

Εισαγωγή στον Προγραμματισμό (CEID_NY131)



Βασικές έννοιες Διαδικαστικού Προστακτικού Προγραμματισμού (2ο Μέρος)

με τη χρήση παραδειγμάτων
Υπολογιστικών διεργασιών

Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
<https://sites.google.com/site/thrambouliskleanthis/>



Βασικές έννοιες Προστακτικού Προγραμματισμού

Imperative programming
Procedural Abstraction Data Abstraction
Function Selection Statement Variable
Statement Data Type
Loop Statement
...τίποτε παραπάνω από αντιγραφή
βασικών εννοιών από την καθημερινή ζωή

Προγραμματισμός – Βασικές έννοιες

- Διεργασία – **Συνάρτηση (Function)**
- Αλγόριθμος – Περιγραφή Διεργασίας
- Λεκτική περιγραφή – Ορίζει τον Αλγόριθμο
- Πρόγραμμα (Πηγαίο – Εκτελέσιμο)
- Μεταγλωττιστής (Compiler)
- IDE (Integrated Development Environment)
- Αφαιρετικότητα (Abstraction)
 - Στις διεργασίες (procedural abstraction)
 - Στα δεδομένα (data abstraction)



Αλγόριθμος: ένα σύνολο βημάτων τα οποία καθορίζουν τον τρόπο εκτέλεσης μίας εργασίας.

Οργάνωση Διάλεξης - Αφαιρετικότητα στις Διεργασίες

- **Αφαιρετικότητα στις Διεργασίες**
 - Ορισμός - Είδη
 - Στοιχεία Διεργασίας
 - Αξιοποίηση Αφαιρετικότητας στον προγραμματισμό (Add2Numbers Version 2)
 - Η συνάρτηση sum (Add2Numbers Version 3)
- Ανάθεση έργου - Παραδείγματα
 - Η Hailstone sequence (Λεκτική περιγραφή)
 - Η επικοινωνία με την μηχανή
 - Fibonacci numbers

Αφαιρετικότητα

- μια απλοποιημένη περιγραφή ή τεκμηρίωση που δίνει έμφαση σε ορισμένα χαρακτηριστικά ενώ ταυτόχρονα αποσιώπα άλλα.
- μια από τις βασικότερες τεχνικές με την οποία ο άνθρωπος **αντιμετωπίζει την πολυπλοκότητα**.

Είδη αφαιρετικότητας

- αφαιρετικότητα στις διεργασίες (**Procedural abstraction**)
 - "φτιάξε ένα καφέ"
 - «πάρε με τηλέφωνο»
- αφαιρετικότητα στα δεδομένα (**data abstraction**)
 - Δώσε μου ένα "βιογραφικό"
 - Δώσε μου την "αναλυτική καρτέλα σου"

Αφαιρετικότητα στις διεργασίες

- επιτρέπει στο χρήστη να **εστιάσει τη προσοχή** του
 - στο **τι** γίνεται και όχι
 - στο **πώς** αυτό γίνεται
- επιτυγχάνεται ορίζοντας δύο τμήματα για μια διεργασία
 - τον τρόπο αναφοράς στην διεργασία - **interface**
 - την υλοποίηση της διεργασίας - **implementation**



Θα **γράψεις** 500 φορές «Δεν θα πετάω αεροπλανάκια στην τάξη»



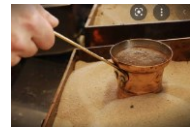
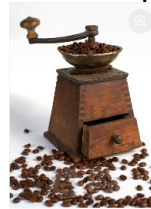
Η διεργασία παρασκευής καφέ

Ένα Ελληνικό
μέτριο
παρακαλώ



- Είναι μια **σύνθετη** (παραγόμενη) **διεργασία** που η υλοποίηση της δίνεται με την χρήση άλλων διεργασιών

- Άλεσε τους κόκκους του καφέ
- Ανάμειξε τα συνθετικά του καφέ
- Βράσε το μείγμα
- ...



Η διεργασία παρασκευής καφέ (συνθετικά)



- Η διεργασία έχει :

- **Όνομα** (Ορίζει το **ΤΙ κάνει** η διεργασία)
- **Περιγραφή** (Ορίζει το **ΠΩΣ το κάνει**)
- Δέχεται **εισόδους** (τις οποίες επεξεργάζεται)
- Παράγει **έξοδο/δους** (τις οποίες διαμορφώνει με βάση τις εισόδους)

Παράδειγμα εφαρμογής Αφαιρετικότητας (Add2Numbers)

Add2Numbers –Λεκτική περιγραφή 1^η εκδοχή

1. Ζήτα τον πρώτο αριθμό από τον χρήστη
2. Πάρε τον πρώτο αριθμό από τον χρήστη
3. Ζήτα τον δεύτερο αριθμό από τον χρήστη
4. Πάρε τον δεύτερο αριθμό από τον χρήστη
5. Υπολόγισε το άθροισμα των δύο αριθμών
6. Εμφάνισε το άθροισμα

printf(...)

scanf(...)

Add2Numbers –Λεκτική περιγραφή 2^η εκδοχή

1. Πάρε τον πρώτο αριθμό από τον χρήστη
2. Πάρε τον δεύτερο αριθμό από τον χρήστη
3. Υπολόγισε το άθροισμα των δύο αριθμών
4. Εμφάνισε το άθροισμα

getNumber(...)

Συνάρτηση ορισμένη από
τον προγραμματιστή

Πηγαίος κώδικας με την getNumber()

Add2Numbers –Λεκτική περιγραφή 2^η εκδοχή

1. Πάρε τον πρώτο αριθμό από τον χρήστη
2. Πάρε τον δεύτερο αριθμό από τον χρήστη
3. Υπολόγισε το άθροισμα των δύο αριθμών
4. Εμφάνισε το άθροισμα

getNumber(...)

getNumber(...)

num1 + num2;

printf(...)

Η getNumber **πρέπει να ζητάει** από τον χρήστη ένα αριθμό, **να** τον **διαβάξει** και **να** τον **επιστρέφει** στην συνάρτηση που την κάλεσε.

Προϋποθέσεις χρήσης:

1. Να έχει **δηλωθεί** η συνάρτηση
Αυτό γίνεται με μια **πρόταση δήλωσης συνάρτησης**, η οποία ορίζει το όνομα της, τα ορίσματα που δέχεται και την επιστρεφόμενη τιμή.
2. Να έχει **οριστεί** η συνάρτηση
Αυτό γίνεται ορίζοντας το **σώμα της συνάρτησης** με τρόπο που να α) αξιοποιεί τα ορίσματα που δέχεται, b) να εκτελεί την διεργασία για την οποία δημιουργείται και c) να επιστρέφει το αποτέλεσμα της διεργασίας.

getNumber() - Η δήλωση συνάρτησης

```
6 int getNumber(void); // protasi dilwsis Synartisis
7
8 int num1,num2; // protasi dilwsis genikis (global) me
9 int res;
```

■ Η δήλωση της συνάρτησης

- Ορίζει
 - Το όνομα της συνάρτησης (getNumber)
 - Τα ονόματα και τους τύπους των ορισμάτων που αυτή δέχεται (void)
 - Τον τύπο της επιστρεφόμενης τιμής (int)
- τοποθετείται πριν τη δήλωση των γενικών μεταβλητών
- είναι απαραίτητη για την κλήση της συνάρτησης και τον έλεγχο τύπων (type checking)

void: η λέξη σημαίνει πως η συνάρτηση δεν δέχεται ορίσματα

getNumber() - Ο ορισμός της συνάρτησης

```
21 int getNumber(void){ // Orismos Synartisis
22     int num;
23     printf("Dose arithmo:");
24     scanf("%d",&num); //klisi synartisis scanf
25     return num; //epistrefei tin timi tis num
26 }
```

Ο ορισμός της συνάρτησης

- **Ορίζει** το σώμα της συνάρτησης (lines 22-25)
 - Αυτό πρέπει να υλοποιεί τη διεργασία που το όνομα της δηλώνει. Πρέπει να ζητάει από τον χρήστη ένα αριθμό (με την printf -line 23), να τον διαβάζει (με την scanf -line 24) και τον επιστρέφει (με την πρόταση return -line 25) στην συνάρτηση που την κάλεσε (η οποία στην περίπτωση μας είναι η main).
- **Επιστρέφει** τιμή, του τύπου που ορίζει η δήλωση της συνάρτησης (**int**)
- Η 1^η πρόταση ορισμού της συνάρτησης (line 21) πρέπει να είναι σύμφωνη με την δήλωση της.

getNumber() - Ο κλήση της συνάρτησης

```
13 num1 = getNumber(); // klisi synartisis kai apodosi epistrefomenis
14 //printf("o 1os arithmos einai: %d\n",num1);
15 num2 = getNumber();
```

Η κλήση της συνάρτησης

- Γίνεται με το όνομα της ακολουθούμενο από τις τιμές που περνάμε στα ορίσματα της (getNumber())
- Αν επιστρέφει τιμή αυτή πρέπει να ανατεθεί σε μία μεταβλητή (num1 = getNumber()) ή να αξιοποιηθεί στο σχηματισμό μιας έκφρασης, π.χ.
getNumber() + getNumber()

Ο compiler ελέγχει αν η κλήση γίνεται σύμφωνα με την δήλωση της (έλεγχος τύπων)

Δοκιμάστε την πρόταση

```
res = getNumber() + getNumber();
```

Η συνάρτηση sum() (εκδοχή 1^η)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void sum(void); // protasi dilwsis synartisis
5
6 int num1,num2; // protasi dilwsis genikis (global) metavlitis
7 int result;
8
9 int main(int argc, char *argv[]) {
10     printf("Programa prosthesis 2 arithmwv me synartisi (V1)\n");
11     printf("Dose ton 1o arithmo:");
12     scanf("%d",&num1);
13     //printf("o 1os arithmos einai: %d\n",num1);
14     printf("Dose ton 2o arithmo:");
15     scanf("%d",&num2);
16     //printf("o 2os arithmos einai: %d\n",num2);
17     sum(); // protasi klisis synartisis
18     printf("to athrisma twv %d kai %d einai: %d\n",num1,num2,result);
19     return 0;
20 }
21
22 void sum(void){ //orismos synartisis
23     result=num1+num2;
24 }
```

Δήλωση συνάρτησης

Κλήση συνάρτησης

Ορισμός συνάρτησης

Η συνάρτηση sum **χρησιμοποιεί** τις **γενικές μεταβλητές** num1, num2 και result οπότε **δεν είναι επαναχρησιμοποιήσιμη (reusable)**

Η συνάρτηση sum() (εκδοχή 2^η)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int sum(int n1, int n2); // protasi dilwsis synartisis
5 //int num1,num2; // protasi dilwsis genikis (global) metavlitis
6 //int result;
7 int main(int argc, char *argv[]) {
8     int num1,num2; // protasi dilwsis topikis (local) metavlitis
9     int result;
10
11     printf("Programa prosthesis 2 arithmwv me synartisi (V1)\n");
12     printf("Dose ton 1o arithmo:");
13     scanf("%d",&num1);
14     printf("Dose ton 2o arithmo:");
15     scanf("%d",&num2);
16     result = sum(num1,num2); // protasi klisis synartisis
17     printf("to athrisma twv %d kai %d einai: %d\n",num1,num2,result);
18     return 0;
19 }
20
21 int sum(int n1, int n2){ //orismos synartisis
22     int result;
23     result=n1+n2;
24     return result;
25 }
```

Δήλωση συνάρτησης

Τοπικές μεταβλητές

Κλήση συνάρτησης

Ορισμός συνάρτησης

Η συνάρτηση sum **δεν χρησιμοποιεί γενικές μεταβλητές** οπότε **είναι επαναχρησιμοποιήσιμη (reusable)**

Η συνάρτηση sum() (εκδοχή 2^η)

Δήλωση συνάρτησης

```
4 int sum(int n1, int n2);
```

Κλήση συνάρτησης

```
16 result = sum(num1,num2);
```

Ορισμός συνάρτησης

```
21 int sum(int n1, int n2){
22     int result;
23     result=n1+n2;
24     return result;
25 }
```

Τα βασικά για συναρτήσεις βασικής βιβλιοθήκης θα βρείτε στο **I2P_Functions** (slides 4-8)

Οργάνωση Διάλεξης - Ανάθεση έργου - Παραδείγματα

- Αφαιρετικότητα
 - Ορισμός - Είδη
 - Στοιχεία Διεργασίας
- **Ανάθεση έργου - Παραδείγματα**
 - **Η Hailstone sequence (Λεκτική περιγραφή)**
 - Η επικοινωνία με την μηχανή
 - Fibonacci numbers

Ανάθεση έργου (hailstone sequence)



Hailstone sequence

The Collatz problem: *one of the world's unsolved problems in mathematics.*

It involves a sequence of numbers called a Hailstone sequence. It is called this because the numbers go up and down again.

*The conjecture is that **these sequences always reach 1**, no matter which positive integer is chosen to start the sequence.*



Hailstone Sequences **follow these rules:**

- If a number is even, divide it by 2
- If a number is odd, multiply it by 3 and add 1

For 25: 25 – 76 – 38 – 19 – 58 – 29 – 88 – 44

...



Hailstone sequence -Λεκτική περιγραφή Version 1

■ Το έργο

Γράψε μας την **hailstone sequence** για τον αριθμό 25

■ Ανάθεση σε άνθρωπο. **Πώς;**

- A1. Πάρε έναν θετικό ακέραιο αριθμό
- A2. Εάν ο αριθμός είναι 1
τερμάτισε
- A3. Εάν ο αριθμός είναι άρτιος
τότε διαίρεσε τον με το 2
διαφορετικά
πολλαπλασίασε τον με το 3 και
πρόσθεσε 1
- A4. Τύπωσε τον αριθμό
- A5. Επανέλαβε από το βήμα A2.

1η έκδοχή

Hailstone sequence - Ψευδοκώδικας

- Δεν αποτελεί λεκτική περιγραφή!!!

```
A1. [Input n] Πάρε έναν ακέραιο αριθμό
A2. [Termination] If n = 1
    exit
A3. [Check n] If n is even
    n := n / 2
    else
    n := n * 3 + 1
A4. [Print n] Τύπωσε τον n
A5. [Next number] Go to A2.
```

Η χρήση του **go to** καταστρέφει τη δόμηση του κώδικα

Hailstone sequence -Λεκτική περιγραφή Version 2

A.1. Ζήτη ένα θετικό ακέραιο αριθμό

A.2 Πάρε τον αριθμό

A.3 Επανάλαβε τα παρακάτω

A3.1 Αν ο αριθμός είναι 1
τερμάτισε

A3.2 Αν ο αριθμός είναι άρτιος
Διαίρεσε τον με το 2

Διαφορετικά

Πολλαπλασίασε τον με το 3 και
πρόσθεσε το 1

A3.3 Τύπωσε τον αριθμό

A.3: Loop Statement

A3.1: Decision Statement

A3.2: Selection Statement

2^η έκδοχή

Η πρόταση επανάληψης δεν έχει συνθήκη τερματισμού

Hailstone sequence -Λεκτική περιγραφή Version 3

- A.1. Ζήτα ένα θετικό ακέραιο αριθμό*
A.2. Πάρε τον αριθμό
*A.3. **Επανάλαβε μέχρι ο αριθμός να γίνει 1***
A3.1 Αν ο αριθμός είναι άρτιος
Διαίρεσε τον με το 2
Διαφορετικά
Πολλαπλασίασε τον με το 3 και
πρόσθεσε το 1
A3.2 Τύπωσε τον αριθμό

3^η εκδοχή