock();
printf("Time = %f\n", ratio*(long)t1 +
   ratio*(long)t2 );
}
```

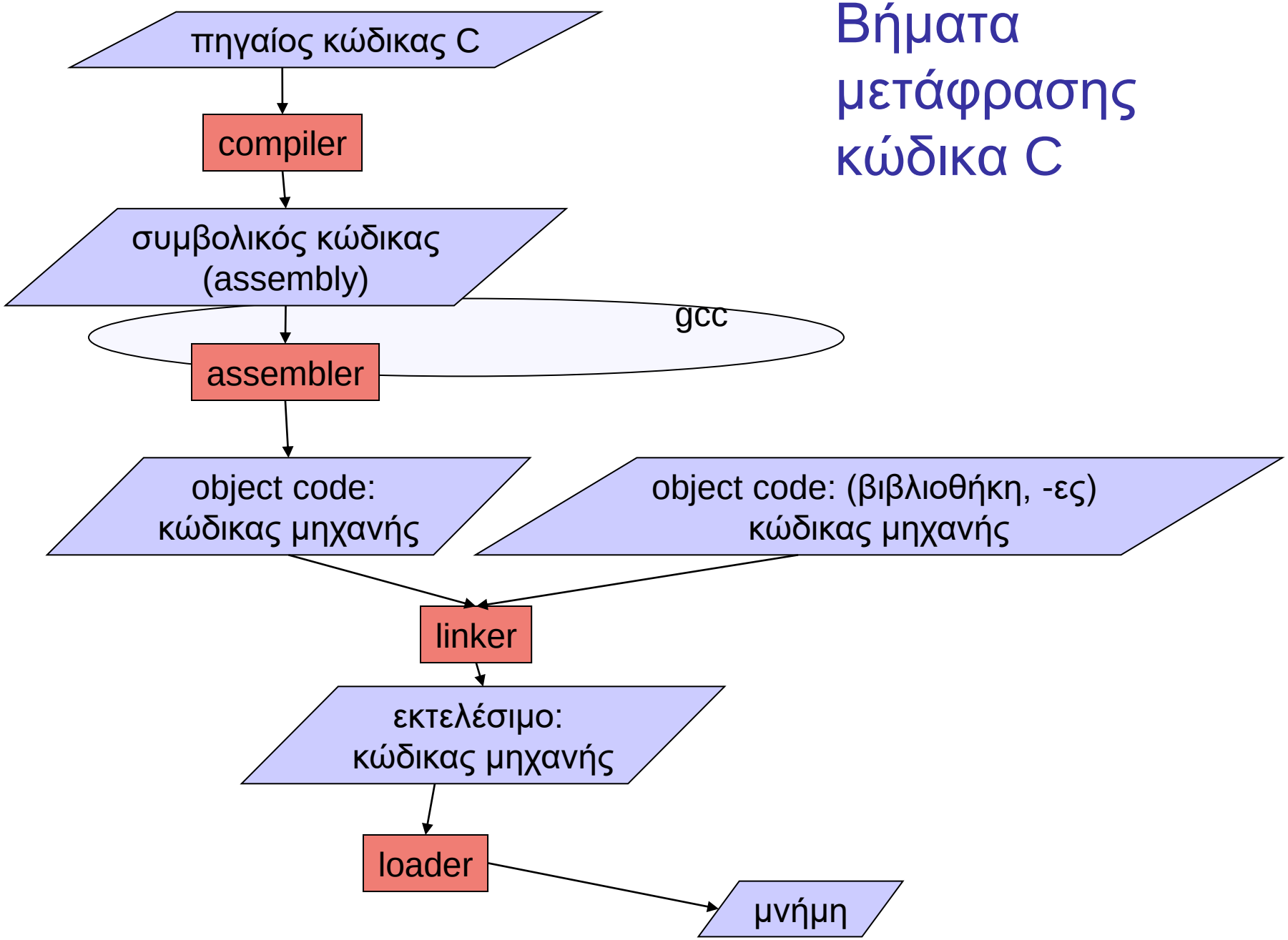
Σε C

Χρόνος: 0.46 sec

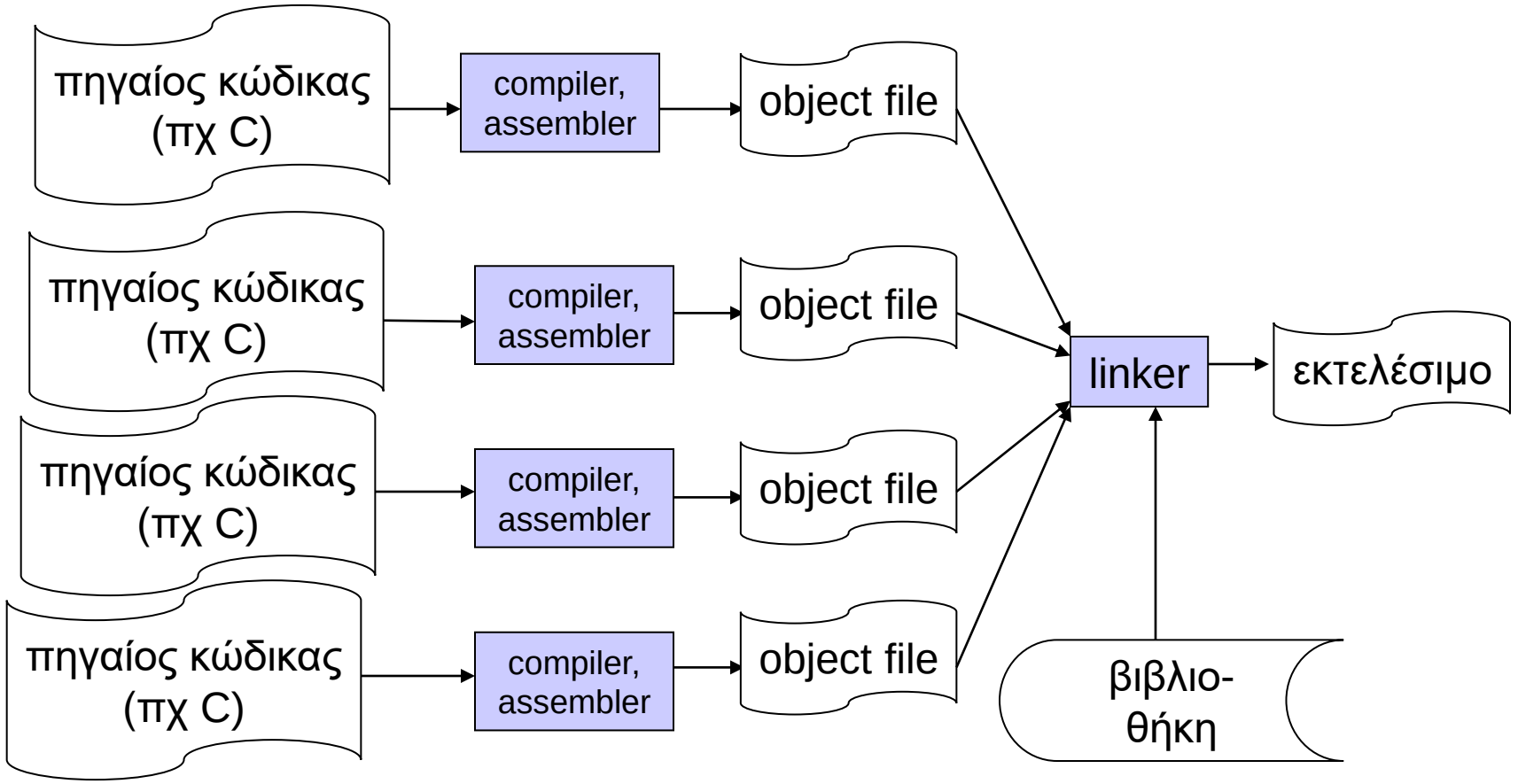
Επεκτάσεις C της python



Βήματα μετάφρασης κώδικα C



Από γλώσσα υψηλού επιπέδου σε εκτελέσιμο



Η γλώσσα C ως αφαίρεση

- Οι γλώσσες υψηλού επιπέδου είναι μια μορφή αφαίρεσης.
  - Η πολυπλοκότητα του **κώδικα μηχανής** αποκρύπτεται από τον προγραμματιστή.
- Ο μεταγλωττιστής (compiler) αντιστοιχίζει στον κώδικα C σύνολα εντολών κώδικα μηχανής

Βρόχος **for** σε C

```c
#include <stdio.h>

int main (void) {

    register int i, sum = 0;

    for ( i = 0; i < 10; i++) {
        sum =  sum + i;
      }
    printf ("\ti: %d\n", sum);

    return 0;
}
```

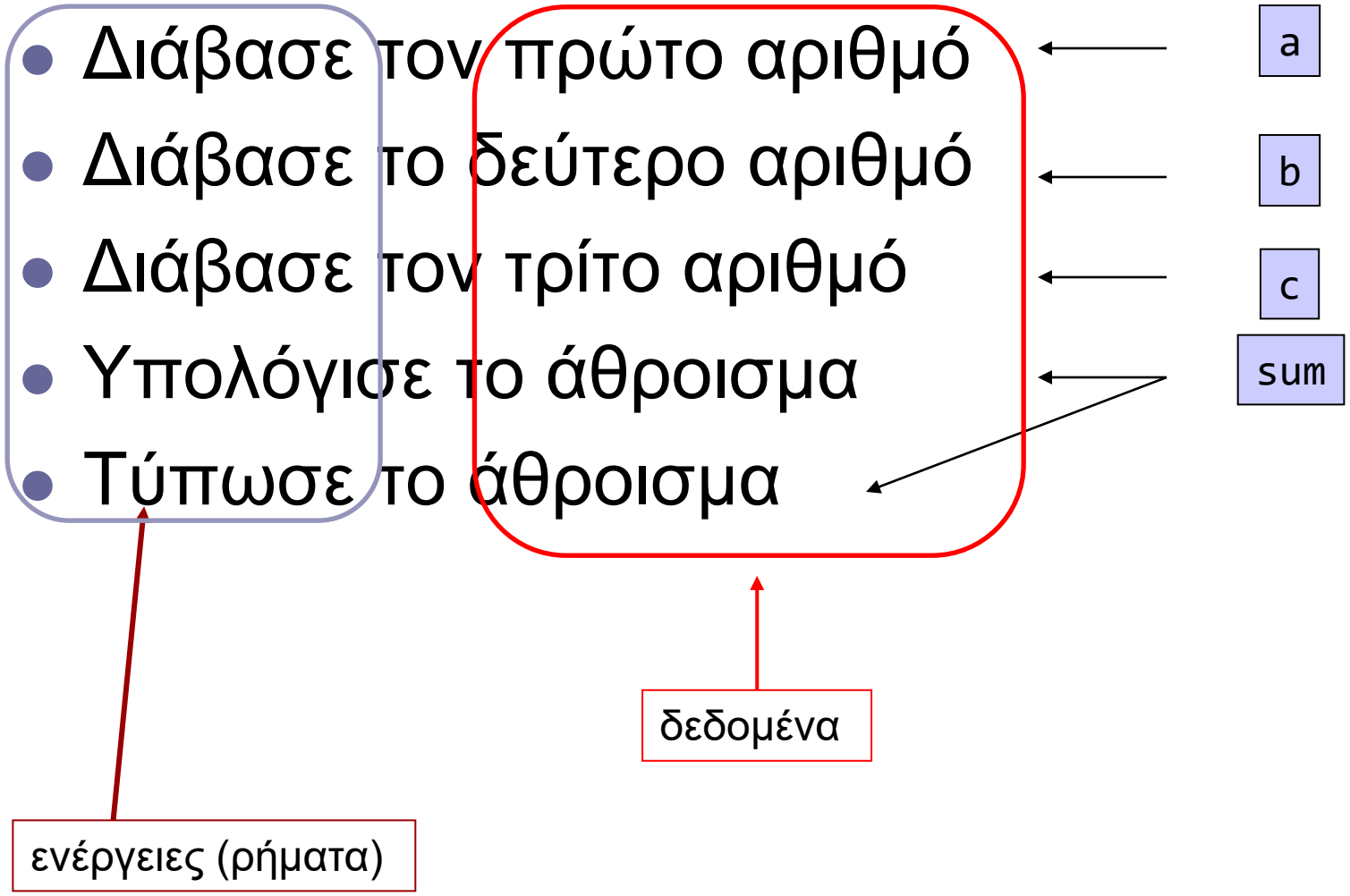
τι κάνει;

Αξιοποίηση αφαιρετικότητας με διεργασίες

```c
#include <stdio.h>
int computeGCD(int, int);

int main ( void ) {
 int a, b;
 int gcd;

 scanf ("%d %d", &a, &b) ;

 gcd = computeGCD (a, b);

 printf("%d\n", gcd);
}
```

```c
int computeGCD(int a , int b) {
  int result;
  int i = a, j = b;

  while (i != j) {
   if ( i > j)
        i = i - j ;
     else
        j = j - i;
   }

  return i;
}
```

Βήμα 1



Προστακτικό στυλ στην περιγραφή της λύσης

Η περιγραφή διαχωρίζεται σε ***ενέργειες*** και ***δεδομένα***

Μεταβλητές αντιστοιχίζονται στα δεδομένα

Βήμα 2

```
int a, b, c, sum;
```

- Διάβασε τον a ← `scanf("%d", &a);`
- Διάβασε το b ← `scanf("%d", &b);`
- Διάβασε τον c ← `scanf("%d”, &c);`
- Υπολόγισε το sum ← `sum = a + b + c;`
- Τύπωσε το sum ← `printf("the sum is: %d\n", sum);`

Ενέργειες (*ρήματα*) ⇒ διεργασίες ⇒ κλήσεις συναρτήσεων
Συναρτήσεις από βιβλιοθήκες ή νέες
Προτάσεις για απλές περιπτώσεις

Βήμα 3: Δεύτερο Πρόγραμμα σε C

```c
#include <stdio.h>

int main ( void ) {
   int a, b, c, sum;
   scanf("%d", &a);
   scanf("%d", &b);
   scanf("%d", &c);
   sum = a + b + c;
   printf ("sum is %d", sum);
 return 0;
}
```

τι κάνει;

Καλές επιλογές ονομάτων ⇒ Ευανάγνωστος κώδικας

```
i= 120;
if (i>j)
func1();
else
func2();
```

```
velocity = 120;
if (velocity > max_velocity)
   decrease_velocity( );
else
   increase_velocity( );
```

Μέχρι τώρα...

- Αφαιρετικότητα
  - Ως εργαλείο για την αντιμετώπιση της πολυπλοκότητας
- Σχεδιασμός Προγράμματος
- *Αναφέραμε* την Αυξητική Ανάπτυξη Προγράμματος
  - Υποστηρίζει top-down ανάπτυξη
  - Διευκολύνει την ανάπτυξη του προγράμματος
  - διευκολύνει τον έλεγχο του κώδικα

Βασική μέθοδος: Αυξητική ανάπτυξη προγράμματος (incremental development)

- Να γραφεί ένα πρόγραμμα που **διαβάζει** έναν αριθμό, να **υπολογίζει** την τρίτη δύναμή του, και στη συνέχεια να **τυπώνει** το αποτέλεσμα.

int number, power;

- Διάβασε έναν αριθμό ⟵ number
- Υπολόγισε την τρίτη δύναμή του ⟵ power
- Τύπωσε το αποτέλεσμα ⟵ power

- **int** number, power;
- Διάβασε number ⟵ readnumber()
- Υπολόγισε power ⟵ computepower()
- Τύπωσε power ⟵ printout()

Σχεδίαση top-down

```
int main( void) {
  int number, power;
  number = readnumber( );
  power = computepower(number );
  printout(power );
  return 0;
}
```

έκδοση 0: θα πρέπει να είναι εκτελέσιμο!

```c
#include <stdio.h>
int readnumber(void);
int computepower(int);
void printout(int);

int main(void) {
 int number, power;
 number = readnumber( );
 power = computepower(number);
 printout(power);

 return 0;
}
```

```c
int readnumber(void ) {
 printf ("function: diabase\n");
 return 5;
}

int computepower(int x ) {
 printf ("function: ypologise\n");
 return x;
}

void printout(int x ) {
  printf("function: typwse\n");
  return ;
}
```

έκδοση 1: πλήρης printout()

```c
#include <stdio.h>
int readnumber(void);
int computepower(int);
void printout(int);

int main(void) {
 int number, power;
 number = readnumber( );
 power = computerpower(number);
 printout(power);

 return 0;
}
```

```c
int readnumber(void) {
 printf ("function: diabase\n");
 return 5;
}

int computepower(int x ) {
 printf ("function: ypologise\n");
 return x;
}

void printout(int x ) {
 printf("function: typwse\n");
 printf("apotelesma: %d\n",x);

 return ;
}
```

έκδοση 2: πλήρης computepower()

```c
#include <stdio.h>
int readnumber(void);
int computepower(int);
void printout(int);

int main(void) {
 int number, power;
 number = readnumber( );
 power = computepower(number);
 printout(power);
 return 0;
}
```

```c
int readnumber (void) {
 printf ("function: diabase\n");
 return 5;
}

int computepower(int x ) {
 printf ("function: ypologise\n");
 return x * x * x;
}

void printout (int x ) {
 printf("function: typwse\n");
 printf("apotelesma: %d\n",x);
 return ;
}
```

Έκδοση 3: πλήρης printout()

```c
#include <stdio.h>
int readnumber(void);
int computepower(int);
void printout(int);

int main(void) {
 int number, power;
 number = readnumber( );
 power = computepower(number);
 printout(power);

 return 0;
}
```

```c
int readnumber (void) {
 int aninput;
 printf ("function: diabase\n");
 scanf("%d", &aninput);
 return aninput;
}

int computepower(int x ) {
 printf ("function: ypologise\n");
 return x * x * x;
}

void printout(int x ) {
 printf("function: typwse\n");
 printf("apotelesma: %d\n",x);
 return ;
}
```

Μέχρι τώρα...

- Οργάνωση Προγράμματος C
  - Ενέργειες -> ρήματα (συντακτικό) -> συναρτήσεις
  - Δεδομένα -> αντικείμενα(συντακτικό) -> μεταβλητές
- Αφαιρετικότητα (abstraction)
  - εργαλείο για την αντιμετώπιση της πολυπλοκότητας
  - Διαχωρίζουμε το
  - ***τι γίνεται*** -> *όνομα, δήλωση συνάρτησης*
  - *από το*
  - ***Πώς γίνεται -> υλοποίηση συνάρτησης,***
  - Χρήση σύνθετων τύπων
- Αυξητική Ανάπτυξη Προγράμματος
  - top-down ανάπτυξη
  - Διευκολύνει *ανάπτυξη* του προγράμματος
  - Διευκολύνει τον *έλεγχο* του προγράμματος
  - Ξεκινάμε γράφοντας κάτι εκτελέσιμο
  - C: για να είναι εκτελέσιμο, πρέπει να έχει main() διαφορά από Pytthon.
- Διαδικασία compile και link για δημιουργία εκτελέσιμου.
  - Γίνονται με gcc/mingw

Μερικές καλές πρακτικές που είδαμε μέχρι τώρα

- Πρώτα **σκέφτομαι** μετά **γράφω** κώδικα.
- Γράφω με **μέθοδο**, ώστε να είναι αμέσως εκτελέσιμος ο κώδικας.
- Γράφω **τμήματα κώδικα** και τα **δοκιμάζω**.
  - Όταν **σχεδιάζω** την υλοποίηση, θα πρέπει να με απασχολεί πώς θα κάνω τις δοκιμές.

Στοιχεία της Γλώσσας C

- Γραμματική και Συντακτικό
- Διαθέσιμοι **τύποι δεδομένων**
  - Απλοί και σύνθετοι τύποι
- Βασική βιβλιοθήκη της C
  - παρέχει ένα σύνολο έτοιμων συναρτήσεων: printf( ), scanf( ), …
- Εκτεταμένη τεκμηρίωση της GNU C library
  - http://www.gnu.org/software/libc/manual/

Δεσμευμένες λέξεις (reserved words)

- Λέξεις κλειδιά (keywords)
- Ονόματα συναρτήσεων της βασικής βιβλιοθήκης
- Ονόματα μακροεντολών που ορίζονται σε αρχεία επικεφαλίδας: EOF, …
- Ονόματα τύπων που ορίζει η βασική βιβλιοθήκη: time_t, …
- Ονόματα εντολών προεπεξεργαστή: include, define
- Ονόματα της μορφής _DATE_, _FILE_, κτλ.

Αναγνωριστές (Identifiers)

- λέξεις που κατασκευάζει ο προγραμματιστής για να ονομάσει
  - μεταβλητές
  - σταθερές
  - συναρτήσεις
  - ...
- Δεν θα πρέπει να είναι **δεσμευμένες**

Τύποι Δεδομένων στη C

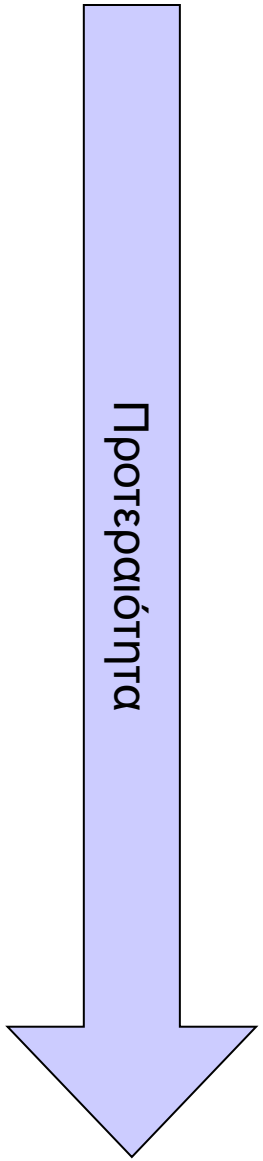
- **`char`** – χαρακτήρας
- **`int`** – ακέραιος
- **`float`** – αριθμός κινητής υποδιαστολής απλής ακρίβειας
- **`double`** – αριθμός κινητής υποδιαστολής διπλής ακρίβειας
- Απαριθμητικός τύπος
  - `enum boolean {FALSE, TRUE};`
- Σύνθετοι τύποι
  - πίνακες και δομές (**`struct`**)
- Τύποι του C90. Το C99 επεκτείνει με header files
  - `#include <stdbool.h>`
  - `#include <stdint.h>`
  - `#include <inttypes.h>`

Ομαδοποίηση Τελεστών

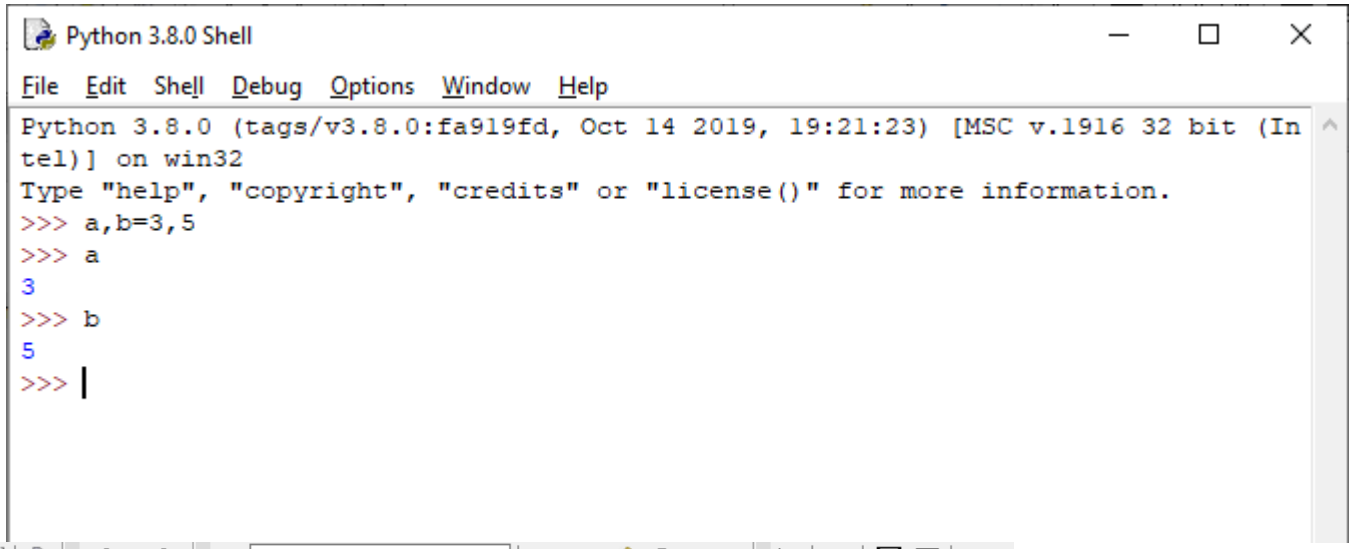
Κατηγορία	Ενδεικτικά C	Ενδεικτικά FORTRAN	Ενδεικτικά Python
Αριθμητικοί	* / % + -	* / + -	Όπως C
Λογικοί	&& || !	.AND. .OR. .NOT.	and, or, not
Συσχετιστικοί	> >= <= == !=	.GT. .GE. .EQ. .NE.	Όπως C (το διάφορο και <>)
Διαχείρισης δυαδικών ψηφίων μιας λέξης	>> & | ^	έκδοση	Όπως C
Τελεστές διαχείρισης μνήμης	& [ ] . ->		

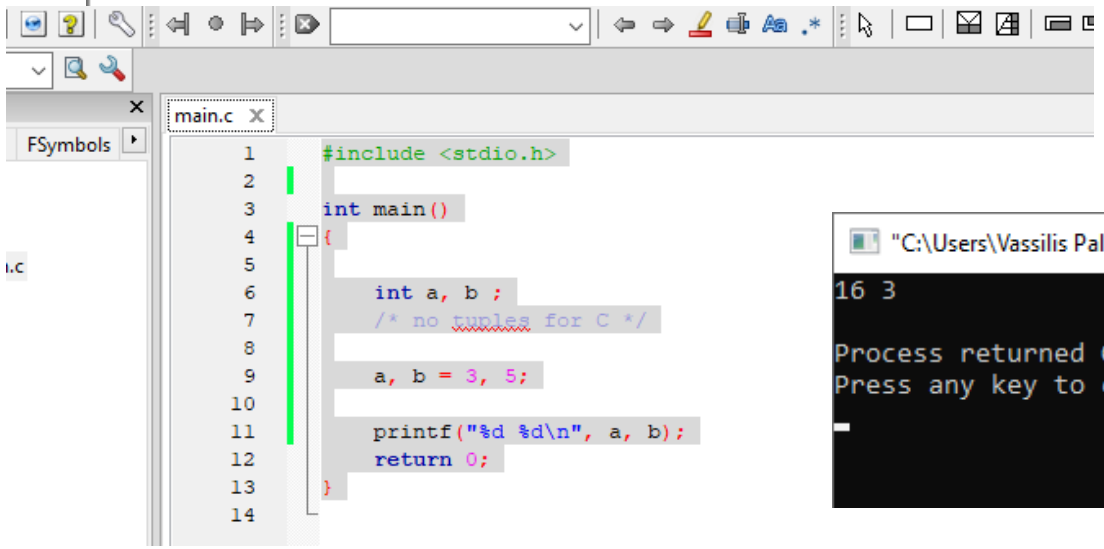
Τελεστές ⇒ Ενέργειες σε δεδομένα

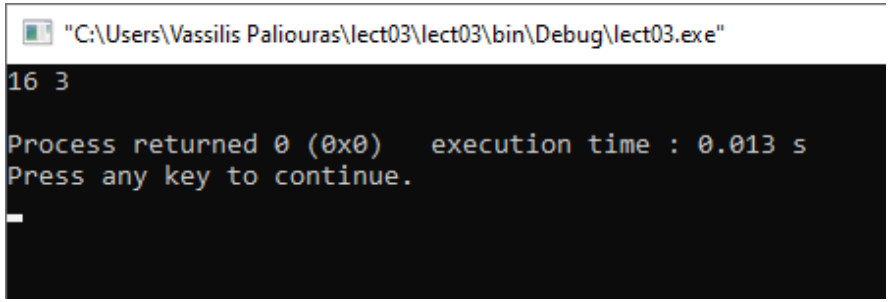
Τελεστές	Προσεταιριστικότητα
( ) [ ] -> .	Αριστερά προς δεξιά
! ~ ++ -- + - * & sizeof	Δεξιά προς αριστερά
* / %	Αριστερά προς δεξιά
+ -	Αριστερά προς δεξιά
<< >>	Αριστερά προς δεξιά
< <= > >=	Αριστερά προς δεξιά
== !=	Αριστερά προς δεξιά
&	Αριστερά προς δεξιά
^	Αριστερά προς δεξιά
|	Αριστερά προς δεξιά
&&	Αριστερά προς δεξιά
||	Αριστερά προς δεξιά
?:	Δεξιά προς αριστερά
= += -= *= /= %= &= ^= |= <<= >>=	Δεξιά προς αριστερά
,	Αριστερά προς δεξιά



Η C δεν έχει tuples







Ο τελεστής , (κόμμα) διαχωρίζει εκφράσεις

Εκφράσεις και προτάσεις

```
a  = f(g[3]) + 3*d
sum = sum + total
```

έκφραση (expression)

πρόταση (statement)

```
...
a  = f(g[3]) + 3*d ;
...
```

Προτάσεις και σύνθετες προτάσεις

πρόταση1;

```
{  πρόταση1;
   πρόταση2;
   πρόταση3; }
```

Σύνθετη πρόταση:

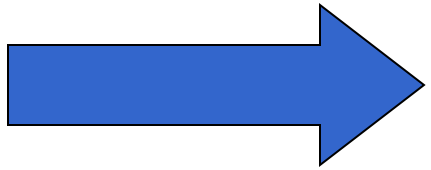
Μπλοκ προτάσεων που ορίζεται με άγκιστρα.

- ; Κενή πρόταση:
  - Δεν είναι συντακτικό λάθος.
  - Δεν κάνει κάτι.
  - Εξηγεί συμπεριφορές.

- 13; Δεν είναι συντακτικό λάθος.

Τελεστής Ανάθεσης =

```
int a, b;
a = 5;
b = a;
```



```
int a, b;
b = a = 5;
```

όλη η έκφραση έχει ως αξία την τιμή 5

σημαίνει

```
b = (a = 5);
```

Η προσεταιριστικότητα τελεστή = από δεξιά προς αριστερά

Σχεσιακοί Τελεστές

- Συγκρίσεις

  > , >= , <, <=

  Παράδειγμα

  a < 5

  αν το a είναι μικρότερο του **πέντε** η έκφραση είναι **αληθής**
  διαφορετικά είναι **ψευδής**

Έλεγχος Ισότητας

- `a == 5 ;  /* αληθές αν το a είναι 5 */`
- `a != 5 ; /* αληθές αν το a δεν είναι 5 */`
- άλλος ο ρόλος του = άλλος του ==

Διαφορετικά σε C, python!

Το a == b ==c δεν είναι το ίδιο!!!

```c
#include <stdio.h>
int main(void) {
   int a = 5, b =5, c =5;
   if (a==b==c)
        printf("equal?\n") ;
   else
        printf("or not?\n");

   return 0;
}
```

```
C:\Users\paliu\gdb\testMar3\bin\Debug\testMar3.exe
or not?

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

testif.py - C:/Users/paliu/AppData/Local/Programs/Python/Python38-32/testif.py (3.8.0)

File Edit Format Run Options Window Help

```
a=5
b=5
c=5
if a==b==c:
    print("equal?\n")
else:
    print("or not?\n")
```

Python 3.8.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct
tel)] on win32
Type "help", "copyright", "credits" or
>>>
== RESTART: C:/Users/paliu/AppData/Loca
equal?

>>>
```

C: προσεταιριστικότητα τελεστή == από αριστερά προς δεξιά

```c
#include <stdio.h>

int main(void) {

        int a = 5, b =5, c =5;

        if (a==b && b==c)
                        printf("equal\n") ;
        else
                        printf("or not\n");

        return 0;
}
```

if (/*…*/) /*…*/ else /*…*/ ;

if (έκφραση)
 (σύνθετη) εντολή 1;
else
 (σύνθετη) εντολή 2;

```
if (a == 5) {
   printf ("a equals five.\n");
   }
else {
   printf("a does not equal five\n");
   }
```

```
if (a == 5)
  printf ("a equals five.\n");
else
  printf("a does not equal five\n");
```

Παράδειγμα 1: **if** με απλή πρόταση

```c
#include <stdio.h>

int main(void) {
  int a = 3;
  printf("a:%d\n", a);
  if (a==5)
     printf("is five\n");

  printf("a:%d\n",a);
  return 0;
}
```

Παράδειγμα 2: `if` με σύνθετη πρόταση

```c
#include <stdio.h>

int main(void) {
   int a = 3;
   printf("a:%d\n", a);
   if (a==5)
      {
      printf("is five\n");
      printf("nothing else\n");
      }
   printf("a:%d\n",a);
   return 0;
}
```

(αντί)-παράδειγμα 1:
Τι θα τυπώσει; Γιατί;

```c
#include <stdio.h>

int main(void) {
  int a = 3;
  printf("a:%d\n", a);
  if (a==5)
     printf("is ");
     printf("five\n");

  printf("a:%d\n",a);
  return 0;
}
```

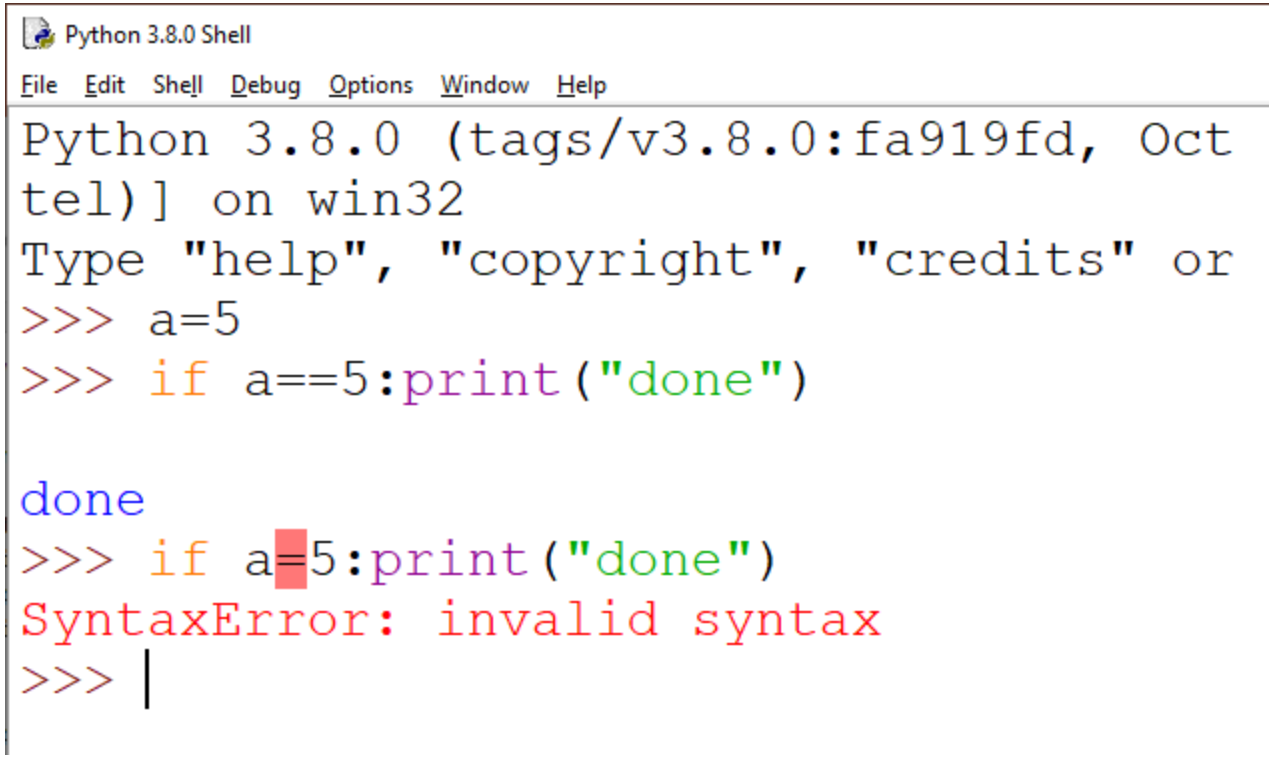
(αντί)-παράδειγμα 2:
Τι θα τυπώσει; Γιατί;

```c
#include <stdio.h>

int main(void) {
  int a = 3;
  printf("a:%d\n", a);
  if (a=5)
     printf("is five\n");

  printf("a:%d\n",a);
  return 0;
}
```

(αντί)-παράδειγμα 3:
Τι θα τυπώσει; Γιατί;

```c
#include <stdio.h>

int main(void) {
  int a = 3;
  printf("a:%d\n", a);
  if (a==5) ;
     printf("is five\n");

  printf("a:%d\n",a);
  return 0;
}
```

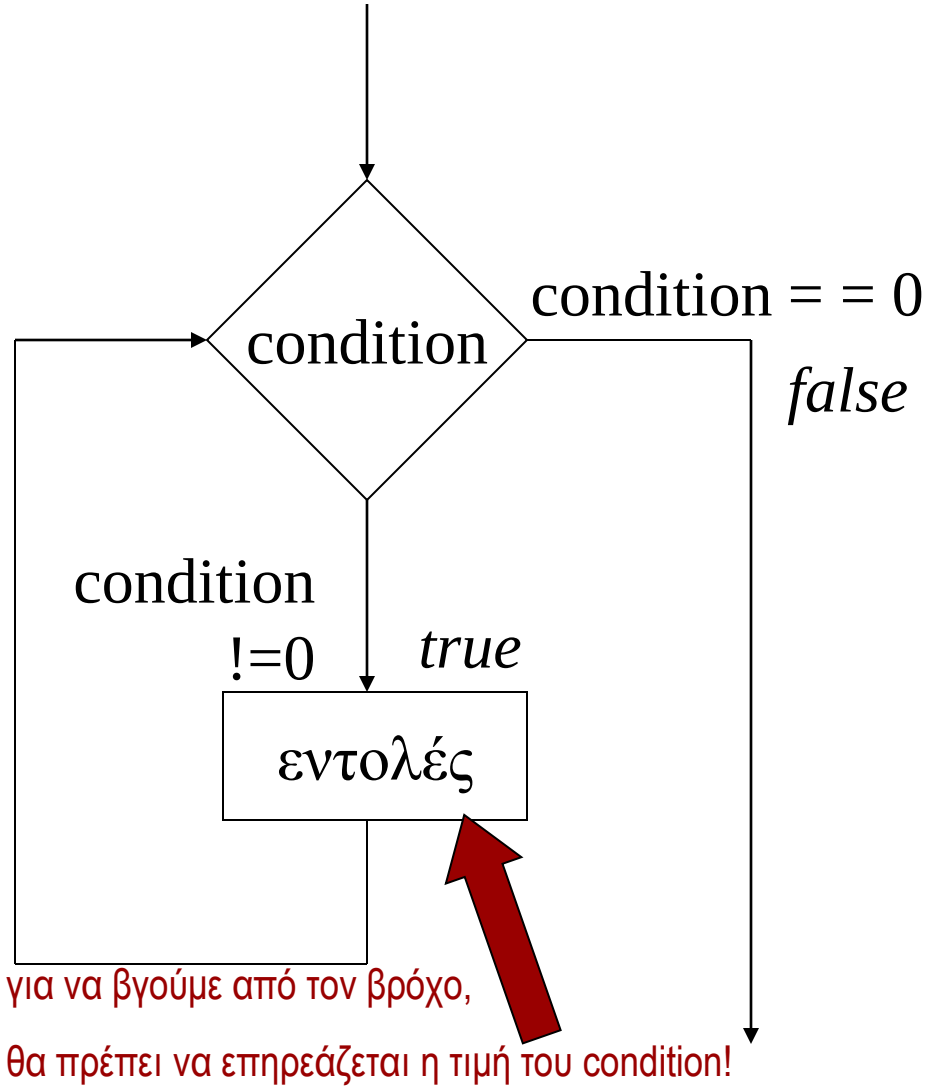
SyntaxError για την Python, όχι για τη C



Βρόχος **while**

συνθήκη **εισόδου** στο βρόχο και **συνέχειας**

```
int main (void) {
   int condition;
   condition = 1;
   while (condition) {
       printf("loop body");
       condition = f();
   }
   return 0;
}
```



Βρόχος **while**

while (έκφραση)

σύνθετη (ή όχι) πρόταση

όσο η έκφραση είναι αληθής, εκτελείται η (σύνθετη) πρόταση.

```
a = 0 ;
while (a < 5) {
  printf ("value of a is %d\n", a);
  a ++;
  }
```

Βρόχος **while** και βρόχος **for**

```
a = 0 ;
while (a < 5) {
   printf ("value of a is %d\n", a);
   a ++;
   }
```

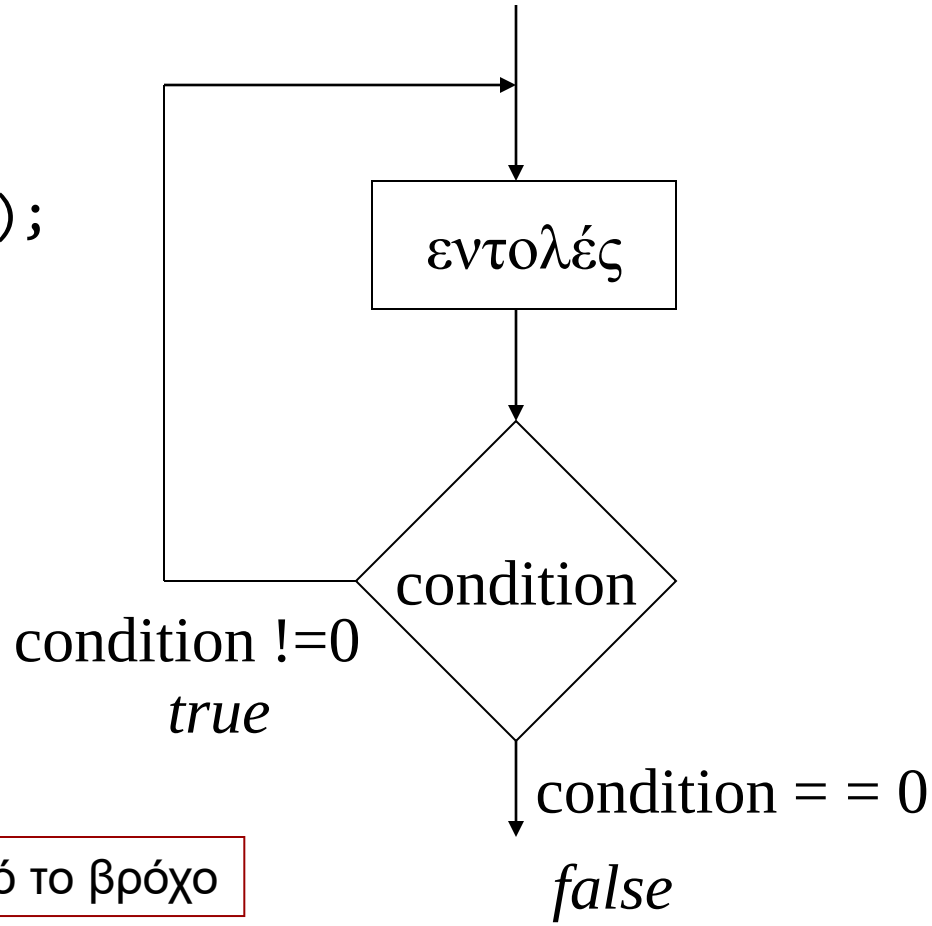
```
for (a = 0; a < 5; a ++) {
   printf ("value of a is %d\n", a);
   }
```

Στη C ο βρόχος **for** ορίζεται ως άλλη γραφή του **while**

```
for (a = 0 ; a < 5; a++)
   printf ("value of a is %d\n", a);
```

Βρόχος **do /*… */ while (/*…*/);**

```
int main (void ) {
   int condition;
   do {
        printf("loop body");
        condition = f();
   } while (condition) ;
  return 0;
}
```



συνθήκη **εξόδου** από το βρόχο

Παραδείγματα χρήσης δομών ελέγχου

1. Σταθερός αριθμός επαναλήψεων
2. Αριθμός επαναλήψεων εξαρτώμενος από τα δεδομένα
3. Ένθετες (nested) δομές ελέγχου

Παράδειγμα 1 – καθορισμένος αριθμός επαναλήψεων

- Να γραφεί ένα πρόγραμμα που διαβάζει **δέκα ακεραίους**, έναν κάθε φορά και τυπώνει το **μερικό άθροισμα**.

- **Στο τέλος** τυπώνεται το συνολικό άθροισμα και το γινόμενό τους.

- (Εδώ λύση χωρίς πίνακες).

Λεκτική περιγραφή

- Διάβασε έναν αριθμό
- Υπολόγισε το μερικό άθροισμα
- Τύπωσε το μερικό άθροισμα
- Έχεις διαβάσει δέκα αριθμούς;
  - Αν όχι, επανάλαβε.

Λεκτική περιγραφή

- Διάβασε έναν αριθμό num
- Υπολόγισε το μερικό άθροισμα sum
- Τύπωσε το μερικό άθροισμα sum
- Έχεις διαβάσει δέκα αριθμούς;
  - Αν όχι, επανάλαβε.

```
int num, sum;
```

Λεκτική περιγραφή

- Διάβασε το num
- Υπολόγισε το sum
- Τύπωσε το sum
- Έχεις διαβάσει δέκα αριθμούς;
  - Αν όχι, επανάλαβε.

Λεκτική περιγραφή

- Επανάλαβε για δέκα φορές {
- Διάβασε το num → scanf()
- Υπολόγισε το sum →computeSum() ή
→sum = sum + num
- Τύπωσε το sum → printf()
- }

```c
#include <stdio.h>

int main(void) {

  int i, num, sum=0;

  for (i=0; i<10; i++) {
   scanf("%d", &num);
   sum = sum + num;
   printf("partial sum: %d\n", sum);
   }

  printf("total: %d", sum);

  return 0;

}
```

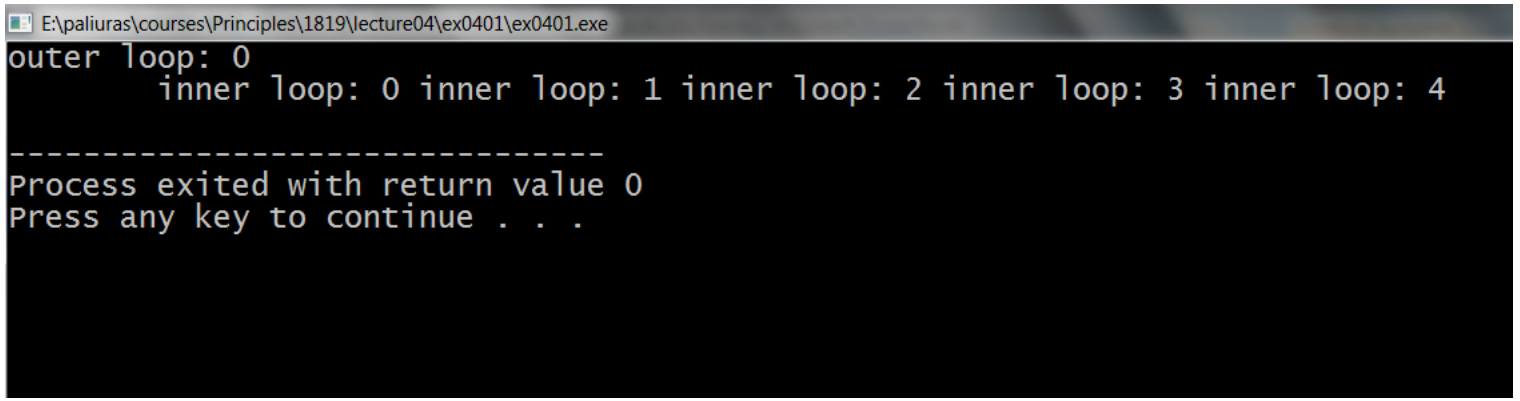
```c
#include <stdio.h>
#define N 10

int main(void) {

  int i, num, sum=0;

  for (i=0; i<N; i++) {
   scanf("%d", &num);
   sum = sum + num;
   printf("partial sum: %d\n", sum);
   }

  printf("total: %d", sum);

  return 0;
}
```

Nested for σε python

```
for i in range(5):
    print('outer loop: %d\n\t' % i, end='')
    for i in range(5):
        print('inner loop: %d ' % i, end ='')
    print('')
```

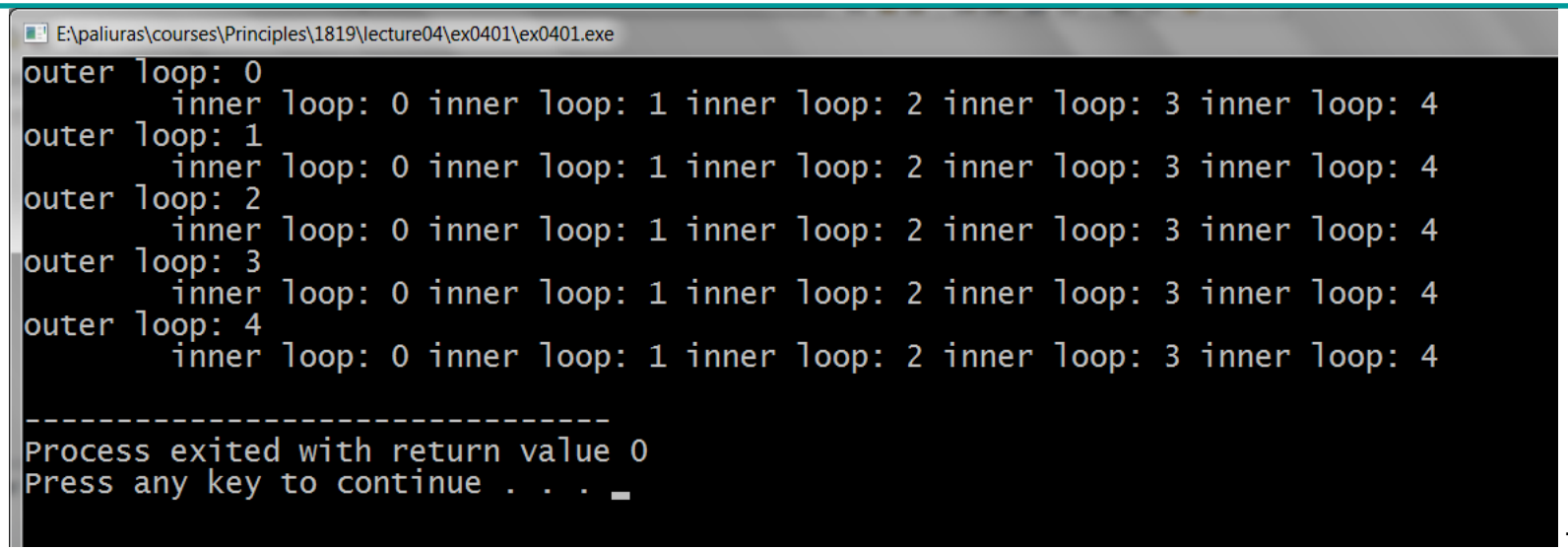
```
C:\Python34\python.exe C:/Users/user/PycharmProjects/testnestedfor/testnestedfor.py
outer loop: 0
        inner loop: 0 inner loop: 1 inner loop: 2 inner loop: 3 inner loop: 4
outer loop: 1
        inner loop: 0 inner loop: 1 inner loop: 2 inner loop: 3 inner loop: 4
outer loop: 2
        inner loop: 0 inner loop: 1 inner loop: 2 inner loop: 3 inner loop: 4
outer loop: 3
        inner loop: 0 inner loop: 1 inner loop: 2 inner loop: 3 inner loop: 4
outer loop: 4
        inner loop: 0 inner loop: 1 inner loop: 2 inner loop: 3 inner loop: 4

Process finished with exit code 0
```

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

int main(void) {

        int i;
        for (i=0; i< 5; i++) {
                printf("outer loop: %d\n\t", i);
                for (i=0; i<5; i++) {
                                printf("inner loop: %d ", i);
                }
                printf("\n");
        }

        return 0;
}
```



```c
#include <stdio.h>

int main(void) {

        int i;
        for (i=0; i< 5; i++) {
                printf("outer loop: %d\n\t", i);
                {
                int i;
                for (i=0; i<5; i++) {
                        printf("inner loop: %d ", i);
                        }
                }
                printf("\n");
        }
        return 0;
}
```

C90: δήλωση μεταβλητής μόνο σε αρχή block



Μπλοκ εντολών με άγκιστρα

C90: δήλωση μεταβλητής μόνο σε αρχή block

```
{
Δηλώσεις ;

Εντολές ;
}
```

Μια διαφορά C90 και C99

```c
#include <stdio.h>

int main(void) {

        for (int i=0; i<5; i++) {
                printf("outer loop: %d\n\t", i);
                for (int i=0; i<5; i++) {
                                printf("inner loop: %d ", i);
                }
                printf("\n");
        }

        return 0;
}
```

Συνήθης πρακτική σε C90: Διαφορετική μεταβλητή σε nested loops

```c
#include <stdio.h>

int main(void) {

        int i, j;
        for (i=0; i<5; i++) {
                printf("outer loop: %d\n\t", i);
                for (j=0; j<5; j++) {
                                printf("inner loop: %d ", j);
                }
                printf("\n");
        }

        return 0;
}
```

Παράδειγμα

- Να γραφεί ένα πρόγραμμα που διαβάζει **ακεραίους**, έναν κάθε φορά και τυπώνει το μερικό άθροισμα και το μερικό γινόμενο, **όσο** ο χρήστης δίνει ως είσοδο **αριθμούς > 0**.
- Αριθμοί ≤ 0 δεν λαμβάνονται υπόψη στους υπολογισμούς.
- Στο **τέλος** τυπώνεται το συνολικό άθροισμα και το γινόμενό τους.

έκδοση 1

```c
#include <stdio.h>
int main( void)
{

  int input, sum = 0 , prod = 1;

  scanf("%d", &input);

  while (input>0) {
        sum = sum + input ;
        printf("partial sum: %d\n", sum);
        prod = prod * input ;
        scanf("%d", &input);
   }

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

  return 0;
}
```

έκδοση 2

```c
#include <stdio.h>

int main( void)
{

  int input, sum = 0 , prod = 1;

  for (scanf("%d", &input); input>0; scanf("%d", &input) ) {
        sum = sum + input ;
        printf("partial sum: %d\n", sum);
        prod = prod * input ;
      }

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

 return 0;
}
```

```c
#include <stdio.h>

int main( void)
{

  int input, sum = 0 , prod = 1;

  scanf("%d", &input);
  for ( ; input>0; ) {
        sum = sum + input ;
        printf("partial sum: %d\n", sum);
        prod = prod * input ;
        scanf("%d", &input);
      }

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

 return 0;
}
```

Οι εκφράσεις αντίστοιχες με while

έκδοση 3

```c
#include <stdio.h>

int main(void)
{

  int input, sum = 0 , prod = 1;

  do {
     scanf("%d", &input);
     if (input >0 ) {
        sum = sum + input ;
        printf("partial sum: %d\n", sum);
        prod = prod * input ;
        }
     } while (input>0) ;

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

return 0;
}
```

έκδοση 4

```c
#include <stdio.h>

int main(void)
{

  int input, sum = 0 , prod = 1;

  do {
     scanf("%d", &input);
     if (input <=0 )
        break;
     sum = sum + input ;
     printf("partial sum: %d\n", sum);
     prod = prod * input ;
     } while (input>0) ;

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

return 0;
}
```

```c
#include <stdio.h>

int main(void)
{

  int input, sum = 0 , prod = 1;

  do {
     scanf("%d", &input);
     if (input <=0 )
        break;
     sum = sum + input ;
     printf("partial sum: %d\n", sum);
     prod = prod * input ;
     } while (1) ;

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

return 0;
}
```

έκδοση 4a

Έξοδος από βρόχο
Ούτως ή άλλως
μόνο με **break**
=>

Απλή συνθήκη στο **while**

Αρχικοποίηση του input για εξασφάλιση εισόδου στο βρόχο while

έκδοση 5

```c
#include <stdio.h>

int main( void)
{

  int input=1, sum = 0 , prod = 1;

  while ( input > 0) {
     scanf("%d", &input) ;

     if (input <=0 )
        break;

     sum = sum + input ;
     printf("partial sum: %d\n", sum);
     prod = prod * input ;
     }

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

  return 0;
}
```

```c
#include <stdio.h>

int main( void)
{

  int input=1, sum = 0 , prod = 1;

  while ( 1 ) {
     scanf("%d", &input) ;

     if (input <=0 )
        break;

     sum = sum + input ;
     printf("partial sum: %d\n", sum);
     prod = prod * input ;
     }

  printf("sum: %d\n", sum);
  printf("product: %d\n", prod);

  return 0;
}
```

Η έκφραση πάντα αληθής while

έκδοση 5.1

Σημείο εξόδου από το βρόχο while

```c
#include <stdio.h>
int getinput(void) ;

int main(void) {
        int a;
        while ( (a = getinput()) > 0) {
                printf("say something\n");
                printf("%d\n", a);
        }

        return 0;
}

int getinput(void) {
        int a;
        scanf("%d", &a);
        return a;
}
```

Σε τι διαφέρουν οι εκφράσεις

```c
a = getinput() > 0

(a = getinput()) > 0

a = (getinput() > 0)
```