Redactarea codului sursă

- redactarea codului nu presupune doar scrierea unui text corect ca sintaxă

- contează și aspectele legate de
  - modalitatea de redactare
  - stilul de formatare
  - aranjarea codului în pagină
  - adăugarea unor comentarii utile

Redactarea codului sursă

- scopuri:

- **Îmbunătățirea lizibilității codului**
- **Asigurarea ușurinței depistării vizuale a erorilor (în special a celor referitoare la logica programului)**
- **Asigurarea consecvenței în redactare (același stil și format pentru întregul cod)**
- **Colaborarea eficientă cu alții (colaboratorilor le va fi mai ușor să interpreteze propriul nostru cod)**
- **Îmbunătățirea vitezei de scriere a codului (dacă ne obișnuim cu un stil de redactare eficient)**
- **Minimizarea comentariilor**

Redactarea codului sursă

- **Majoritatea companiilor de profil lucrează după un standard impus**

- **Majoritatea echipelor de programatori respectă anumite convenții de scriere a codului**

- **Standardul și convențiile conțin reguli de redactare, formatare și comentare a codului**

Exemplu: convenții impuse de compania *id Software* *(Quake, Doom, etc.)*

Use real tabs that equal 4 spaces.

Use typically trailing braces everywhere (if, else, functions, structures, typedefs, class definitions, etc.)

```
if ( x ) {
}
```

The else statement starts on the same line as the last closing brace.

```
if ( x ) {
} else {
}
```

Exemplu: convenții impuse de compania *id Software* *(Quake, Doom, etc.)*

Function names start with an upper case:

```
void Function( void );
```

Variable names start with a lower case character.

```
float x;
```

Typedef names use the same naming convention as variables, however they always end with "_t".

```
typedef int fileHandle_t;
```

Exemplu: convenții impuse de compania *id Software* *(Quake, Doom, etc.)*

Indent the names of class variables and class methods to make nice columns. The variable type or method return type is in the first column and the variable name or method name is in the second column.

```
class idVec3 {
float           x;
float           y;
float           z;
float           Length( void ) const;
const float *   ToFloatPtr( void ) const;
}
```

Redactarea codului sursă

- **Indiferent de standardul ales, cea mai importantă este consecvența**
  - **aplicarea aceloraşi convenții în întregul cod sursă**
  - **schimbarea bruscă a stilului de lucru creează confuzie, scade ușurința cu care se paote interpreta codul**

Redactarea codului sursă

- Există o serie de elemente de stil comune majorității standardelor

- De exemplu, aproape întotdeauna codul sursă se indentează:
  - Fiecare linie de cod se ocupă propria linie de text
  - Fiecare bloc de cod {…} la un tab (sau câteva spații) distanță de sursa în care este încorporat

Exemplu:

```
void sort(int arr[], int left, int right)
{
int i = left, j = right;
int tmp;
int pivot = arr[(left + right) / 2];

while (i <= j)
{
while (arr[i] < pivot)
i++;

while (arr[j] > pivot)
j--;

if (i <= j)
{
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
i++;
j--;
}
}

if (left < j)
sort(arr, left, j);
if (i < right)
sort(arr, i, right);
}
```

Fără indentare

- dificil de identificat blocurile de cod distincte
- dificil de identificat codul aferent diverselor instucțiuni (if, while etc.)
- poziționarea elementelor de sintaxă pereche (eg. acolade) dificil de perceput
- per total, codul este greu de interpretat vizual
  - necesită urmărirea cu atenție a diverselor simboluri și identificatori

Exemplu:

```
void sort(int arr[], int left, int right)
{
   int i = left, j = right;
   int tmp;
   int pivot = arr[(left + right) / 2];

   while (i <= j)
   {
      while (arr[i] < pivot)
         i++;

      while (arr[j] > pivot)
         j--;

      if (i <= j)
      {
         tmp = arr[i];
         arr[i] = arr[j];
         arr[j] = tmp;
         i++;
         j--;
      }
   }

   if (left < j)
      sort(arr, left, j);
   if (i < right)
      sort(arr, i, right);
}
```

Cu indentare

- blocurile de cod se observă clar
- este evident care cod se execută în cadrul instrucțiunilor de tip if , while
- se observă clar perechile de acolade
- elementele consitutente ale codului sunt mult mai ușor de identificat vizual

- **Scrierea unui cod corect și inteligibil presupune o proiectare adecvată a aplicației**

- **Proiectarea unei aplicații este o așa-numită “*wicked problem*”**
  - **o problemă care se clarifică abia atunci când se ajunge la o primă soluție**
  - **trebuie rezolvată din nou pentru a găsi o soluție mai bună**

- **Proiectarea unei aplicații**
  - **presupune ajungerea la un compromis:**
    - **consumul de resurse: memorie vs putere de calcul**
      - **se poate îmbunătăți viteza de calcul cu un consum mai mare de memorie**
      - **un consum mai mic de memorie presupune mai multe scrieri/citiri => performanță redusă**
    - **performanță vs siguranța datelor**
      - **se pot face verificări și testări exhaustive (de exemplu prin intermediul excepțiilor)**
      - **testările la runtime consumă resurse de calcul**

- **Proiectarea unei aplicații**
  - **presupune ajungerea la un compromis:**
    - **calitatea aplicației vs timpul alocat dezvoltării ei**
      - **anticiparea exhaustivă a bug-urilor și problemelor de design necesită timp și efort suplimentar**
    - **claritatea codului vs performanță**
      - **cod greu de interpretat, dar care este mai performant**
      - **exemple: aproape toate situațiile care presupun operații pe biți; aproape toate situațiile care presupun implementarea unei operații într-un limbaj low-level (assember)**

Exemplu:

```
a ^= b;
b ^= a;
a ^= b;
```

???

^ este operatorul be biți XOR

Exemplu:

```
a ^= b;
b ^= a;
a ^= b;
```

Interschimbare rapidă fără variabilă auxiliară (doar pentru integer)

- mai eficient decât alternativa binecunoscută, dar
- mai dificil de interpretat vizual (cel puțin la prima vedere)

Exemplu:

```
float func(float x)
{
   union
   {
      float x;
      int i;
   }u;

   u.x = x;
   u.i = 0x5f3759df - (u.i >> 1);
   return x*u.x*(1.5f - 0.5f*x*u.x*u.x);
}
```

???

Exemplu:

```
float func(float x)
{
   union
   {
      float x;
      int i;
   }u;

   u.x = x;
   u.i = 0x5f3759df - (u.i >> 1);
   return x*u.x*(1.5f - 0.5f*x*u.x*u.x);
}
```

Calculează cu aproximație sqrt(x), mai eficient decât funcția sqrt standard

- **Proiectarea unei aplicații**
  - **presupune impunerea de restricții**
    - **resurse de calcul finite**
    - **memorie insuficientă**
    - **timp limitat**
    - **capacitate de lucru limitată**
  - **este un proces nedeterminist**
    - **dacă două echipe diferite își propun să proiecteze soluția aceleiași probleme, probabil vor aborda design-uri diferite**
    - **mai multe modalități de a rezolva aceeași problemă**
      - **unele mai clare**
      - **altele mai eficiente**
      - **altele mai ușor de implementat**

- **Proiectarea unei aplicații**
  - **este un proces euristic**
    - **adesea se utilizează**
      - **principii și metode consacrate, dar netestate în situația actuală**
      - **reguli “de bun simț”**
      - **încercarea unor tehnici care uneori dau roade**
    - **implică o secvență de etape de încercare și eșec**
  - **este un proces emergent**
    - **design-ul nu se cunoaște în totalitate de la bun început**
    - **se extinde și se îmbunătățește pe măsură ce lucrul progresează**
    - **problemele se ameliorează pe baza experienței dobândite**

Redactarea codului – stabilirea denumirilor

- **declarațiile implicite sunt de evitat**
  - **ex: anumite compilatoare de C permiteau omiterea tipului de dată:**

```
n;
```
echivalent cu
```
int n = 0;
```

  - **valabil și pentru tipul de return al funcțiilor**

```
main()
{

   return 0;
}
```
echivalent cu
```
int main()
{

   return 0;
}
```

  - **compilatoarele moderne și competente generează erori sau, în cel mai rău caz, warnings**

Redactarea codului – stabilirea denumirilor

- **variabilele ar trebui declarate explicit**
  - **din viteză acest lucru se evită**
  - **problematic în cazul limbajelor care nu necesită specificarea tipului de dată la declarare (Visual Basic, Matlab etc.)**
  - **ex:**

```
clear;
clc;
im = imread('img.png');
a = edge(I);
```

scalar? matrice de întregi, nr reale, bool?
trebuie consultată documentația funcției edge, ceea ce necesită timp suplimentar și nu se poate în cazul în care codul este listat

Redactarea codului – stabilirea denumirilor

- variabilele ar trebui declarate explicit
  - ex 2: specificatorul *auto* introdus în standardul C++11

```
int main()
{
   float x = 2;
   auto a = func(x);
   return 0;
}
```

  - cum trebuie utilizat *a*?
  - care este tipul său de dată?
  - trebuie consultată sursa / documentația funcției apelate

Redactarea codului – stabilirea denumirilor

- este foarte importantă respectarea unor convenții de denumire
  - același stil pentru un scop anume
    - ex: pentru numărul de elemente ale unui vector vom folosi fie *nrElements*, fie *numElements*, fie *elementCount*, niciodată o combinație de denumiri în stiluri diferite în cadrul aceleiași aplicații
  - majoritatea compilatoarelor moderne permit listarea variabilelor
    - se pot depista “dublurile” (variabile denumite în stiluri diferite dar cu același rol)
    - se pot depista variabilele declarate și neutilizate

Redactarea codului – stabilirea denumirilor

- este foarte importantă respectarea unor convenții de denumire a variabilelor
  - denumiri explicite care sugerează rolul variabilei în cadrul aplicației

```
int n, m;
float *t;
```

???

vs

```
int nRows, nCols;
float *translationMatrix;
```

  - dar sunt de evitat denumirile exagerat de lungi

```
int numberOfRows, numberOfColumns;
float *translationMatrixForMainSceneGeometry;
```

Redactarea codului – stabilirea denumirilor

- **numele variabilelor, funcțiilor și metodelor trebuie să sugereze atât rolul lor, cât și să ofere indicii cu privire la posibilele lor valori**

- **ex1: numele unei variabile int ar trebui să sugereze valori întregi: nrElements, nrColumns, counter, etc.**

- **ex2: numele unei variabile bool ar trebui să sugereze valori de adevăr: *ok*, *found*, *done*, *success*.**
  - **denumirile *status*, *sourceFile* nu sunt adecvate deoarece nu trimit cu gândul la true, false. Mai potrivite ar fi *statusOK* sau *sourceFileAvailable***

Redactarea codului – stabilirea denumirilor

- **variabilele ar trebui, de cele mai mult ori, declarate cât mai aproape de porțiunea de cod unde sunt utilizate**

- **este utilă folosirea specificatorilor de tip *const* sau *final***
  - **indică faptul că variabila poate fi doar inițializată, nu și modificată pe parcurs**

- **se preferă limitarea în măsură cât mai mare a scopului variabilelor (domeniul de valabilitate)**
  - **variabilele se declară în cel mai mic segment de cod posibil**
  - **o sursă cu multe variabile globale este greu de analizat, corectat și depanat**

Redactarea codului – stabilirea denumirilor

- **fiecare variabilă trebuie să aibă doar un singur rol în cadrul aplicației**
  - **contraexemplu:**

```
int n;
// ... read n
float *vec = new float[n];
//... read elements of vec
n = 0;
while (vec[n] < threshold)
   n++;
```

***n* are dublu rol, se preferă folosirea a două variabile distincte**

Redactarea codului – comentarii

- o documentație de calitate la nivel de cod
  - depinde în mare măsură de adoptarea unui stil de programare eficient și corect
    - structurarea logică a programului
    - denumiri adecvate pentru variabile, funcții etc
    - minimizarea complexității structurilor de date
    - etc.
  - depinde în mai mică măsură de prezența sau abundența comentariilor

Redactarea codului – comentarii

```
for (i = 2; i <= num; i++)
{
meetsCriteria[i] = true;
}
for (int i = 2; i <= num / 2; i++)
{
j = i + i;
while (j <= num)
{
meetsCriteria[j] = false;
j = j + i;
}
}
for (i = 2; i <= num; i++)
{
if (meetsCriteria[i])
{
std::cout << i + " meets criteria.";
}
}
```

Redactarea codului – comentarii

```
for (i = 2; i <= num; i++)
{
meetsCriteria[i] = true;
}
for (int i = 2; i <= num / 2; i++)
{
j = i + i;
while (j <= num)
{
meetsCriteria[j] = false;
j = j + i;
}
}
for (i = 2; i <= num; i++)
{
if (meetsCriteria[i])
{
std::cout << i + " meets criteria.";
}
}
```

- codul este criptic, dificil de citit
- este redactat într-un stil dezorganizat
- numele variabilelor și funcțiilor nu sunt sugestive
- în forma actuală necesită multiple comentarii
- prezența comentariilor ar rezolva parțial problema

Redactarea codului – comentarii

```
for (primeCandidate = 2; primeCandidate <= num; primeCandidate++)
{
   isPrime[primeCandidate] = true;
}

for (int factor = 2; factor < (num / 2); factor++)
{
   int factorableNumber = factor + factor;
   while (factorableNumber <= num)
   {
      isPrime[factorableNumber] = false;
      factorableNumber = factorableNumber + factor;
   }
}

for (primeCandidate = 2; primeCandidate <= num; primeCandidate++)
{
   if (isPrime[primeCandidate])
   {
      std::cout << primeCandidate + " is prime.";
   }
}
```

- variantă îmbunătățită
- stilul de redactare facilitează lectura codului
- rolul variabilelor și funcțiilor rezultă din denumirile lor
- necesită comentarii minimale

Redactarea codului – comentarii

- în mod ideal, o secvență de cod nu ar trebui să necesite comentarii

- comentariile se impun, totuși, în cazul unor secvențe de cod complexe, unde funcționalitatea nu rezultă în mod imediat

- există numeroase bune practici care minimizează necesitatea comentariilor

Clase:

- **interfața trebuie să consituie o abstracție adecvată a claselor care o implementează**

- **numele clasei trebuie să descrie rolul acesteia**

- **din interfața unei clase trebuie să rezulte clar modul de utilizare a clasei respective**

- **interfața unei clase trebuie să permită utilizarea clasei ca un *black box***
  - **implementarea nu trebuie să afecteze modul de utilizare a interfeței**

Funcții:

- **numele funcției trebuie să descrie clar operația implementată de aceasta**

- **fiecare funcție trebuie să îndeplinească o singură sarcină clar stabilită**

- **fiecare parte a funcției care ar putea fi implementată într-o altă funcție de sine stătătoare trebuie implementată într-o altă funcție de sine stătătoare**

- **modul de utilizare al unei funcții trebuie să rezulte clar din prototipul (declarația) acesteia**

Variabile:

- **numele variabilelor trebuie să fie suficient de descriptivfuncției trebuie să descrie clar operația implementată de aceasta**

- **fiecare funcție trebuie să îndeplinească o singură sarcină clar stabilită**

- **fiecare parte a funcției care ar putea fi implementată într-o altă funcție de sine stătătoare trebuie implementată într-o altă funcție de sine stătătoare**

- **modul de utilizare al unei funcții trebuie să rezulte clar din prototipul (declarația) acesteia**

Organizarea datelor:

- **se practică utilizarea variabilelor auxiliare pentru a spori claritatea implementării**

- **tipurile de dată trebuie să fie suficient de simple cât să minimizeze complexitatea**

- **datele compelxe trebuie accesate prin intermediul rutinelor abstracte (tipuri de dată abstracte)**

Controlul execuției:

- **fluxul execuției trebuie să rezulte clar din organizarea codului**
- **instrucțiunile interdependente trebuie grupate în cadrul acelorași rutine**
- **în ramura principală a unei instrucțiuni *if* trebuie să se trateze situația normală, iar cea excepțională în *else* (ex. o eroare)**
- **o buclă repetitivă trebuie să îndeplinească o singură sarcină (să aibă un singur rol)**
- **clasele și rutinele imbricate (*nested*) trebuie minimizate**

Design:

- **reprezentarea schematică a programului trebuie să fie similară cu structura sa logică**

- **trebuie să se evite complexitatea de dragul complexității**
  (complicarea intenționată a codului pentru a da impresia unui program sofisticat)

- **detaliile legate de implementare trebuie ascunse în măsură cât mai mare (i.e. prin intermediul tipurilor abstracte, interfețelor etc.)**

- **programul trebuie scris în ideea rezolvării unei probleme și nu cu scopul de a folosi anumite principii de programare sau un anumit limbaj de programare**