Διάλεξη 16 - Πολυπλοκότητα και Προεπεξεργαστής

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινίσεις

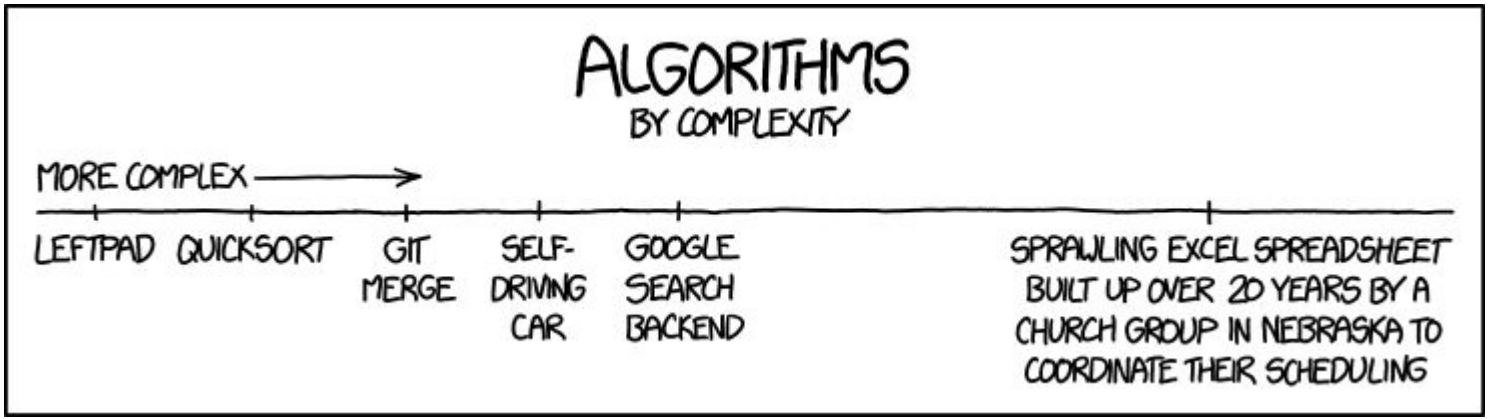
- N/A

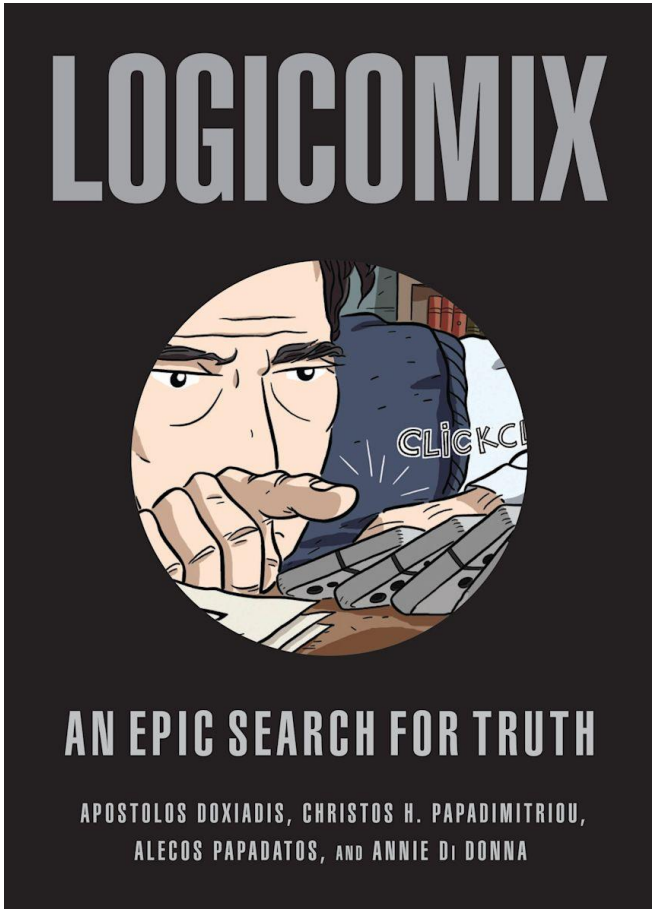
Χθες

- Επίλυση Προβλημάτων
- Επίλυση Απορριών

Σήμερα

- Πολυπλοκότητα
- Επίλυση Προβλημάτων
- Προεπεξεργαστής




LOGICOMIX
CLiCKCL
AN EPIC SEARCH FOR TRUTH
APOSTOLOS DOXIADIS, CHRISTOS H. PAPADIMITRIOU,
ALECOS PAPADATOS, AND ANNIE DI DONNA

Γ4 **Ferrari FXX K**

Μεγ. ταχύτητα km: 425 ▲	Επιτάχυνση 0-100km: 2,7 ▼
Κυβικά cc: 6.262 ▲	Ιπποδύναμη bhp: 1.050 ▲
Βάρος kg: 1.495 ▼	Αξία €: 3.100.000 ▲

Γ1: McLaren P1
Γ2: Lotec Sirius
Γ3: Koenigsegg Regera

Β2 **SSC Tuatara**

Μεγ. ταχύτητα km: 444 ▲	Επιτάχυνση 0-100km: 2,5 ▼
Κυβικά cc: 6.941 ▲	Ιπποδύναμη bhp: 1.350 ▲
Βάρος kg: 1.250 ▼	Αξία €: 1.300.000 ▲

Β1: Lamborghini Aventador s
Β3: Bugatti Chiron
Β4: SSC Ultimate Aero TT

Ζ1 **Lamborghini Veneno Roadster**

Μεγ. ταχύτητα km: 355 ▲	Επιτάχυνση 0-100km: 2,9 ▼
Κυβικά cc: 6.498 ▲	Ιπποδύναμη bhp: 750 ▲
Βάρος kg: 1.490 ▼	Αξία €: 5.000.000 ▲

Ζ2: Aston Martin Vulcan
Ζ3: Vector Avtech WX8
Ζ4: Ferrari LaFerrari

Πολυπλοκότητα (Complexity)

Η πολυπλοκότητα είναι ένα μέτρο εκτίμησης της απόδοσης αλγορίθμων ως συνάρτηση του μεγέθους του προβλήματος που επιλύουν. Δύο μετρικές:

1. Χρόνος εκτέλεσης
2. Χώρος μνήμης που απαιτείται

Παράδειγμα: έστω n το μέγεθος του προβλήματος. Θέλουμε να μπορούμε να εκφράσουμε τον χρόνο εκτέλεσης του αλγορίθμου μας ως: `t = f(n)`.

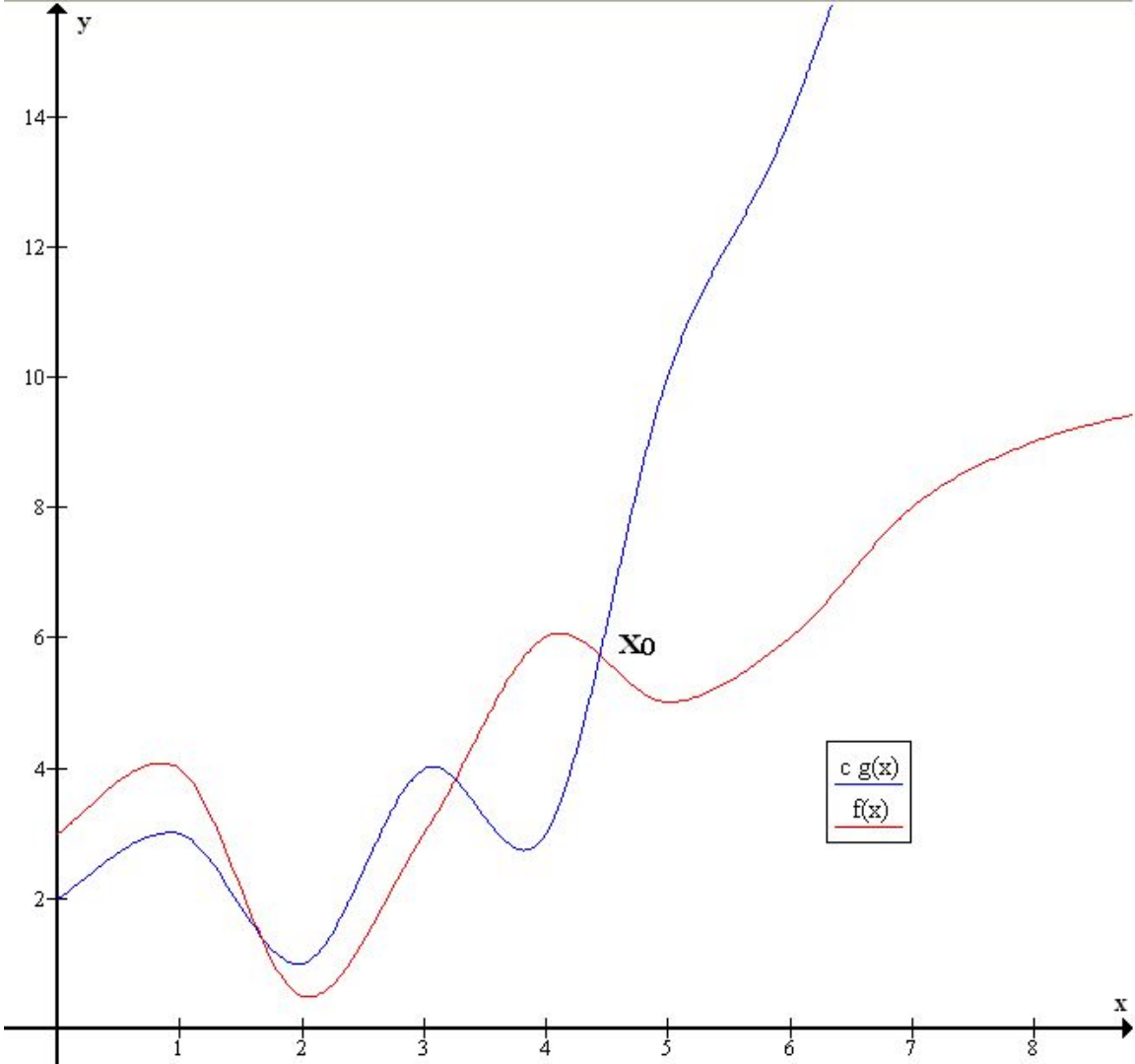
Ένα Λεπτό

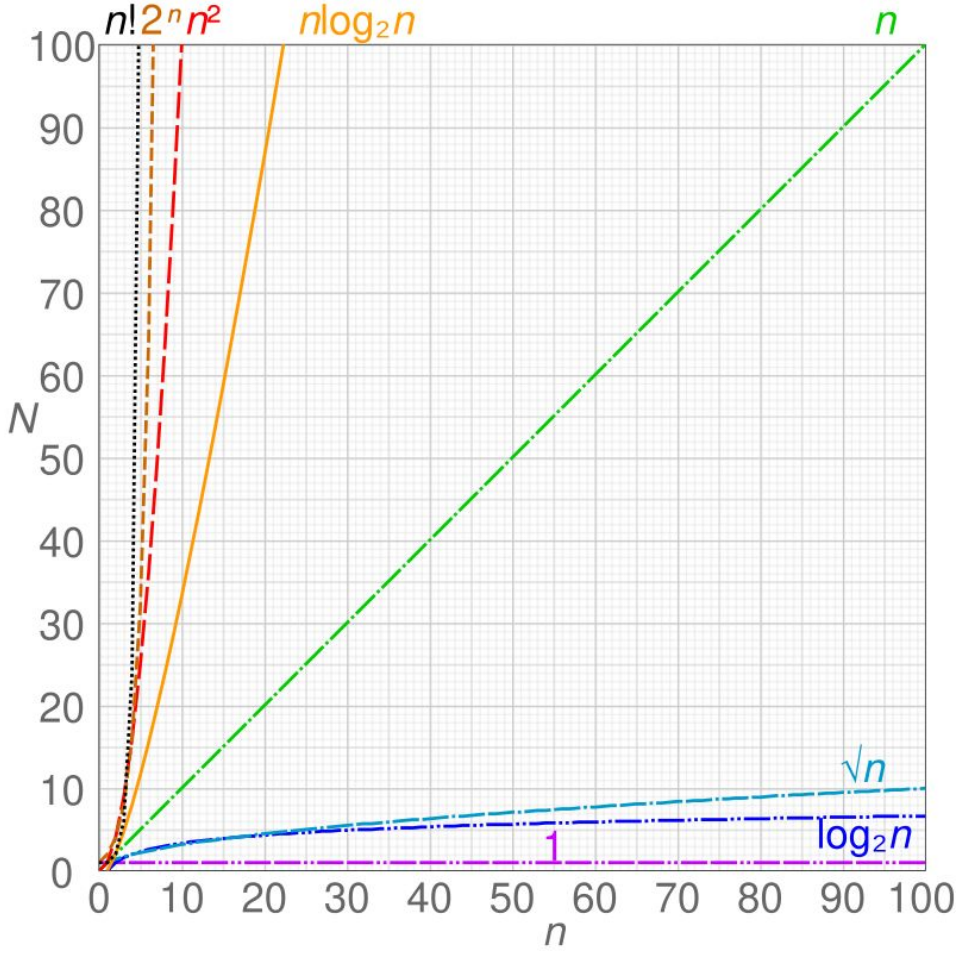
Πολυπλοκότητα και Big-O Notation

- Ορισμός κλάσεων πολυπλοκότητας
  a. Άνω όριο $$g = O(f) \Leftrightarrow \exists c.\exists n_0.\forall n > n_0. \quad g(n) < c \cdot f(n)$$
  b. Κάτω όριο $$g = \Omega(f) \Leftrightarrow \exists c.\exists n_0.\forall n > n_0. \quad g(n) > c \cdot f(n)$$
  c. Τάξη μεγέθους
  $$g = \Theta(f) \Leftrightarrow \exists c_1, c_2.\exists n_0.\forall n > n_0. \quad c_1 \cdot f(n) < g(n) < c_2 \cdot f(n)$$

- Διάταξη μερικών κλάσεων πολυπλοκότητας
$$O(1) < O(logn) < O(\sqrt{n}) < O(n) < O(nlogn)$$
$$< O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

$f(x) = O(g(x))$




n!
2ⁿ
n²
nlog₂n
n
√n
log₂n
1
N
n
0
10
20
30
40
50
60
70
80
90
100

Θέλω να το γινόμενο όλων των περιττών από το 1 μέχρι το N που διαιρούνται με το 7. Πως;

Ένα for loop με μια μεταβλητή που αυξάνεται και έλεγχο για mod 7 = 0:

```
for(product = 1, i = 1; i < N ; i += 2) {
  if ( i % 7 == 0 )
    product *= i;
}
```

Θέλω να το γινόμενο όλων των περιττών από το 1 μέχρι το N που διαιρούνται με το 7. Πως;

Ένα for loop με μια μεταβλητή που αυξάνεται και έλεγχο για mod 7 = 0:

```
for(product = 1, i = 1; i < N ; i += 2) {
  if ( i % 7 == 0 )
    product *= i;
}
```

Χρόνος: O(N)
Χώρος: O(1)

Θέλω μια συνάρτηση atoi που να παίρνει ένα πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πως;

```
int atoi(char digits[]) {
  int result = 0;
  for(int i = 0; digits[i]; i++) {
     result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Θέλω μια συνάρτηση atoi που να παίρνει ένα πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πως;

```
int atoi(char digits[]) {
  int result = 0;
  for(int i = 0; digits[i]; i++) {
     result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Χρόνος: O(N)
Χώρος: O(1)

Χρήση της συνάρτησης `getchar()`

Διαδοχικές κλήσεις της getchar() διαβάζουν διαδοχικούς χαρακτήρες. Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {

  int ch, sum = 0;

  printf("Enter characters: ");

  while( (ch = getchar()) != '\n' && ch != EOF ) {

       printf("%c", ch);

       sum++;

  }

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

  return 0;

}
```

Χρήση της συνάρτησης `getchar()`

Διαδοχικές κλήσεις της getchar() διαβάζουν διαδοχικούς χαρακτήρες. Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int ch, sum = 0;
  printf("Enter characters: ");
  while( (ch = getchar()) != '\n' && ch != EOF ) {
       printf("%c", ch);
       sum++;
  }
  printf("\nTotal characters: %d\n", sum);
  return 0;
}
```

Χρόνος: O(N)
Χώρος: O(1)

Δυναμικοί Πίνακες με την συνάρτηση malloc()

Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Παράδειγμα:

```
int * array = malloc(N * sizeof(int));
for(int i = 0 ; i < N ; i++)
  array[i] = i * i;
```

Δυναμικοί Πίνακες με την συνάρτηση malloc()

Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Παράδειγμα:

```
int * array = malloc(N * sizeof(int));
for(int i = 0 ; i < N ; i++)
  array[i] = i * i;
```

Χρόνος: O(N)
Χώρος: O(N)

Το γνωστό μας Παράδειγμα: Υπολογισμός Βαθμολογίας

```
// Compute grades using the class formula
int grade(int final_exam, int homework, int lab, int year) {
    if (year <= 1) {
        return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
    } else {
        return final_exam * 70 / 100 + homework * 30 / 100;
    }
}
```

Το γνωστό μας Παράδειγμα: Υπολογισμός Βαθμολογίας

```
// Compute grades using the class formula
int grade(int final_exam, int homework, int lab, int year) {
    if (year <= 1) {
        return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
    } else {
        return final_exam * 70 / 100 + homework * 30 / 100;
    }
}
```

Χρόνος: O(1)
Χώρος: O(1)

Εύρεση Μέγιστου Στοιχείου σε Πίνακα N x N

```
int find_max(int **matrix, size_t n) {
  int i, j, max = -1;
  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      if (matrix[i][j] > max) max = matrix[i][j];
    }
  }
  return max;
}
```

Εύρεση Μέγιστου Στοιχείου σε Πίνακα N x N

```
int find_max(int **matrix, size_t n) {
  int i, j;
  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      if (matrix[i][j] > max) max = matrix[i][j];
    }
  }
  return max;
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Η συνάρτηση παραγοντικό (factorial)

```
int factorial(int n) {
  if (n == 0) return 1;
  return n * factorial(n - 1);
}
```

Η συνάρτηση παραγοντικό (factorial)

```
int factorial(int n) {
  if (n == 0) return 1;
  return n * factorial(n - 1);
}
```

Χρόνος: O(n)
Χώρος: O(n)

Η συνάρτηση fibonacci

```
int fib(int n) {
  if (n == 0 || n == 1) return 1;
  return fib(n - 1) + fib(n - 2);
}
```

Η συνάρτηση fibonacci

```
int fib(int n) {
  if (n == 0 || n == 1) return 1;
  return fib(n - 1) + fib(n - 2);
}
```

Χρόνος: $O(2^n)$
Χώρος: $O(n)$

Η συνάρτηση strlen

Μια πιθανή υλοποίηση:

```
size_t strlen(char * str) {
  size_t length = 0;
  while(*str++) length++;
  return length;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strlen ως προς το μέγεθος της συμβολοσειράς;

Η συνάρτηση strlen

Μια πιθανή υλοποίηση:

```
size_t strlen(char * str) {
  size_t length = 0;
  while(*str++) length++;
  return length;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strlen ως προς το μέγεθος της συμβολοσειράς;

Χρόνος: O(n)
Χώρος: O(1)

Η συνάρτηση strcmp

Μια πιθανή υλοποίηση:

```
int strcmp(char * str1, char * str2) {
  while(*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strcmp ως προς το μέγεθος των συμβολοσειρών (έστω n και m);

Η συνάρτηση strcmp

Μια πιθανή υλοποίηση:

```
int strcmp(char * str1, char * str2) {
  while(*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strcmp ως προς το μέγεθος των συμβολοσειρών (έστω n και m);

Χρόνος: O(min(m, n))
Χώρος: O(1)

Άθροισμα τέλειων τετραγώνων

```
int isPerfectSquare(int n) {
  int root = sqrt(n);
  return root * root == n;
}
int low = atoll(argv[1]);
int high = atoll(argv[2]);
int i, sum = 0;
for(i = low ; i <= high ; i++) {
  if (isPerfectSquare(i))
    sum += i;
```

Άθροισμα τέλειων τετραγώνων

```
int isPerfectSquare(int n) {
  int root = sqrt(n);
  return root * root == n;
}
int low = atoll(argv[1]);
int high = atoll(argv[2]);
int i, sum = 0;
for(i = low ; i <= high ; i++) {
  if (isPerfectSquare(i))
    sum += i;
```

Χρόνος: O(n)
Χώρος: O(1)

Άθροισμα τέλειων τετραγώνων

```
int low = atoll(argv[1]);

int high = atoll(argv[2]);

int i, sum = 0;

for(i = sqrt(low) ; i <= sqrt(high) ; i++)

  sum += i*i ;
```

Άθροισμα τέλειων τετραγώνων

```
int low = atoll(argv[1]);
int high = atoll(argv[2]);
int i, sum = 0;
for(i = sqrt(low) ; i <= sqrt(high) ; i++)
  sum += i*i ;
```

Χρόνος: O(√n)
Χώρος: O(1)

Εύρεση του κατόπτρου ενός ακεραίου

```
int mirror(int n) {
  int result = 0, tmp;
  while(n > 0) {
    tmp = n % 10;
    result = 10 * result + tmp;
    n /= 10;
  }
  return result;
}
```

Εύρεση του κατόπτρου ενός ακεραίου

```
int mirror(int n) {
  int result = 0, tmp;
  while(n > 0) {
    tmp = n % 10;
    result = 10 * result + tmp;
    n /= 10;
  }
  return result;
}
```

Χρόνος: O(logn)
Χώρος: O(1)

Έλεγχος αν ένας αριθμός είναι πρώτος - τι χρονική πολυπλοκότητα έχει;

Μεταγλωττιστές και Προεπεξεργαστές

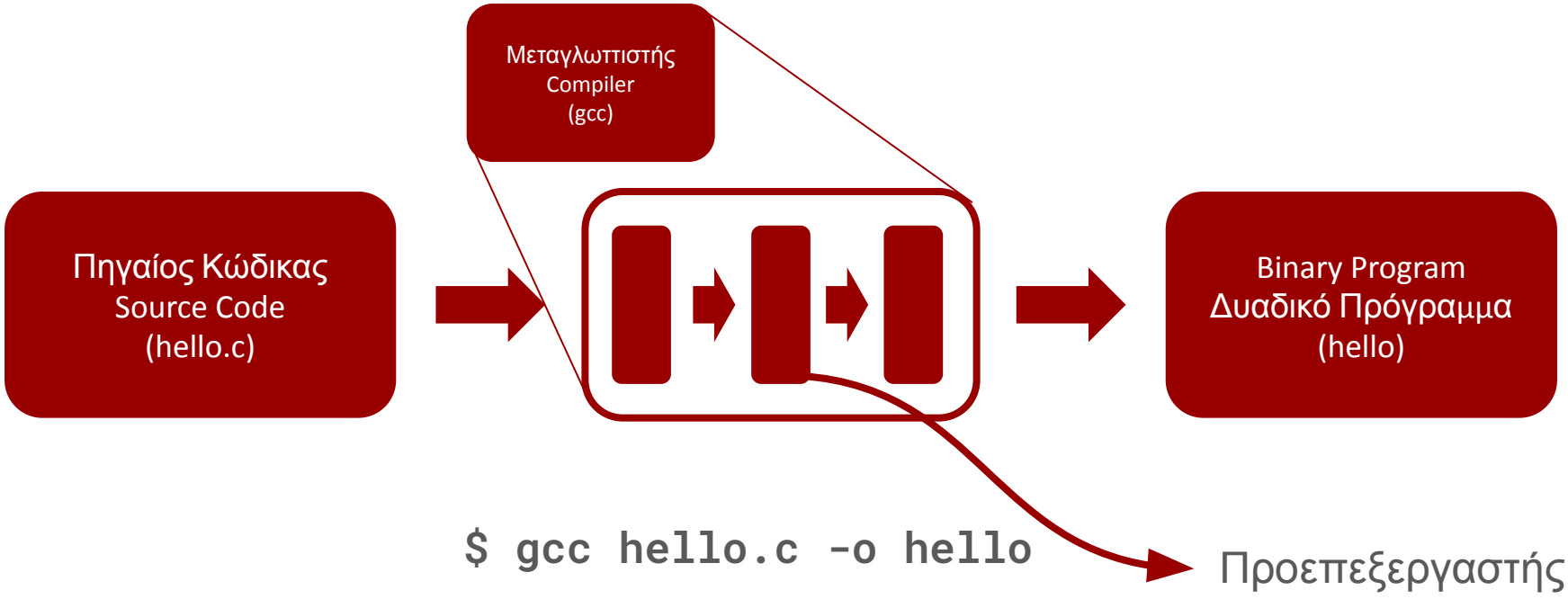
Μεταγλωττιστές (Compilers)

Μεταγλωττιστής (compiler) είναι ένα πρόγραμμα που μετατρέπει εντολές μιας γλώσσας προγραμματισμού σε κώδικα μηχανής ώστε να μπορεί να διαβαστεί και να τρέξει από τον υπολογιστή.


```
$ gcc hello.c -o hello
```

Προεπεξεργαστής (Preprocessor)

Ο προεπεξεργαστής (preprocessor) είναι ένα υποσύστημα του μεταγλωττιστή.



Προεπεξεργαστής (Preprocessor)

Μπορούμε να δούμε την έξοδο του προεπεξεργαστή με το -E (ή τρέχοντας το πρόγραμμα του προεπεξεργαστή το ίδιο - cpp):


```
$ gcc -E hello.c -o processed.c
```

ή

```
$ cpp hello.c -o processed.c
```

Εντολές Προεπεξεργαστή (Preprocessor Directives)

Όλες οι εντολές στον προεπεξεργαστή ξεκινάνε με το σύμβολο #. Ενδεικτικά:

1. `#include`
2. `#define`
3. `#if #else #elif #endif`
4. `#ifdef #ifndef`

Η εντολή #include

Η εντολή `#include <file.h>` εισάγει τα περιεχόμενα του αρχείου `file.h` στο σημείο που γράφτηκε στο πρόγραμμα.

Που βρίσκονται αυτά τα αρχεία; Σε προκαθορισμένους φακέλους στο λειτουργικό σας ή σε φακέλους που προσδιορίζονται με το όρισμα **`-I`** του gcc

Δεύτερη εκδοχή: `#include "file.h"` - σε αυτήν την περίπτωση ο μεταγλωττιστής ψάχνει πρώτα στον φάκελο που βρίσκεται το πηγαίο αρχείο.

Η εντολή #define

Η εντολή `#define` μας επιτρέπει να ορίσουμε μακροεντολές (macros)

1. Ορισμός σταθεράς:

```
#define TRUE 1
```

2. Ορισμός υπολογισμού:

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))
```

Η μακροεντολή αντικαθίσταται στον κώδικα πριν την μεταγλώττιση

Τι θα επιστρέψει το παρακάτω πρόγραμμα;

```
#define PROD 2*5
int main() {
  return 20 / PROD;
}
```

Η μακροεντολή αντικαθίσταται στον κώδικα πριν την μεταγλώττιση

Τι θα επιστρέψει το παρακάτω πρόγραμμα;

```
#define PROD 2*5

int main() {

  return 20 / PROD;

}
```

```
$ cpp prod.c
# 0 "prod.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3
# 0 "<command-line>" 2
# 1 "prod.c"


int main() {
  return 20 / 2*5;
}
$ gcc -o prod prod.c
$ ./prod
$ echo $?
50
```

Η τιμή της μακροεντολής μπορεί να περαστεί από την γραμμή εντολών

Χρησιμοποιούμε την σύνταξη -DMACRO=VALUE όπου MACRO η μακροεντολή και VALUE η τιμή που θέλουμε να δώσουμε. Για παράδειγμα:

```
int main() {

  return 20 / PROD;

}
```

```
$ gcc -DPROD=10 -o prod prod.c
$ ./prod
$ echo $?
2
```

Η εντολή #if #else #endif

Έχουν μορφή παρόμοια με την δομή ελέγχου if στην C αλλά δρουν στο επίπεδο του κώδικα. Σε τι θα προεπεξεργαστεί το ακόλουθο πρόγραμμα:

```
int main() {
#if 0
  return 42;
#else
  return 1;
#endif
}
```

Η εντολή #if #else #endif

Έχουν μορφή παρόμοια με την δομή ελέγχου if στην C αλλά δρουν στο επίπεδο του κώδικα. Σε τι θα προεπεξεργαστεί το ακόλουθο πρόγραμμα:

```
int main() {
#if 0
  return 42;
#else
  return 1;
#endif
}
```

```
$ gcc -E example.c
# 0 "example.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "example.c"
int main() {


  return 1;

}
```

Η εντολή #ifdef #ifndef

Με τις εντολές #ifdef / #ifndef μπορούμε να ελέγξουμε αν μια μακροεντολή έχει οριστεί ή όχι:

```
#define DEBUG
#ifdef DEBUG
  printf("debugging is on\n");
#else
  printf("debugging is off\n");
#endif
#ifndef DEBUG
  printf("optimizations are on\n");
#endif
```

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 154-159, 161.

- Big-O Notation
- C Preprocessor

Ευχαριστώ και καλό σαβ/κο εύχομαι!
Keep Coding ;)