ΗΥ352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

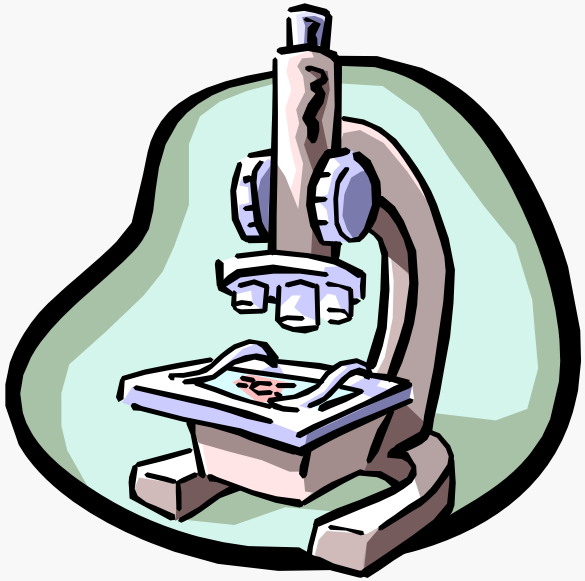
ΔΙΔΑΣΚΩΝ

Αντώνιος Σαββίδης

ΕΝΟΤΗΤΑ 6

ΟΔΗΓΙΕΣ ΚΑΛΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αριθμός διαλέξεων 1 – Διάλεξη 1η



Περιεχόμενα

- *Οργάνωση και εσωτερική δομή αρχείων*
- Πολιτικές μεγεθών και ονομασίας
- Κενά, γραμμές, και σχόλια
- Εποπτεία, έλεγχος και backups του κώδικα

Οργάνωση και εσωτερική δομή αρχείων (1/6)

1. Διάλεξε κατάλληλη πολιτική ονομασίας για: υποσύστημα, τμήμα, υποτμήμα, πριν αρχίσεις να δημιουργείς αρχεία και directories
2. Οργάνωσε τα αρχεία με directories ανά υποσύστημα και ονόματα από την αρχιτεκτονική
3. Ονόμασε τα αρχεία με κατάλληλη περιγραφική σημασιολογία (δεν φοβόμαστε τα μεγάλα ονόματα) μη βάλετε ως πρόθεμα στα αρχεία το αναγνωριστικό όνομα η συντομογραφία του subsystem
4. Εάν υπάρχουν file name conflicts τότε κάνουμε include ένα folder πίσω

Subsystem AIUnits για σε war simulator

```
AIUnits/Soldier.h
AIUnits/Soldier.cpp
AIUnits/Commander.h
AIUnits/Commander.cpp
#include “Soldier.h”                ← no file names conflicts
#include “AIUnits/Soldier.h”        ← has file name conflicts
```

Οργάνωση και εσωτερική δομή αρχείων (2/6)

4. Μία βασική κλάση ανά αρχείο (μαζί με τις embedded classes). Ονομασία αρχείων με το στυλ *<class>.h* και *<class>.cpp* αντίστοιχα.
   - Αντί για *.h*, *.cpp* μπορεί, ανάλογα με τον compiler, να έχουμε και άλλα file extensions, π.χ., *.hpp* ή *.cc* ή *h++* ή *c++*
5. Κρύψε τα private και protected μέλη μεγάλων κλάσεων, οι οποίες πρόκειται να χρησιμοποιηθούν από άλλους προγραμματιστές, όταν αυτές παγιωθούν, σε επιπλέον headers με ονομασίες *<class>_Private.h* και *<class>_Protected.h,* με απ ευθείας #include στο σώμα της κλάσης
6. Κρύψε σε επιπλέον headers όλους τους επιπρόσθετους τύπους, κλάσεις, συναρτήσεις, templates, κλπ, όταν δεν πρέπει να είναι ορατοί παρά μόνο σε μία συγκεκριμένη κλάση
   - Όμως τύποι που έχουν νόημα ύπαρξης μόνο στα πλαίσια συγκεκριμένης κλάσης πρέπει να ορίζονται πάντα μέσα σε αυτή

CSD

Οργάνωση και εσωτερική δομή αρχείων (3/6)

```
// MyClass.h
class MyClass {
 private:
 #include “MyClass_Private.h”
 protected:
 #include “MyClass_Protected.h”
 public: ...
};
```

Εάν ο τύπος *MyType* χρησιμοποιείται αποκλειστικά από την *MyClass*, ο παρακάτω τρόπος δεν είναι βέλτιστος.

```
// MyClass_MyType.h
struct MyType { ... };

// MyClass.cpp
#include “MyClass.h”
#include “MyType.h”
```

```
// MyClass.h
class MyClass {
  Εάν μπλέκεται μόνο σε private μέλη:
  private:
  struct MyType { ... };

   Εάν εμπλέκεται σε public member signature:
  public:
  struct MyType { ... };
};
```

CSD

Οργάνωση και εσωτερική δομή αρχείων (4/6)

7. `#ifndef - #define - #endif` σε όλα τα headers.
8. Λογική κατάτμηση των αρχείων με κατάλληλα περιγραφικά σχόλια
   - **κεφάλαια (π.χ. διαφορετικά κεφάλαια μπορεί να είναι: γενική περιγραφή, include files, static data, τοπικά utility macros ή συναρτήσεις, και υλοποίηση κλάσεων),**
   - **ενότητα (υλοποίηση μελών κλάσεων – μπορεί να ομαδοποιούνται σε μία ενότητα πολλές συγγενείς συναρτήσεις),**
   - **παράγραφος (τμήματα υλοποίησης ειδικών μελών, π.χ., συναρτήσεις ενός dispatch table)**
9. Σημειώστε με ***TODO*** και ***FIXME*** comments τα σημεία στον κώδικα όπου λείπει υλοποίηση ή θέλουν διόρθωση με κατάλληλα σχόλια σχετικά με το τι πρέπει να συμπληρωθεί
10. Σε include / import να έχετε ξεχωριστές ομάδες (standard library, third party, own libraries)
11. Τα headers πρέπει να ορίζονται έτσι ώστε η σειρά του **`#include`** να μην παίζει κανένα ρόλο; για να γίνει αυτό, θα πρέπει κάθε header να μπορεί να κάνει compile επιτυχώς από μόνο του. (δηλ. να κάνει ήδη include ότι χρειάζεται)
12. Δεν αφαιρούμε από την λίστα των included headers κάποια, επειδή υποθέτουμε ότι ήδη είναι included από κάποιο header του **`#include`** list.

Οργάνωση και εσωτερική δομή αρχείων (5/6)

```
// MyClass.h
#ifndef MYCLASS_H
#define MYCLASS_H
■Κάνουμε #include μόνο ότι χρειάζεται το"MyClass.h" για να κάνει compile.
■Βάζουμε εδώ ότι ανήκει φυσιολογικά στο "MyClass.h".
#endif // Header end point.

// MyClass.cpp
//-----------------------------------// Διαχωριστής κεφαλαίων.
////////////////////////////////////// Διαχωριστής ενοτήτων.
//******************                  Διαχωριστής παραγράφων.

#include <stdio.h>
#include <string.h>
#include <list>
#include "MyClass.h"
#include "AnotherClass.h"
```

Οργάνωση και εσωτερική δομή αρχείων (6/6)

```
// MyClass.cpp
#include “header1.h”
#include “header2.h”
```

- Εάν το header1.h πρέπει να γίνει `#include` πριν το header2.h, σημαίνει ότι υπάρχουν ορισμοί από το header1.h που χρησιμοποιούνται από το header2.h.
- Για να αποφύγουμε να θυμόμαστε τέτοιες εξαρτήσεις αρκεί να βάλουμε ένα #include “header1.h” στο header2.h.
- Με αυτό, και υποθέτοντας ότι τα `#ifndef - #define - #endif` υπάρχουν σωστά, οποιαδήποτε σειρά του inclusion είναι σωστή.

```
#include “header2.h”
```

- Εδώ, επειδή γνωρίζουμε ότι το header2.h κάνει include το header1.h, νομίζουμε ότι είναι πλεονάζον να κάνουμε και include το header1.h.
- Αυτό είναι λάθος. Πρέπει να βάλουμε ένα `#include` για το header1.h, αφού ούτε η ταχύτητα compilation επηρεάζεται, ενώ οι εξαρτήσεις φαίνονται καλύτερα στον κώδικα.
- Επίσης, μπορεί κάποια στιγμή το header1.h να μην γίνεται included από το header2.h, και να προκαλούσαμε compile error.

Περιεχόμενα

- Οργάνωση και εσωτερική δομή αρχείων
- *Πολιτικές μεγεθών και ονομασίας*
- Κενά, γραμμές, και σχόλια
- Εποπτεία, έλεγχος και backups του κώδικα

CSD

Πολιτικές μεγεθών και ονομασίας (1/2)

1. Αποφύγετε υλοποιήσεις συναρτήσεων με περισσότερες των 20 εντολών, ενώ προτιμήστε οι περισσότερες συναρτήσεις να έχουν λιγότερο από 10 εντολές
2. Αποφύγετε αρχεία με πλέον των 1000 LOC. Σε τέτοιες περιπτώσεις τεμαχίζουμε την υλοποίηση σε περισσότερα αρχεία, βάζοντας λογικά συγγενή τμήματα κώδικα μαζί.
3. Ελέγξτε καλά τα πολλαπλά φωλιασμένα loops (με βάθος ≥ 3).
4. Ελέγξτε καλά τις πολλαπλά φωλιασμένες εντολές εκτέλεσης υπό συνθήκης (με βάθος ≥ 3).
5. Πάντα να «δουλεύετε» λίγο περισσότερο σε σε πολύπλοκα loops ή εντολής εκτέλεσης υπό συνθήκη για να πετύχετε βέλτιστη και διαυγή κατάτμηση και υλοποίηση.
6. Πάντα να δουλεύετε σε εντολές εκτέλεσης υπό συνθήκη με πολύπλοκες λογικές εκφράσεις – συνήθως η πρώτη υλοποίηση είναι η χειρότερη.

Πολιτικές μεγεθών και ονομασίας (2/2)

7. Ελέγξτε καλά τις συναρτήσεις που έχουν πλέον των τεσσάρων παραμέτρων.
8. Ονομάστε παρόμοιες συναρτήσεις με κοινά αποδεκτά αναγνωριστικά. Αποφύγετε την χρήση διαφορετικών ονομάτων για την ίδια «δουλειά»:
   - ***Write / Store / Save / Dump***
   - ***Copy / Create / Make / Produce / Instantiate***
9. Χρησιμοποιήστε περιγραφικά ονόματα, ακολουθώντας μία τυποποιημένη και συνεπή πολιτική στο πρόγραμμά σας (π.χ. κεφαλαίο το πρώτο γράμμα κάθε λέξης σε ονομασία συναρτήσεων – **`FunctionName`**, χρήση κεφαλαίων με διαχωρισμό λέξεων μέσω του underscore για σταθερές – **`CONST_NAME`**, κεφαλαίο το πρώτο γράμμα κάθε λέξης, πλην της πρώτης, για μεταβλητές - **`varName`**).
10. Δηλώστε τις τοπικές μεταβλητές όσο το δυνατόν πλησιέστερα του σημείου χρήσης.
11. Εάν η αναγνωσιμότητα του κώδικα δεν επηρεάζεται, μπορείτε να επαναχρησιμοποιήσετε τις τοπικές μεταβλητές (αυτό θα χρειαστεί σε πολύ ειδικές περιπτώσεις απαιτήσεων απόδοσης – αλλιώς ξεχάστε το).

Περιεχόμενα

- Οργάνωση και εσωτερική δομή αρχείων
- Πολιτικές μεγεθών και ονομασίας
- *Κενά, γραμμές, και σχόλια*
- Εποπτεία, έλεγχος και backups του κώδικα

Κενά, γραμμές και σχόλια (1/3)

1. Αφήνουμε ένα κενό μεταξύ των διαφορετικών πραγματικών ορισμάτων στις κλήσεις συναρτήσεων. Εάν εμπλέκονται μεγάλες εκφράσεις, τις στοιχίζουμε σε διαφορετικές γραμμές.
2. Αφήνουμε κενά μεταξύ των τελεστών και των ορισμάτων τους. Εάν δεν θυμάστε τους κανόνες προτεραιότητας, βάλτε παρενθέσεις για να επιβάλετε την προτεραιότητα.
3. Ακολουθήστε μία σταθερή πολιτική για την στοίχιση κώδικα και την εισαγωγή tabs, η οποία σας επιτρέπει καλό οπτικό έλεγχο και εποπτεία.
4. Αποφύγετε τη χρήση της κενής εντολής `;` - προτιμήστε το κενό block `{}`.
5. Τα σημεία εξόδου από μία συνάρτηση να είναι ευδιάκριτα. Πιστοποιήστε και ελέγξτε την σωστή έξοδο από τη συνάρτηση.
6. Στις συναρτήσεις με πολλά σημεία εξόδου να αποφεύγετε τα fallbacks

Κενά, γραμμές και σχόλια (2/3)

6. Σύντομα σχόλια για τη χρήση μίας κλάσης στο header file. Παρέχουμε πληροφορία χρήσης (και τυχόν περιορισμούς) στο public API της κλάσης. Διαχωρίζουμε σε λογικά σύνολα τα members και βάζουμε σύντομα σχόλια για το καθένα.
7. Βάζουμε σχόλια για λεπτομέρειες υλοποίησης στο *.cpp* αρχείο. Δεν αφήνουμε σκιώδεις λεπτομέρειες υλοποίησης χωρίς τεκμηρίωση με σχόλια, γιατί αυτές ξεχνιούνται εύκολα.
8. Σύντομη τεκμηρίωση κάθε συνάρτησης, κυρίως για ότι δεν είναι πραγματικά προφανές: περιγραφή ορισμάτων, τι επεξεργασία κάνει, τι επιστρέφει, πως χρησιμοποιείται (εάν έχει νόημα) το αποτέλεσμα, συμπεριφορά σε λάθος παραμέτρους, κλπ.
9. Τεμαχίστε και τεκμηριώστε τα διάφορα λογικά επεξεργαστικά βήματα κάθε συνάρτησης.
10. Συνολικά σχόλια στην αρχή του αρχείου. Ποιος είναι ο προγραμματιστής, λίστα με TODO, FXIME, revision history.

Κενά, γραμμές και σχόλια (3/3)

παράδειγμα

```
a*b+d*c-*e->f;
a * b - d * c - (*(e->f))
int f(int, int);            ← Σίγουρα χρειάζεται επεξήγηση.
int add (int x, int y);     ← Αυτό –τεκμηρίωση με καλή χρήση αναγνωριστικών.
X* X::Make (const X&);      ← Γιατί δεν είναι const ? Είναι δυναμικό το result ?
// Αυτή η συνάρτηση θα αλλάξει.  ← Πότε και γιατί ? είναι σωστή τώρα ?

if (      !(r = Parse_ExprMulExpr(e))   &&  ←Χρήση κενών και γραμμών για ευκρίνεια.
          !(r = Parse_ExprPlusExpr(e))  &&
          !(r = Parse_ExprSubExpr(e))   &&
          !(r = Parse_ExprDivExpr(e))   ) {...}

sprintf(    ← Κλήση όταν τα ορίσματα λαμβάνουν αρκετό χώρο κειμένου.
   errorText,
   "Parse error: %s, at row %d, col %d.\n",
   error,
   currToken.row,
   currToken.col
);
```

Περιεχόμενα

- Οργάνωση και εσωτερική δομή αρχείων
- Πολιτικές μεγεθών και ονομασίας
- Κενά, γραμμές, και σχόλια
- *Εποπτεία, έλεγχος και backups του κώδικα*

Εποπτεία, έλεγχος και backups (1/2)

- Η χρήση ενός source control system *επιβάλλεται*
- Διατηρούμε ένα ***σημειωματάριο σχεδίασης*** (ηλεκτρονικό κατά προτίμηση ή χειρόγραφο) για κάθε υποσύστημα
  - Νέες ιδέες, τεχνικές σχεδίασης, ψευδοκώδικα, αρχιτεκτονική, χρονοδιάγραμμα, αλγοριθμική σχεδίαση, κώδικα, κλπ
  - Χρησιμοποιήστε το ως υλικό τεκμηρίωσης και ελέγχου της ανάπτυξης
  - Γράφουμε πάντα που σταματάμε και ποια είναι τα επόμενα προγραμματιστικά βήματα κάθε μέρα (δηλ. σώνουμε την προγραμματιστική κατάσταση)
  - Εάν διακόψετε για χρόνο πλέον της εβδομάδας, θα είναι πολύτιμη πληροφορία για επανέναρξη
- Ένα TODO file είναι *υποχρεωτικό* με λεπτομερείς οδηγίες, priorities, pseudo code και κρισιμότητα για τα επόμενα βήματα
  - Ένα σωστό TODO file δεν απαιτεί τίποτε άλλο παρά να το διαβάσετε

Εποπτεία, έλεγχος και backups (2/2)