METODOLOGIA DE LA PROGRAMACIÓ
Guia d'estil de programació

L'objectiu d'aquest document és establir un conjunt de regles i criteris que defineixen un estil de programació i codificació en C++. Un estil de programació ben definit i comú ofereix importants avantatges a l'hora d'escriure i mantenir els programes que codifiquem, així com a l'hora de compartir el codi amb un equip de persones (en el nostre cas, amb els companys de classe i amb els professors):

- Reduïm l'esforç de programació a decidir sobre el disseny de l'algorisme i no sobre qüestions de format i estil.
- El codi es més coherent i consistent.
- Els programes són més fàcils d'interpretar i seguir.
- Evitem cometre errors comuns.
- El codi pot ser més fàcilment modificat i mantingut per programadors diferents.

No existeix un únic estil de programació. Cada organització o equip de programadors ha de definir el seu propi estil. El que us proposem en aquest document està basat en una recopilació d'estàndards i pràctiques habituals entre els programadors en C++ i és el que utilitzarem en els exemples i exercicis de l'assignatura. Fixeu-vos en el codi que us donem nosaltres per veure quines regles d'estil seguim. Una part de l'avaluació de tots els exercicis i pràctiques tindrà en compte que els programes estiguin escrits seguint aquest estil de programació.

Criteris generals d'escriptura del codi

- Posar només una instrucció per línia de codi.
- Utilitzar tabulacions per marcar l'estructura i els diferents blocs del programa.
- Alinear les claus que obren i tanquen un bloc del programa, de la forma següent:

```
while (...)
{
   if (...)
   {
      ...
   }
   else
   {
      ...
   }
}
```

- Deixar sempre un espai en blanc després de les paraules clau i dels caràcters ",", ";" i ":"
- Deixar espais en blanc abans i després dels operadors binaris
- No deixar espai en blanc entre el nom d'una funció i '('

Alguns exemples d'aquests criteris generals:

```
a = (b + c) * d;

while (true)
{
   ...
}

doSomething(a, b, c, d);

case 100:

for (i = 0; i < 10; i++)
{
   ...
}
```

- Si una línia és massa llarga i s'ha de dividir perquè no cap sencera en pantalla, deixar una tabulació extra a l'inici de la següent línia.

Els noms de les coses

- Tots els noms haurien de ser significatius del que representen. Les úniques excepcions haurien de ser variables que s'utilitzin com a comptadors en bucles o paràmetres genèrics en funcions.

- Els noms de les variables comencen per minúscules. Si el nom conté més d'una paraula, les següents paraules es fan començar per majúscula sense separació de cap tipus. Aquest tipus d'escriptura s'anomena 'camelCase' i és molt popular.

```
nom, nomAlumne, nombreAlumnesClasse
```

- Els noms de les funcions haurien de ser verbs expressant l'acció principal que fa la funció. El nom ha de seguir les mateixes regles que per les variables.

```
suma(), calculaMitja()
```

- Els noms dels tipus i classes comencen per majúscules. Si el nom conté més d'una paraula, les següents paraules es fan començar per majúscula sense separació de cap tipus.

```
class Alumne
class LlistaAlumnes
```

- Els noms de les constants s'escriuen en majúscules. Si el nom conté més d'una paraula se separen amb el caràcter "_"

```
NOMBRE_MAXIM_ALUMNES
```

- Si hi ha alguna variable global al programa, el nom de la variable ha de començar per "::"

```
::nombreAlumnes
```

- Els noms de les variables que siguin membres privats d'una classe comencen per "m_". A continuació se segueix la mateixa regla que pels noms de les altres variables.

```
m_nomAlumne
```

- Els noms de les constants d'una enumeració van precedits pel nom de l'enumeració.

```
enum Color { COLOR_RED, COLOR_GREEN, COLOR_BLUE };
```

Variables, constants i expressions

- Sempre que es pugui, inicialitzar les variables en el moment de la declaració. En aquest cas, declarar i inicialitzar només una variable per línia de codi

```
int nAlumnes = 100;
float nota = 0.0;
```

- Si no es pot inicialitzar la variable en el moment de la declaració, inicialitzar-la just a la línia següent.

```
int nAlumnes;
cin >> nAlumnes;
```

- Declarar les variables just en el moment de la seva primera utilització, no abans.

- Evitar variables globals. Idealment, en un programa no hi ha d'haver variables globals, excepte casos molt ben justificats.

- Evitar l'ús de números màgics en els programes. Utilitzar constants per qualsevol valor constant que s'hagi d'utilitzar dins del programa.

- Declarar les constants utilitzant `const`. No utilitzar `#define`

- En apuntadors i referències els caràcters * i & van enganxats al tipus i no al nom de la variable.

```
char* nomAlumne
int& nombreAlumnes
```

- No assumir un ordre predeterminat en l'avaluació dels operands d'una expressió. Poseu parèntesis per a que quedi clar l'ordre en que espereu que s'avaluïn els operands.

Funcions

- Una funció no hauria de sobrepassar les 30 línies de codi (sense comptar comentaris, línies en blanc i línies amb {}). Quan això passa sol ser una indicació que la funció s'hauria de dividir en dues o més.

- Els noms dels paràmetres han de ser significatius en el context de la funció. És a dir, si la funció es diu `calculaMitjanaEdat` llavors el paràmetre s'hauria de dir `edats`. En canvi, si la funció es diu `calculaMitjana` llavors el paràmetre s'hauria de dir `numeros`, tot i que el tipus del paràmetre i el cos de les dues funcions fos el mateix. De totes formes, la segona funció (més genèrica) és més recomanable perquè és més fàcilment reutilitzable en altres casos, per exemple per a calcular la mitjana de pes.

- Si una funció té massa paràmetres, s'han d'especificar en línies diferents de la forma següent:

```
int myComplicatedFunction(unsigned unsignedValue,
                          int intValue,
                          char* charPointerValue,
                          int* intPointerValue,
                          myClass* myClassPointerValue,
                          unsigned* unsignedPointerValue);
```

- Els paràmetres que corresponguin a objectes d'alguna classe s'han de passar per referència. Si la funció no els ha de modificar s'han de declarar com a `const`

Estructures de control

- No es poden utilitzar les instruccions `goto`, `continue`, `break` (excepte dins de la instrucció `switch`).

- Es recomana que les variables que s'utilitzin dins d'un bucle es declarin i inicialitzin just abans del bucle.

- Es recomana que en la capçalera d'un bucle `for` només s'utilitzi la variable de control del bucle.

- No incloure cap altra instrucció en la mateixa línia que comprova una condició en un bucle `while` o en una estructura `if`. Encara que el cos del bucle o del condicional sigui una sola instrucció posar-la en una línia diferent.

- Format d'una instrucció `switch`

```
switch (condicio)
{
   case ABC :
          ...
          break;
   case DEF :
          ...
          break;
   default :
```

```
            ...
            break;
    }
```

Classes

- En la declaració de la classe primer s'ha de posar la part pública i després la part privada.

```
class NomClasse
{
public:
   <constructor/destructor>
   <mètodes que modifiquen les dades de la classe>
   <mètodes que permeten accedir a les dades de la classe>
private:
   <dades>
   <mètodes privats>
};
```

- Totes les dades d'una classe han de ser privades. Es pot fer una excepció amb classes que es comporten com `structs` i només tenen mètodes per recuperar i donar valor a les dades.

- Especificar sempre un constructor per defecte.

- Els mètodes que no modifiquen l'objecte s'han de definir com a `const`.

- Cada classe s'ha de declarar en un fitxer .h separat. El nom del fitxer ha de coincidir amb el nom de la classe. La definició de la classe s'ha de posar en un fitxer .cpp amb el mateix nom.

- Sempre que es pugui, inicialitzar els atributs de la classe utilitzant la llista d'inicialització d'atributs enlloc de fer la inicialització al codi del constructor.

```
class MyClass
{
public:
  MyClass(int value): m_value(value){  }
private:
  int m_value;
};
```

Fitxers de capçalera

- Incloure els fitxers de capçalera necessaris en aquest ordre:

    1. Fitxer .h directament relacionat amb el fitxer .cpp
    2. Altres fitxers .h locals del mateix projecte
    3. Fitxers .h de llibreries externes
    4. Fitxers .h de la llibreria estàndard.

```
#include "my_file.h"
#include "other_file.h"
#include <iostream>
```

- Incloure a tots els fitxers de capçalera una definició per evitar conflictes quan s'inclou el mateix fitxer des de múltiples punts del mateix projecte.

```
#ifndef MY_CLASS_H
#define MY_CLASS_H
...
#endif
```

Comentaris

- Incloure un comentari al principi de cada fitxer de codi font que expliqui el contingut del fitxer:

```
/**
 * FITXER exemple.cpp
 * AUTOR Ernest Valveny
 * DATA 15/01/2015
 * VERSIO 2.5
 * Aquest fitxer conté uns quants exemples que il·lustren l'estil de
 * de programació que farem servir a l'assignatura
 */
```

- Incloure un comentari abans de la definició de cada funció

```
/**
 * calculaMitja
 * Aquesta funció calcula la mitja de dos números reals
 * @param x: primer número real
 * @param y: segon número real
 * @return Mitja dels dos números que es passen com a paràmetres
 */
```

- Incloure un comentari abans de la declaració de cada classe

```
/**
 * CLASS Alumne
 * Classe que permet guardar les dades d'un alumne de l'assignatura
 * Les dades que conté són: nom, NIU, grup de classe i nota final de
 * l'assignatura
 * Conté mètodes per accedir i modificar les dades i també per calcular
 * la nota final de l'alumne donades totes les seves notes del curs
 */
```