Προγραμματισμός Η/Υ Ι

01.
Γνωριμία με το μάθημα
(Εισαγωγικές έννοιες)

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2019-2020 | ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ Τ.Ε.

Προγραμματισμός ...

η τέχνη της έκφρασης λύσεων για προγράμματα έτσι ώστε ένας υπολογιστής να μπορεί να εκτελέσει αυτές τις λύσεις

- εξεύρεση και βελτίωση λύσεων
- καλύτερη κατανόηση του προβλήματος μέσω της διαδικασίας προγραμματισμού

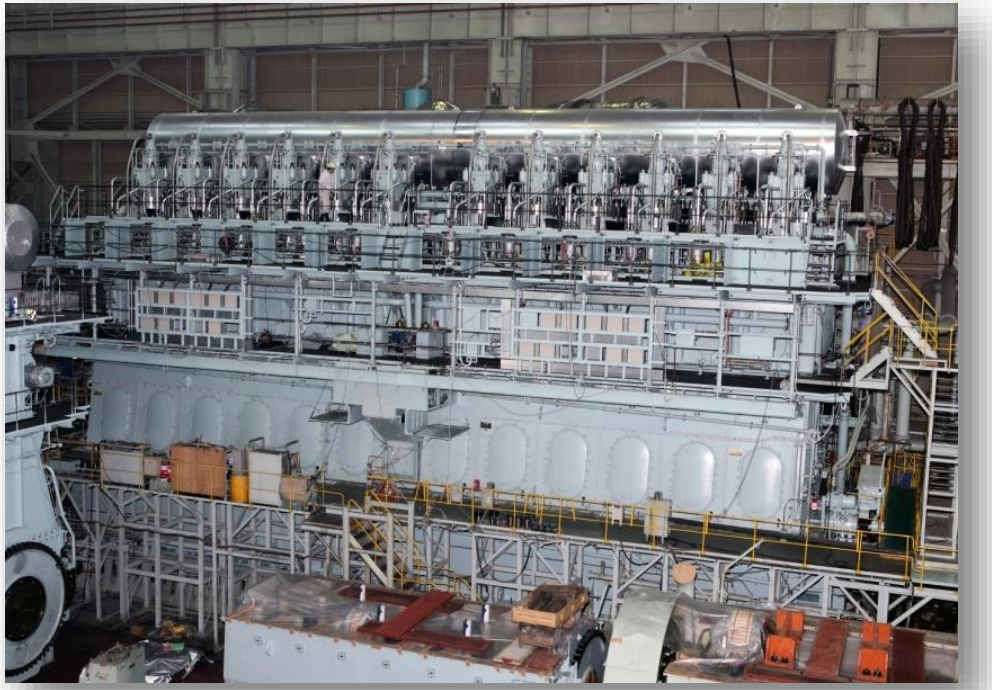
Γιατί να παρακολουθήσω το μάθημα;
Γιατί προγραμματισμός;

- ο πολιτισμός μας βασίζεται στο λογισμικό
  - εάν δεν κατανοώ το λογισμικό → νόμιζω ότι όλα γίνονται «μαγικά» → αποκλείομαι από τους πιο ενδιαφέροντες, κερδοφόρους και κοινωνικά χρήσιμους τεχνικούς τομείς εργασίας
  - προγράμματα υπολογιστών
    - εφαρμογές PC με GUI
    - υπολογισμούς και εφαρμογές ελέγχου σε διάφορα συστήματα
      - π.χ. κινητά, ψηφιακές φωτογραφικές μηχανές, αυτοκίνητα
    - εφαρμογές επεξεργασίας κειμένου
- πολύτιμη πνευματική άσκηση → οξύνει την ικανότητά μας να σκεφτόμαστε
- ένας τρόπος να αλλάξουμε τον κόσμο προς το καλύτερο

Γιατί Προγραμματισμός;

- σημείωση: τα περισσότερα προγράμματα δεν τρέχουν σε πράγματα που μοιάζουν με PC
  - μια οθόνη, ένα πληκτρολόγιο, ένα κουτί κάτω από το τραπέζι

Γιατί Προγραμματισμός;
Πλοία



- Σχεδίαση (σκελετού πλοίου)
- Κατασκευή (ναυπηγείο → ψηφιακό)
- Μηχανή (ηλεκτρονική ανάφλεξη)

- Παρακολούθηση (αυτονομία, gps)
- Διαχείριση (χρονοδιάγραμμα)

Γιατί Προγραμματισμός;
Αεροπλάνα

- Επικοινωνία
- Έλεγχος
- Κατασκευή

- Διαχείριση

Γιατί Προγραμματισμός;
Τηλεπικοινωνίες

- Ποιότητα φωνής
- Διεπαφές
- Χρεώσεις
- Φορητότητα

- Αξιοπιστία
- Μεταγωγή
- Εικόνες

Γιατί Προγραμματισμός;
Ενέργεια

- Έλεγχος
- Διαχείριση
- Ανάλυση
- Σχεδίαση

- Επικοινωνίες
- Οπτικοποίηση

Γιατί Προγραμματισμός;
PC, tablet, worksation

- Υπάρχουν πολλά περισσότερα στους υπολογιστές από παιχνίδια, επεξεργασία κειμένου, περιήγηση και υπολογιστικά φύλλα!

Ύλη μαθήματος

1. Εισαγωγή στη C
2. Βασικό συντακτικό και δομή προγράμματος
   - τύποι δεδομένων
   - εντολές ελέγχου ροής προγράμματος
3. Τελεστές
   - αριθμητικοί , λογικοί, ολίσθησης
4. Είσοδος, έξοδος δεδομένων
   - συναρτήσεις εισόδου/εξόδου
5. Πίνακες
   - μονοδιάστατοι και δισδιάστατοι
6. Συμβολοσειρές
7. Συναρτήσεις
8. Δείκτες

Ευχαριστίες

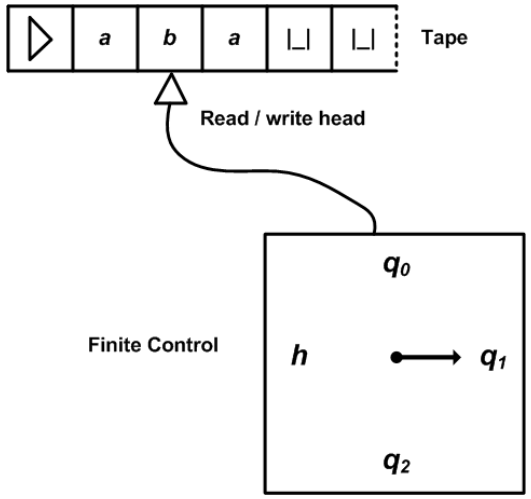
- οι διαφάνειες που θα χρησιμοποιήσουμε στο μάθημα έχουν στην πλειοψηφία τους σχεδιαστεί και υλοποιηθεί από τον Δρ. Ελευθέριο Κοσμά βασιζόμενες
  - κυρίως, σε υλικό του καθηγητή Κώστα Βασιλάκη
  - δευτερευόντως, σε υλικό των καθηγητών
    - Παναγιώτη Τσαπάρα
    - Βασίλη Χριστοφίδη
  - στη βιβλιογραφία του μαθήματος

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Σύντομη αναφορά

Λίγο Ιστορία

- Οι πρώτες γλώσσες προγραμματισμού δεν ήταν για υπολογιστές
  - αυτόματη δημιουργία πρωτοτύπων για ραπτομηχανές
  - μουσικά κουτιά ή ρολά για πιάνο
  - η αφαιρετική μηχανή του Turing



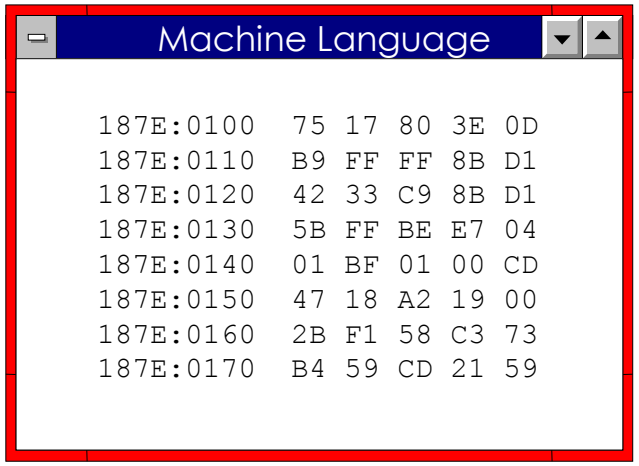
Γλώσσες Προγραμματισμού

1η γενιά: Γλώσσες Μηχανής

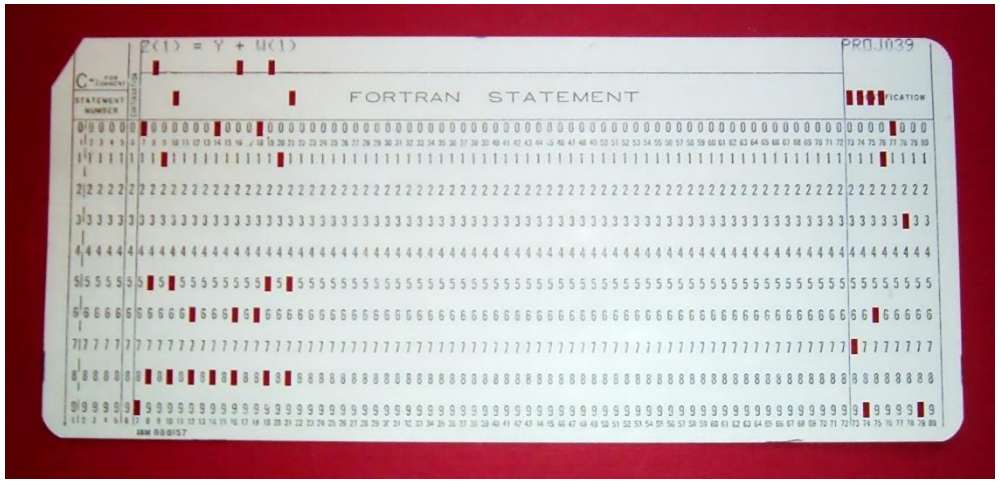
ο προγραμματιστής

1. μετατρέπει το πρόβλημα του σε ένα πρόγραμμα
   - π.χ. υπολογισμός μέγιστου κοινού διαιρέτη δύο αριθμών
2. γράφει ακριβώς τις εντολές που θα πρέπει να εκτελέσει ο υπολογιστής
   - πρέπει να γνωρίζει ακριβώς τη δυαδική αναπαράσταση των εντολών

☞ στους πρώτους υπολογιστές οι εντολές κωδικοποιούνταν σε διάτρητες κάρτες



Program entered and executed as machine language

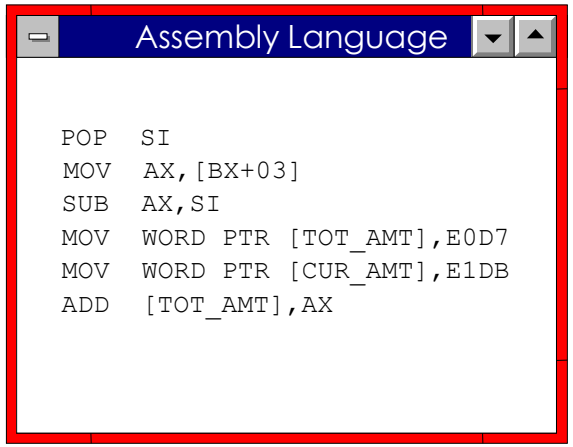


Punched card from a Fortran program: Z(1) = Y + W(1)

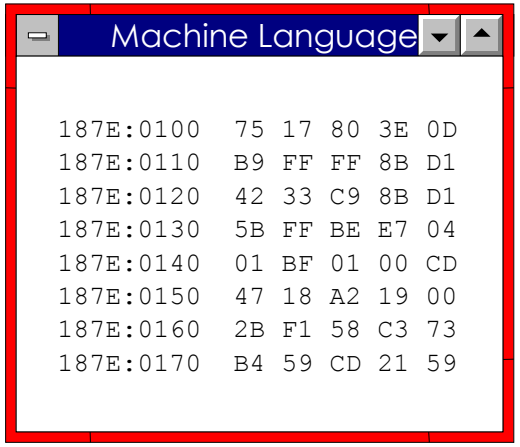
Γλώσσες Προγραμματισμού

2η γενιά: Συμβολική Γλώσσα (Assembly)

- ο προγραμματιστής
  - δε χρειάζεται να ξέρει ακριβώς την δυαδική αναπαράσταση των εντολών
  - χρησιμοποιεί πιο κατανοητούς μνημονικούς κανόνες
- ο συμβολομεταφραστής (assembler) μετατρέπει/μεταφράζει τα σύμβολα σε γλώσσα μηχανής (op-codes)
- οι συμβολικές γλώσσες εξαρτώνται από το hardware



program **entered** as **assembly** language

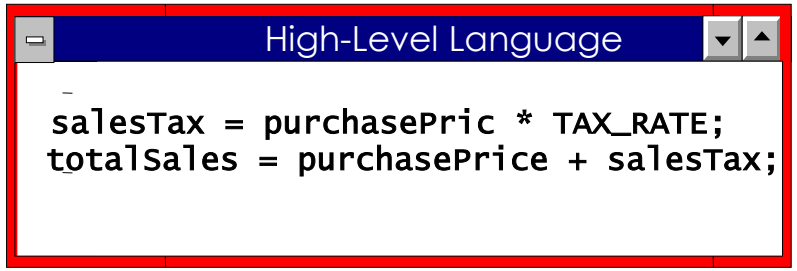


program is **translated** and **executed** as **machine** language

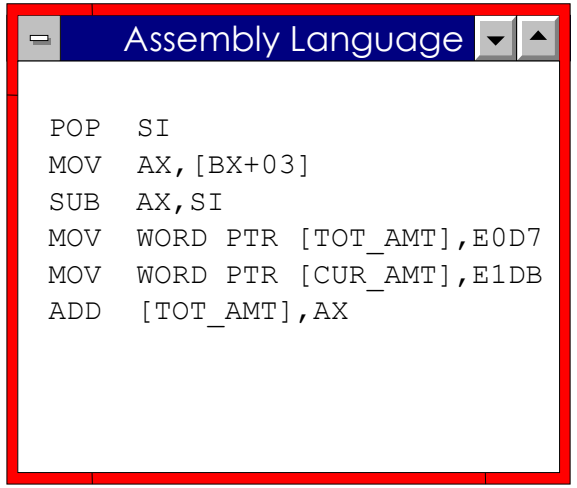
Γλώσσες Προγραμματισμού

3η γενιά: Υψηλού επιπέδου (high-level)

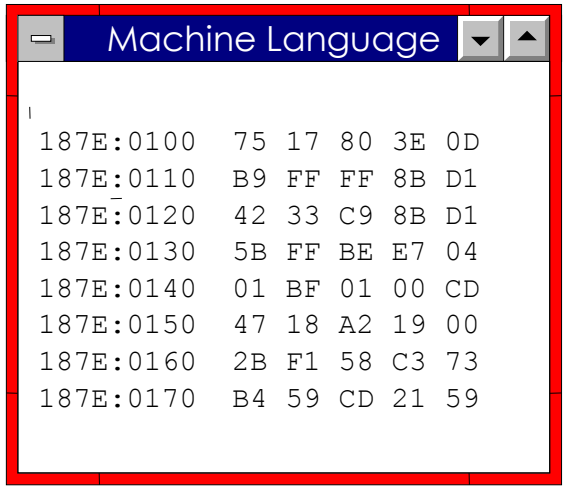
- ο προγραμματιστής δίνει εντολές στον υπολογιστή σε μια κατανοητή και καλά δομημένη γλώσσα (source code)
- ο μεταγλωτιστής (compiler) τις μετατρέπει σε ενδιάμεσο κώδικα (object code)
- ο ενδιάμεσος κώδικας μετατρέπεται σε γλώσσα μηχανής (machine code)



program **entered** as **high-level** language



program is **translated** into the **instruction set**

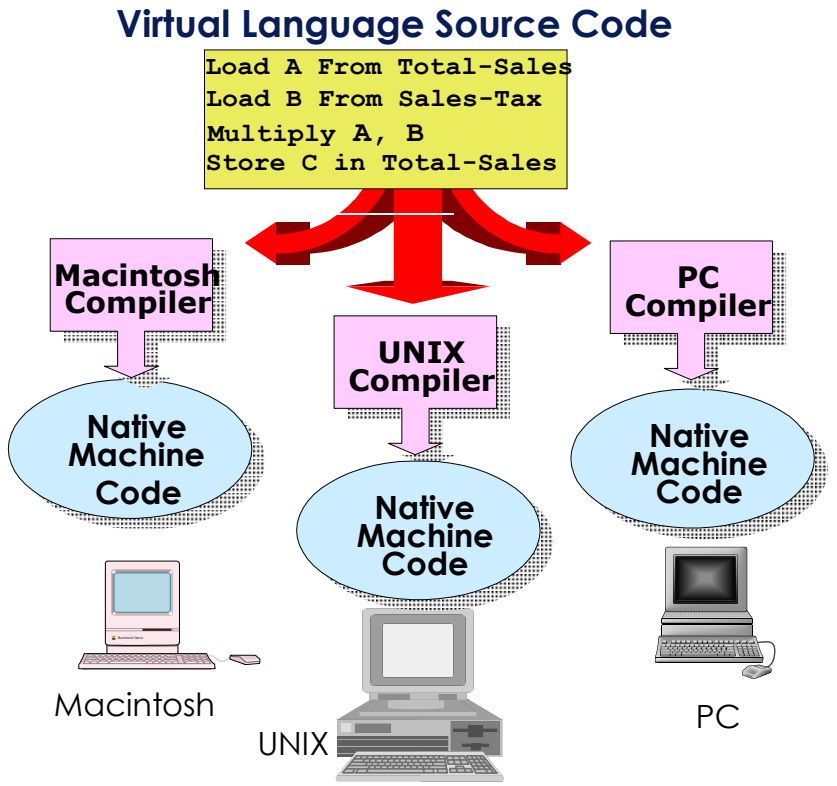


program is translated and **executed** as **machine** language

Γλώσσες Προγραμματισμού

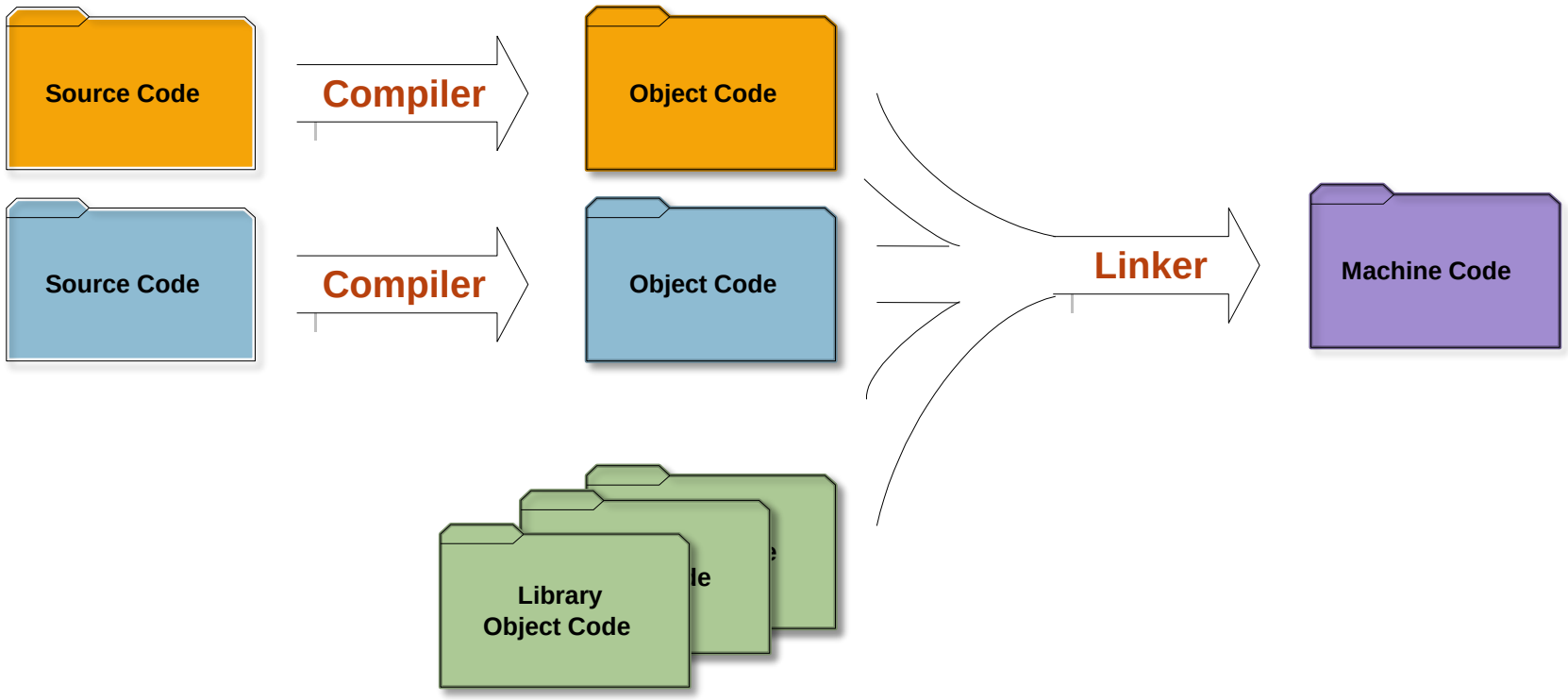
3η γενιά: Υψηλού επιπέδου (high-level) II

- ο προγραμματιστής δίνει εντολές στον υπολογιστή σε μια κατανοητή και καλά δομημένη γλώσσα (source code)
- ο μεταγλωτιστής (compiler) τις μετατρέπει σε ενδιάμεσο κώδικα (object code)
- ο ενδιάμεσος κώδικας μετατρέπεται σε γλώσσα μηχανής (machine code)
- το ίδιο πρόγραμμα μπορεί να μεταγλωττιστεί και να εκτελεστεί σε διαφορετικές μηχανές



Γλώσσες Προγραμματισμού

3η γενιά: Υψηλού επιπέδου (high-level) III



Compiling, **linking**, **and** **executing** **a program**

Γλώσσες Προγραμματισμού

Πέντε γενεές

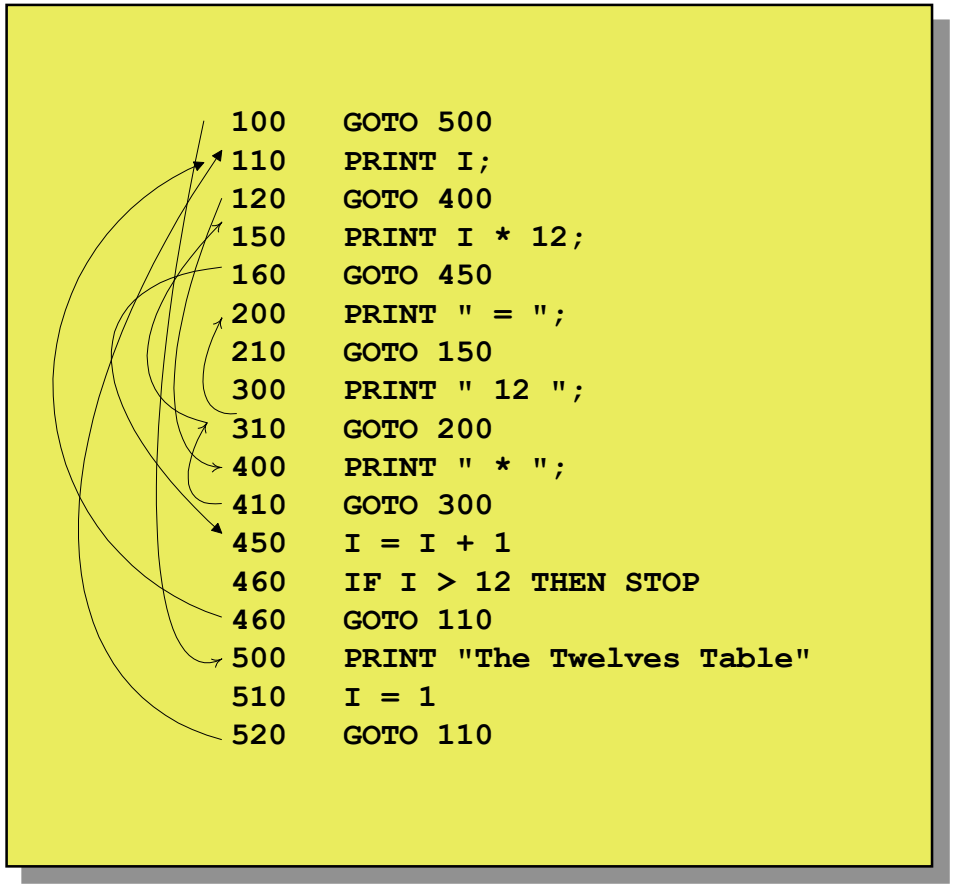
1. Γλώσσες μηχανής (machine language)
2. Συμβολικές γλώσσες (assembly language)
3. Γλώσσες υψηλού επιπέδου (high-level language)
4. Εξειδικευμένες γλώσσες
5. "Φυσικές" γλώσσες

✍ κάθε γενιά προσθέτει και ένα επίπεδο αφαίρεσης

Προγραμματιστικά Παραδείγματα

Προγραμματισμός των πρώτων ημερών

- spaghetti code
- δύσκολο να διαβαστεί και να κατανοηθεί η ροή του



```
100   GOTO 500
110   PRINT I;
120   GOTO 400
150   PRINT I * 12;
160   GOTO 450
200   PRINT " = ";
210   GOTO 150
300   PRINT " 12 ";
310   GOTO 200
400   PRINT " * ";
410   GOTO 300
450   I = I + 1
460   IF I > 12 THEN STOP
460   GOTO 110
500   PRINT "The Twelves Table"
510   I = 1
520   GOTO 110
```

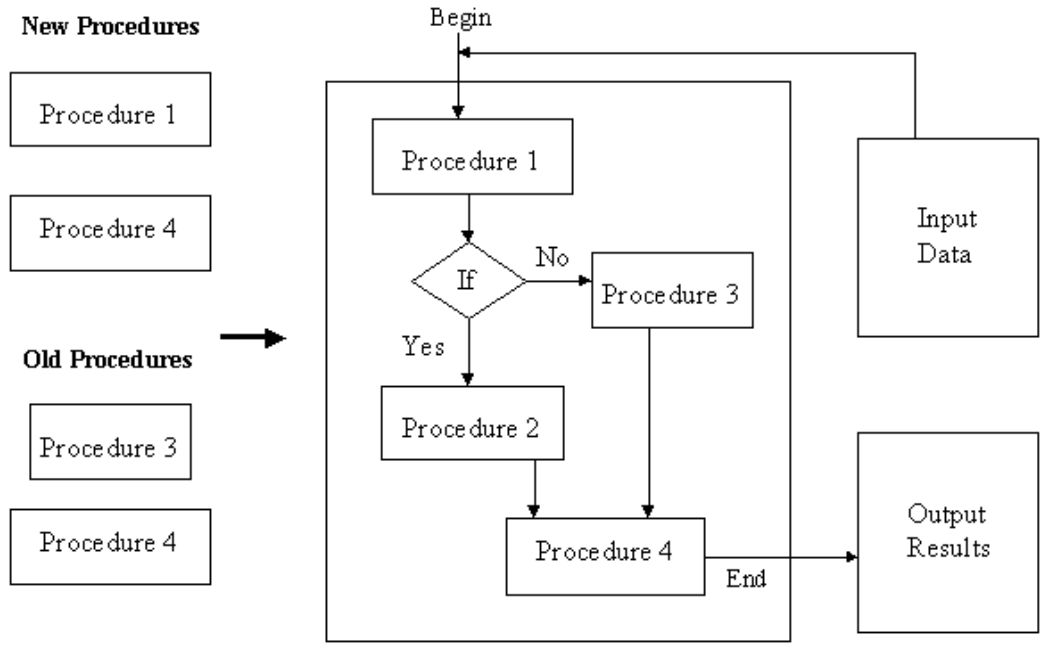
Προγραμματιστικά Παραδείγματα

Δομημένος προγραμματισμός

- τέσσερεις προγραμματιστικές δομές
  1. sequence: ακολουθιακές εντολές
  2. selection: επιλογή με if-then-else
  3. iteration: δημιουργία βρόχων
  4. recursion: αναδρομή

- ο κώδικας σπάει σε λογικά blocks που έχουν ένα σημείο εισόδου και εξόδου
  - κατάργηση της `GOTO` εντολής

- οργάνωση του κώδικα σε διαδικασίες (procedures)



Προγραμματιστικά Παραδείγματα
Διαδικασιακός προγραμματισμός (functional programming)

- το πρόγραμμα μας σπάει σε πολλαπλές διαδικασίες
  - κάθε διαδικασία λύνει ένα υπο-πρόβλημα και αποτελεί μια λογική μονάδα (module)
  - μια διαδικασία μπορούμε να την επαναχρησιμοποιήσουμε σε διαφορετικά δεδομένα

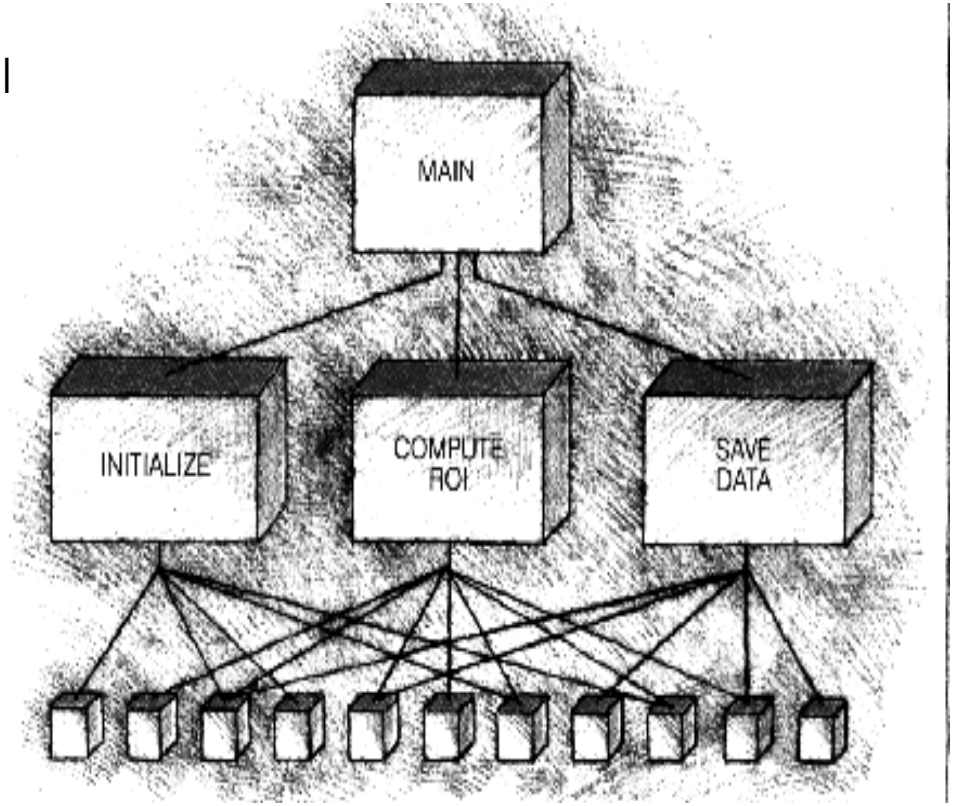
- το πρόγραμμα μας είναι τμηματοποιημένο (modular)

Προγραμματιστικά Παραδείγματα

Διαδικασιακός προγραμματισμός - Κοινά δεδομένα

- ο διαδικασιακός προγραμματισμός τμηματοποιεί τον κώδικα αλλά όχι απαραίτητα τα δεδομένα
  - π.χ., με τη χρήση καθολικών μεταβλητών (global variables) όλες οι διαδικασίες μπορεί να χρησιμοποιούν τα ίδια δεδομένα και άρα να επικοινωνούν/εξαρτώνται μεταξύ τους

✍ πρέπει να αποφεύγουμε τη χρήση καθολικών μεταβλητών!



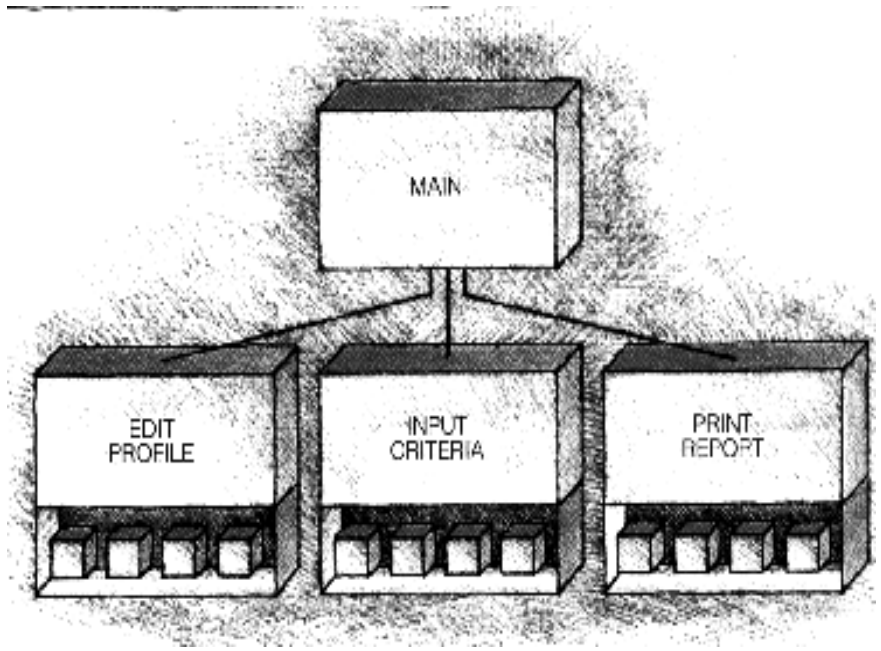
Προγραμματιστικά Παραδείγματα

Τμηματοποιημένος προγραμματισμός (modular programming)

απόκρυξη δεδομένων

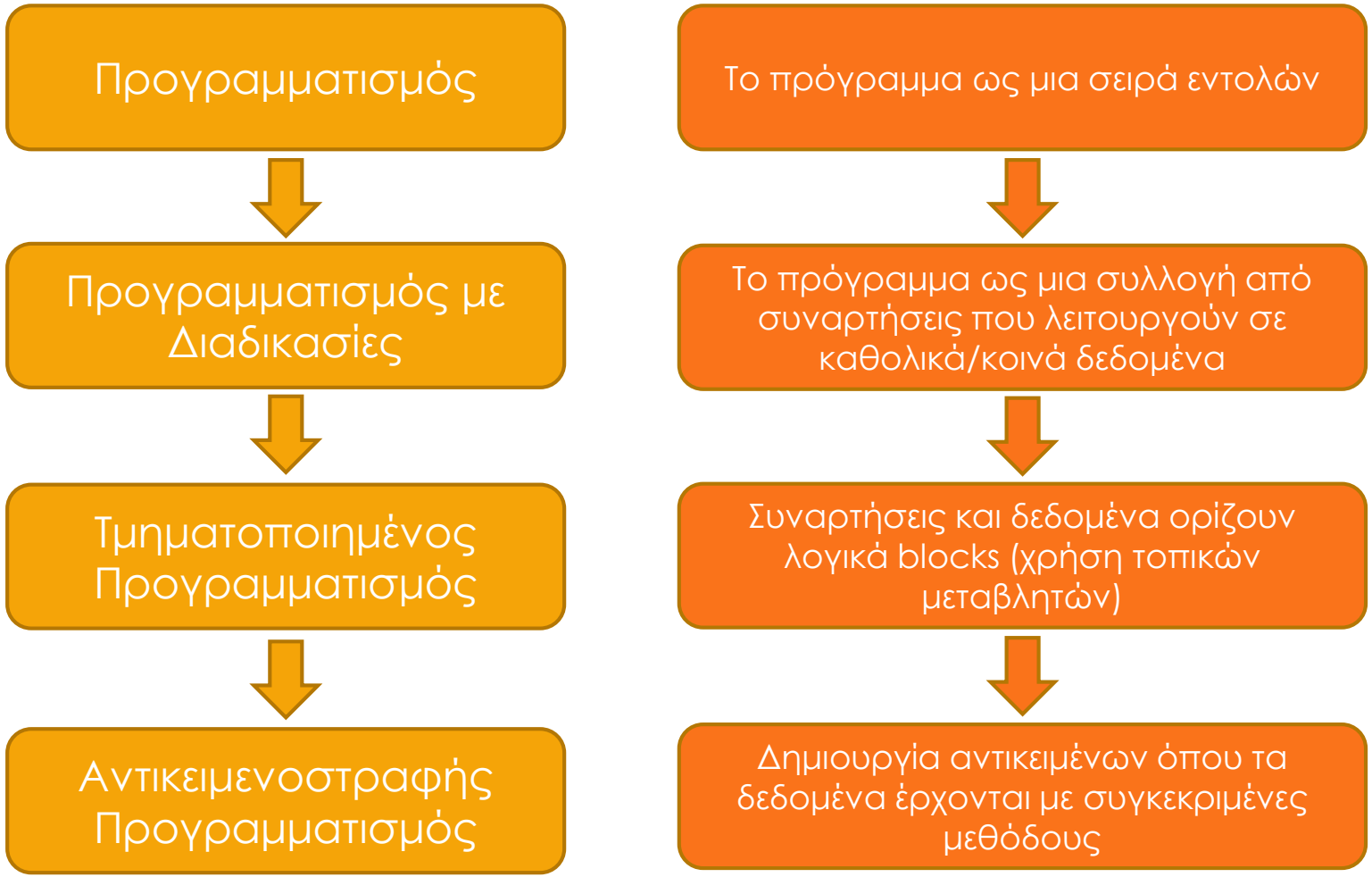
- με τη δημιουργία τοπικών μεταβλητών μέσα στις διαδικασίες αποφεύγουμε την ύπαρξη κοινών δεδομένων
- η επικοινωνία μεταξύ των διαδικασιών γίνεται με ορίσματα και επιτρεφόμενες τιμές

👍 ο κώδικας γίνεται πιο εύκολο να σχεδιαστεί, να γραφτεί και να συντηρηθεί



Προγραμματιστικά Παραδείγματα

Η εξέλιξη του προγραμματισμού - Σύνοψη



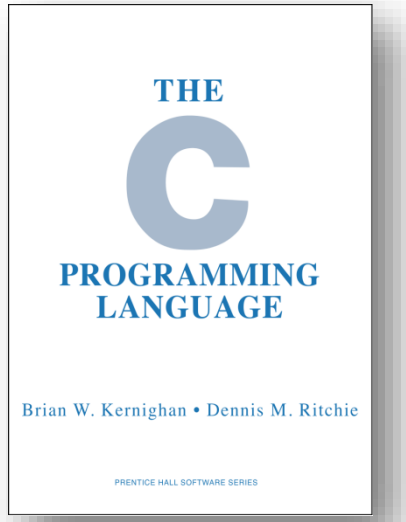
Εισαγωγή στη C

Λίγα λόγια για τη C

Bell Laboratories

- γλώσσα προγραμματισμού υψηλού επιπέδου
- σχεδιάστηκε και υλοποιήθηκε από τον Dennis Ritchie στις αρχές της δεκαετίας του 1970 (Bell Labs)
  - ? γιατί ονομάστηκε **C**;
    - πολλά από τα χαρακτηριστικά της προήλθαν από μια παλαιότερη γλώσσα, η οποία ονομαζόταν "**B**" (Ken Thompson)
- βασικός σκοπός: χρήση για την ανάπτυξη του λειτουργικού συστήματος UNIX
  - πρόδρομος του Linux, Android
- αρκετά διαδεδομένη
  - για τη δημιουργία λογισμικού συστημάτων (system)
  - για δημιουργία εφαρμογών (applications)

Η ιστορίας της C



- *1969-1973*, AT&T Bell Labs από Dennis Ritchie ως συνέχεια των BCPL, B
  - ανάπτυξη του UNIX
- *1978*, "The C Programming Language" K&R: Kernigham & Ritchie
  - Η πρώτη περιγραφή της C
- *1983*, σύσταση ANSI Standardization Committee X3J11
- *1989*, οριστικοποίηση και αποδοχή του προτύπου ANSI/ISO Standard (ANSI C)
  - ANSI: American National Standards Institute
- *1990*, το ANSI C γίνεται διαθέσιμο στο κοινό
  - από τότε όλοι οι μεταγλωττιστές C υποστηρίζουν το συγκεκριμένο πρότυπο
- *1990-99*, αναθεώρηση του προτύπου
  - μικρές αλλαγές, C99
- *σήμερα*, όλοι οι μεταγλωττιστές της C συμμορφώνονται με το πρότυπο ANSI C
  - και πολλοί με το C99

Χαρακτηριστικά της C - Πλεονεκτήματα

- ευέλικτη (λίγοι περιορισμοί) και πολύ γρήγορη
  - υποστηρίζει το δομημένο προγραμματισμό
    - μεγάλη εγκατεστημένη βάση προγραμμάτων
  - στηρίζεται στην ύπαρξη και κλήση συναρτήσεων
    - παρέχει έτοιμες συναρτήσεις
  - υποστηρίζει προγραμματισμό χαμηλού επιπέδου
    - ακόμη και την ενσωμάτωση κώδικα μηχανής (assembly code)
- λιτή
  - λίγες δεσμευμένες λέξεις και απλό συντακτικό
  - ευανάγνωστος και εύκολα διατηρήσιμος κώδικας
- μεταφέρσιμη
  - δυνατότητα μεταφοράς κώδικα και εκτέλεσής του σε διαφορετικά συστήματα

"Hello World"

Το πρώτο μας πρόγραμμα σε C

Πρόγραμμα C

1ο Παράδειγμα: "Hello World"

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.    printf ("Hello World!");      // εκτύπωσε "Hello World" στο τερματικό
8.    printf ("\n");                // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - main()

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.     printf ("Hello World!");              // εκτύπωσε "Hello World" στο τερματικό
8.     printf ("\n");                        // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

κάθε πρόγραμμα C

- ξεκινά με την εκτέλεση της κύριας συνάρτησης με όνομα **main**
- έχει μόνο μία συνάρτηση **main**

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - main() II

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.      printf ("Hello World!");              // εκτύπωσε "Hello World" στο τερματικό
8.      printf ("\n");                        // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

- κάθε πρόγραμμα C
  - ξεκινά με την εκτέλεση της κύριας συνάρτησης με όνομα **main**
  - έχει μόνο μία συνάρτηση **main**

- οι παρενθέσεις δηλώνουν ότι πρόκειται για μία συνάρτηση
- τα άγκιστρα δήλωνουν ότι πρόκειται για τον ορισμό της συνάρτησης

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Άγκιστρα

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.      printf ("Hello World!");              // εκτύπωσε "Hello World" στο τερματικό
8.      printf ("\n");                        // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

- Τα άγκιστρα **{ … }** ορίζουν μία λογική **ενότητα** (**block**) του κώδικα
  - αυτό μπορεί να είναι μία **συνάρτηση**, ένα **if statement**, ...

- οι εντολές της C πρέπει να περιέχονται μέσα σε μία λογική ενότητα

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - printf()

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.    printf ("Hello World!");          // εκτύπωσε "Hello World" στο τερματικό
8.    printf ("\n");                    // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

- η συνάρτηση **printf()**
  - εμφανίζει την ακολουθία των χαρακτήρων που βρίσκονται μέσα στα εισαγωγικά (**"..."**)
  - καλείται από την κύρια συνάρτηση **main()**, ως εντολή στις γραμμές 7 και 8
  - βρίσκεται προ-μεταγλωττισμένη στη βιβλιοθήκη **stdio.h**
- οι εντολές της C τελειώνουν πάντα με το ελληνικό ερωτηματικό (**;**)
- το **\n** ονομάζεται χαρακτήρας διαφυγής και δημιουργεί μία νέα γραμμή

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Η οδηγία #include

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.     printf ("Hello World!");              // εκτύπωσε "Hello World" στο τερματικό
8.     printf ("\n");                        // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

- οδηγία προ-επεξεργαστή: πληροφορεί το μεταγλωττιστή της C ότι θα γίνει χρήση της βιβλιοθήκης προ-μεταγλωττισμένων συναρτήσεων
  - οι έτοιμες βιβλιοθήκες της C (π.χ. `stdio.h`)
    - περιέχουν προ-μεταγλωττισμένες συναρτήσεις
    - είναι υλοποιημένες για συγκεκριμένο λειτουργικό σύστημα
    - **`stdio.h`**: περιέχει συναρτήσεις που έχουν να κάνουν με την είσοδο και έξοδο δεδομένων
      - standard input output

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Η οδηγία #include II

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.     printf ("Hello World!");              // εκτύπωσε "Hello World" στο τερματικό
8.     printf ("\n");                        // εμφάνισε αλλαγή γραμμής στο τερματικό
9.  }
```

- ξεκινάει πάντα με το σύμβολο **#** και δεν τελειώνει με ελληνικό ερωτηματικό (**;**)
- υπάρχει σχεδόν πάντα στην αρχή κάθε προγράμματος και περιλαμβάνει οδηγίες για τον προ-επεξεργαστή
- τα αρχεία που προσδιορίζονται με την οδηγία **#include** ονομάζονται αρχεία επικεφαλίδας (header files)
  - συνήθως, έχουν κατάληξη "**.h**"
  - μπορούμε να δημιουργήσουμε δικά μας αρχεία επικεφαλίδας (βιβλιοθήκες συναρτήσεων)
- σύνταξη:
  - **#include <standard header file>**
  - **#include "programmer's header file"**

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Σχόλια

```
1.  /* Το πρόγραμμα εκτυπώνει "Hello World"
2.  στο παράθυρο του τερματικού */
3.
4.  #include <stdio.h>
5.
6.  main ()
7.  {
8.     printf ("Hello World!");
9.     printf ("\n");
10. }
```

```
// εκτύπωσε "Hello World" στο τερματικό
// εμφάνισε αλλαγή γραμμής στο τερματικό
```

σχόλιο πολλαπλών γραμμών

σχόλιο μίας γραμμής

ακόμη ένα σχόλιο μίας γραμμής

- αγνοούνται από το μεταγλωττιστή
- μίας γραμμής: `// ...`
- πολλαπλών γραμμών:

```
/* ...
...
... */
```

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Άλλες μορφές

```c
/* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */

#include <stdio.h>

main ()
{
   printf ("Hello World!");                // εκτύπωσε "Hello World" στο τερματικό
   printf ("\n");                          // εμφάνισε αλλαγή γραμμής στο τερματικό
}
```

```c
/* Το πρόγραμμα εκτυπώνει "Hello World" στο παράθυρο του τερματικού */

#include <stdio.h>

main ()
{
   printf ("Hello World!\n");        // εκτύπωσε "Hello World" στο τερματικό και άλλαξε γραμμή
}
```

Πρόγραμμα C

1ο Παράδειγμα: "Hello World" - Άλλες μορφές II

```c
#include <stdio.h>

main ()
{
   printf ("Hello World!");
   printf ("\n");
}
```

```c
#include <stdio.h>

main ()
{
   printf ("Hello World!\n");
}
```

```c
#include <stdio.h>

main () {
   printf ("Hello World!\n");
}
```

```c
#include <stdio.h>

void main ()
{
   printf ("Hello World!\n");
}
```

- **void** (πριν το όνομα της συνάρτησης) → η συνάρτηση δεν επιστρέφει τιμή

```c
#include <stdio.h>

void main (void)
{
   printf ("Hello World!\n");
}
```

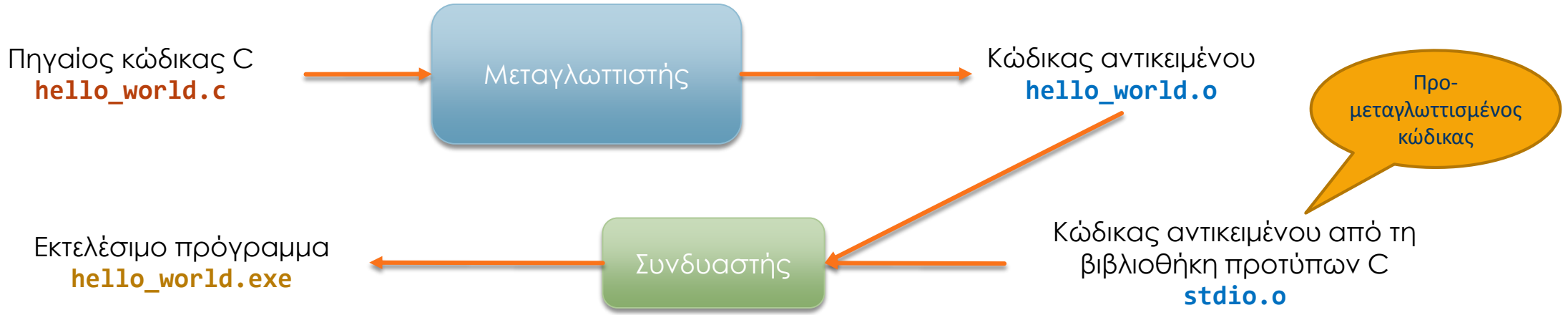
- **void** (στις παρενθέσεις) → η συνάρτηση δε δέχεται ορίσματα

```c
#include <stdio.h>

int main (void)
{
   printf ("Hello World!\n");
   return 0; // επέστρεψε την τιμή 0
}
```

- **int** (πριν το όνομα της συνάρτησης) → η συνάρτηση επιστρέφει έναν **ακέραιο**
  - με χρήση της εντολής **return**

Μεταγλώττιση και σύνδεση



1. γράφουμε πηγαίο κώδικα C (`hello_world.c`)
   - ο πηγαίος κώδικας είναι αναγνώσιμος από τον άνθρωπο
2. ο μεταγλωττιστής μεταφράζει αυτό που γράψαμε σε «αντικείμενο» κώδικα (`hello_world.o`)
   - ο «αντικείμενος» κώδικας είναι αρκετά απλός ώστε ο υπολογιστής να "καταλάβει"
3. ο συνδυαστής συνδέει τον κωδικά μας με τον «αντικείμενο» κώδικα συστήματος που απαιτείται για την εκτέλεση
   - π.χ. βιβλιοθήκες εισόδου / εξόδου (`stdio.o`), κώδικας λειτουργικού συστήματος και κώδικας παραθύρου
4. το αποτέλεσμα είναι ένα εκτελέσιμο πρόγραμμα
   - π.χ. ένα αρχείο `hello_world.exe` στα Windows ή ένα αρχείο `hello_world.out` στο Unix

Πρόγραμμα C
Μεταγλωττιστής

- ανιχνεύει λανθασμένη εφαρμογή των κανόνων της γλώσσας → μηνύματα λάθους
  - π.χ. ορθογραφικά λάθη, συντακτικά λάθη, αδήλωτες μεταβλητές, ...
    - η C κάνει διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων (case sensitive)
      - π.χ. άλλο το **`Printf()`** και άλλο το **`printf()`**
  - ένα μήνυμα λάθους μπορεί να μη συμβαίνει στη γραμμή που υποδεικνύει, αλλά στις αμέσως προηγούμενες γραμμές
  - αν εμφανίζει πολλά μηνύματα λάθους, προσπαθήστε να βρείτε και να διορθώσετε μόνο το πρώτο λάθος
    - μια νέα μεταγλώττιση μπορεί να εμφανίσει λιγότερα ή και καθόλου λάθη
- δεν ανιχνεύει λάθη λογικής που ενδέχεται να περιέχονται στο πρόγραμμά σας
- μπορεί να εμφανίσει μηνύματα προειδοποίησης (warnings)
  - καλό είναι να μην τα αγνοείτε