Normes bàsiques de programació PRO1: C1

Professorat de Programació 1

4 de febrer de 2020

1 Preàmbul

Totes les afirmacions que segueixen s'han d'interpretar només en el context d'una assignatura d'aprenentatge de la programació, i no de forma universal.

Consideracions generals:

- Els programes escrits en C++ tenen l'extensió `.cc`.
- Cada línia de codi té una única instrucció bàsica.

2 Estructura d'un programa

Els programes tenen cinc apartats:

1. Inclusions i espai de noms
2. Definició de constants — pot ser buit
3. Definició de nous tipus (typedef i structs) — pot ser buit (es veurà al C3)
4. Procediments — pot ser buit (es veurà al C2)
5. Programa principal (`main`)

2.1 Inclusions i espai de noms

Només es posen les inclusions necessàries, amb la directiva `#include`. Les inclusions admeses al C1 són:

- `<iostream>`: defineix les instruccions `cin` (llegir), `cout` (escriure) i `endl` (salt de línia). Si fem servir aquestes instruccions sense fer l'include, resultarà en un error de compilació.

  Està prohibit:

  - ✗ Usar caràcters com `'\n'` per escriure salt de línia (només es permet `endl`).
  - ✗ Usar altres operacions d'entrada/sortida com ara `getline()`.
  - ✗ La lectura i escriptura a l'estil de C, amb `scanf()`, `printf()` o similars.

- `<cmath>`: defineix operacions i transformacions matemàtiques. Nosaltres NOMÉS podrem fer servir la constant `M_PI` (π amb la millor precisió permesa per la màquina), les funcions trigonomètriques `sin()`, `cos()` i `tan()`, i l'arrel quadrada `sqrt()`.

  Està prohibit usar cap altra operació matemàtica que no siguin les anteriors.

A continuació de les inclusions, cal afegir sempre la línia:

```
using namespace std;
```

Està prohibit:

- ✗ Usar qualsevol altre espai de noms o qualsevol inclusió que no siguin les anteriors.
- ✗ Excepte `#include`, totes les altres directives estan prohibides.

2.2 Definició de constants

Cada constant s'ha de definir en una línia apart, usant la sintaxi:

```
const nom_de_tipus nom_de_constant = valor;
```

La paraula clau `const` fa que es defineixi una constant i no una variable global. Una variable és global si està declarada fora de tots els procediments, el `main` inclòs.

Està prohibit:

- ✗ Usar variables globals.
- ✗ Usar 'nombres màgics'. Per exemple:

  ```
  if (n > 350) cout << "Esta bastant ple." << endl;
  ```

  En lloc seu, cal fer servir constants.

  ```
  const int MAX_PERSONES = 700;
  ...
  if (n > MAX_PERSONES/2) cout << "Esta bastant ple." << endl;
  ```

2.3 Tipus bàsics i operadors

Els únics tipus bàsics que es poden fer servir són: `int`, `double`, `bool` i `char`. Durant la primera part del curs es tractarà també el tipus `string` com a bàsic. Com es veurà a la segona part del curs aquest tipus de dades no és en realitat simple.

Està prohibit:

- ✗ Usar qualsevol altre tipus bàsic o modificador. Per exemple, no es pot usar `short (int)`, `long (int)`, `long long (int)`, `unsigned int` o `float`.

Les variables amb tipus bàsics es poden declarar (amb o sense valor inicial), assignar, comparar, llegir i escriure. A més, es poden utilitzar els operadors `+=`, `-=`, `*=`, `/=`, i `%=`.

```
#include <iostream>
using namespace std;

const string SALUTACIO = "Hola!!!";

int main() {
    cout << SALUTACIO << endl;
    string s, t;
    cin >> s >> t;
    if (s > t) {
        string aux = s;
        s = t;
        t = aux;
    }
    cout << s << " es mes petit o igual que " << t << endl;
}
```

2.4 Programa principal

El programa principal és el punt on comença l'execució. Té sempre l'estructura:

```
int main() {
    instrucció 1;
    ....;
    instrucció N;
}
```

Encara que el `main` ha de retornar un enter que indica l'estat final del programa (0, si el programa ha anat bé), el compilador entén que hi ha implícitament la instrucció `return 0;` al final del programa. Per tant, no cal posar-la mai. Addicionalment, cal fixar-se que el nostre `main` no té cap paràmetre.

3 Noms de constants i variables

Els noms de constants i variables han de ser significatius, tot seguint les següents convencions:

- Els formarem amb lletres. Ocasionalment, poden contenir algun dígit. Sempre han de començar per una lletra.

  ```
  int cost;  // NO: int 1cost;
  ```

- Si un nom està format per diverses paraules, aquestes se separen amb un '_'.

  ```
  int max_cost;  // millor que: int maxcost;
  ```

- Els noms de variables s'escriuen amb lletres minúscules.

  ```
  int cost;      // NO: int COST;
  ```

- Els noms de constants s'escriuen exclusivament amb lletres majúscules.

  ```
  const double PROPORCIO_DIVINA = (1 + sqrt(5.0))/2;
  const int MAX_PERSONES = 1000;
  ```

- A les variables, a vegades, és preferible usar noms curts per augmentar la llegibilitat.

  ```
  int max;  // millor que: int max_cost; si és l'únic màxim calculat
  ```

- Cal evitar posar noms negats a les variables booleanes.

```
bool trobat = false;   // millor que: bool no_trobat = true;
bool seguir = true;    // millor que: bool no_seguir = false;
```

4 Declaració i inicialització de variables

- Una variable es declara quan es necessita, mai abans, i sovint s'inicialitza en la pròpia declaració. En aquest cas, cal declarar-la individualment.

Correcte:

```
double x = a - b;
double y = c - d;
```

Penalitzable:

```
double x, y;
x = a - b;
y = c - d;
```

```
double x = a - b, y = c - d;
```

- Les variables s'han de declarar dins de l'àmbit de visibilitat més intern possible. En cas contrari diem que la variable està declarada fora d'àmbit.

Correcte:

```
int x, y;
cin >> x >> y;
if (x > y) {
    int aux = x;
    x = y;
    y = aux;
}
cout << x << " " << y << endl;
```

Penalitzable:

```
int x, y, aux;
cin >> x >> y;
if (x > y) {
    aux = x;
    x = y;
    y = aux;
}
cout << x << " " << y << endl;
```

En el cas penalitzable, la variable auxiliar `aux` existeix abans i després del condicional. Tanmateix, el seu valor només el volem per emmagatzemar el de x temporalment.

- Cal evitar que variables d'àmbits interns amaguin altres variables d'àmbits més externs. Encara que el codi fos correcte, seria molt confús.

Correcte:

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
        cout << "*";
    cout << endl;
}
```

Penalitzable:

```
for (int i = 0; i < n; ++i) {
    for (int i = 0; i < n; ++i)
        cout << "*";
    cout << endl;
}
```

- Algunes situacions típiques en què no s'inicialitza una variable en la pròpia declaració són:

  – Per llegir.

```
int x, y;
cin >> x >> y;
```

– Quan el valor depèn d'una condició.

```
double segons;
if (graus) segons = 3600*angle;
else segons = 60*angle;
```

– Quan el valor depèn d'un bucle.

```
// Volem sumar tots els nombres de l'entrada i saber
// la posicio de qualsevol que sigui negatiu.
// Sabem que n'hi ha un, com a mínim.
double suma = 0;
int pos_negatiu, n;
cin >> n;
for (int i = 0; i < n; ++i) {
    int x;
    cin >> x;
    suma += x;
    if (x < 0) posicio_negatiu = i;
}
```

– En algun cas extrem com el següent:

```
int i1 = 0;
int i2 = 0;
int i3 = 0;
int i4 = 0;
int i5 = 0;
int i6 = 0;
int i7 = 0;
int i8 = 0;
```

és millor estalviar línies:

```
int i1, i2, i3, i4, i5, i6, i7, i8;
i1 = i2 = i3 = i4 = i5 = i6 = i7 = i8 = 0;
```

5 Expressions booleanes

Hem de tenir en compte que:

- Les expressions booleanes en C++ s'avaluen d'esquerra a dreta i parant quan es coneix el resultat. Això es pot aprofitar per simplificar el codi.

Més compacte:

```
// escriu el primer element
// parell en una seqüència
// amb marca final -1
// o -1 si no existeix
int x;
cin >> x;
while (x != -1 and x%2 != 0) {
    cin >> x;
}
cout << x << endl;
```

Primera versió:

```
// escriu el primer element
// parell en una seqüència
// amb marca final -1
// o -1 si no existeix
int x;
cin >> x;
bool trobat = false;
while (x != -1 and not trobat) {
    if (x%2 == 0) trobat = true;
    else cin >> x;
}
cout << x << endl;
```

- Sempre hem de parar atenció a les comparacions d'igualtat o de desigualtat entre nombres reals, especialment si aquests són el resultat d'una sèrie d'operacions, perquè poden tenir petits errors d'arrodoniment que poden fer fallar el programa.

És penalitzable:

× Comparar una variable booleana amb `true` o amb `false`.

Correcte:

```
if (trobat) ...
if (not repetit) ...
```

Penalitzable:

```
if (trobat == true) ...
if (repetit == false) ...
```

6 Bucles

Les úniques formes admeses de la instrucció `for` són:

```
for (tipo var = expr1; var < expr2; ++var) ...
for (tipo var = expr1; var > expr2; --var) ...
```

i les anàlogues amb `<=` i `>=`, respectivament, on `var` és la variable de control, `tipo` és és el seu tipus i pot ometre's si el valor de `var` s'usa més endavant, i `expr1`, `expr2` són expresions que no contenen `var`.

Està prohibit:

× Modificar la variable de control d'un `for` dins del cos del `for`.

És penalitzable:

× Posar condicions complexes al `for`. En aquest cas, es transformarà a `while`.

Correcte:

```
int i = 0;
while (not trobat and i < n) {
    int e;
    cin >> e;
    if (e == x) trobat = true;
    ++i;
}
```

Penalitzable:

```
for (int i = 0; not trobat and i < n; ++i) {
    int e;
    cin >> e;
    if (e == x) trobat = true;
}
```

7 Lectura d'una seqüència d'entrada

Hi han tres formes de representar una seqüència a l'entrada:

- Primer ens donen el número d'elements de la seqüència i, a continuació, la seqüència.
- Primer ens donen la seqüència i després una marca de fi.
- Directament ens donen la seqüència.

L'estructura de la lectura serà diferent depenent de la forma de representació. A continuació mostrem el codi per cada cas.

Número d'elements + seqüència

```
int n;
cin >> n;

for (int i = 0; i < n; ++i) {
     tipus_elem x;
     cin >> x;
     // tractar element x
     ...
}
```

Seqüència + marca de fi

```
tipus_elem x;
cin >> x;
while (x != marca_fi) {
    // tractar element x
    ...
    cin >> x;
}
```

Directament seqüència

```
tipus_elem x;
while (cin >> x) {
    // tractar element x
    ...
}
```

8 Estil

8.1 Indentació

- S'indenta amb quatre espais per nivell.
- Cal evitar els tabuladors, configurant l'editor per desactivar-los.
- Les línies de més de 78 caràcters s'han de trencar, ja que sovint no es poden imprimir bé.
- Si un tros de codi té massa indentacions, cal fer-lo més llegible usant algun procediment auxiliar (a partir del C2).

8.2 Espais

Com a norma general, cal deixar exactament un espai de separació entre les paraules clau, instruccions, literals, etcètera. Les excepcions més importants són:

- Els punts i coma s'escriuen enganxats a allò que tinguin a la seva esquerra.

```
int x;                  // NO: int x  ;
```

- Els operadors binaris `*` (producte), `/` (divisió) i `%` (residu), i l'operador unari `-` (canvi de signe) s'escriuen sense espais per emfasitzar la seva alta precedència.

```
int a = x + y*z;        // NO: int a = x + y * z;
int a = -b;             // NO: int a = - b;
```

- Els operadors `++` (preincrement) i `--` (predecrement) s'escriuen a l'esquerra de la variable que modifiquen, sense cap espai.

```
++i;                    // NO:  ++ i;
```

8.3 Claus

Hi ha moltes maneres i variants de posar les claus. La que nosaltres recomanem és:

```
if (condició1) {
    ...
}
else if (condició2) {
    ...
}
else {
    ...
}

while (condició) {
    ...
}

for (inicialització; condició; modificació) {
    ...
}
```

Fixa't que:

- No es posa mai res a la dreta d'una clau, ja sigui d'obrir o de tancar.
- Sempre que s'obre una clau, va precedit d'un espai en blanc.

Quan dintre del cos d'un `if`, d'un `while` o d'un `for` només es realitza una acció, i aquesta cap a la mateixa línia, es pot evitar posar les claus per fer un codi més curt i compacte:

```
int n;
cin >> n;
int prod = 1;
for (int i = 0; i < n; ++i) prod *= i;
```

8.4 Parèntesis

- Els parèntesis se separen de les paraules clau i la part interna d'un parèntesi no se separa d'allò que contingui.

```
if (x > 4 and y < 10)          // NO: if(  x > 4 and y < 10  )
(...)
while (x != -1)                // NO: while(  x != -1  )
```

- Com a norma general, només es posen els parèntesis necessaris per canviar la precedència dels operadors.

```
int x = a + b*c;               // NO: int x = a + (b*c);
```

- En els casos en què la precedència no és òbvia a la vista (per exemple, quan caldria recordar si les operacions de la mateixa prioritat s'avaluen d'esquerra a dreta o a l'inrevés), és convenient afegir parèntesis per augmentar la llegibilitat.

```
int a = (x/y)*z;               // Millor que: int a = x/y*z;
```

- Excepcionalment, si una expressió és tan complicada que costa llegir-la sense parèntesis redundants, aquests es poden afegir. En aquests casos, però, sovint és millor usar variables intermitges per simplificar les expressions.

8.5 Línies en blanc

Convé separar amb (una, per exemple) línies en blanc els diferents apartats d'un programa (inclusions, constants, tipus, procediments i `main`). Dins de cada apartat, convé separar cadascun dels seus subapartats amb els mateix nombre de línies en blanc, amb dues excepcions: totes les declaracions de constants i totes les definicions de tipus s'escriuen sovint sense cap separació.

A vegades, quan un tros de codi és llarg, és bo posar una línia en blanc que en separi dos blocs conceptuals.

Separant trossos de codi:

```
#include <iostream>
using namespace;

int main() {
    int n;
    cin >> n;

    int suma = 0;
    for (int i = 0; i < n; ++i) {
        int x;
        cin >> x;
        suma = suma + x;
    }

    cout << suma << endl;
}
```

Sense separar codi:

```
#include <iostream>
using namespace;
int main() {
    int n;
    cin >> n;
    int suma = 0;
    for (int i = 0; i < n; ++i) {
        int x;
        cin >> x;
        suma = suma + x;
    }
    cout << suma << endl;
}
```

9 Comentaris i documentació

Per documentar s'usa `//`. Ens reservem `/* */` per a la fase de depuració de programes. Així es poden compilar només certs trossos de codi, esborrant lògicament (però no físicament) els trossos que no es volen compilar. El programa final no ha d'incloure codi "esborrat" amb `/* */`.

Cal documentar el codi en la justa mesura. Posar comentaris redundants (per exemple, explicar el que fa cadascuna de les instruccions) pot ser tan perniciós com ometre els comentaris importants.

Aquest és un exemple del que NO s'ha de fer:

```
int n;                          // declarem una variable entera
cin >> n;                       // llegim el valor de n
int suma = 0;                   // declarem suma i l'inicialitzem a 0
for (int i = 0; i < n; ++i) {   // mentre hagin elements a l'entrada
    int x;                      // declarem una variable entera
    cin >> x;                   // llegim l'element actual
    suma = suma + x;            // actualitzem suma amb x
}
cout << suma << endl;           // escrivim suma
```

10 Consideracions finals

Construccions no recomanades en un curs introductori[1]:

- × Els postincrements: `i++; j--;`
- × La construcció `do while`.
- × Els operadors sobre bits: `&`, `|`, `<<`, `>>`, `^`.
- × La construcció *condició_booleana* `?` *expressió1* `:` *expressió2*.
- × Els operadors booleans obsolets `&&`, `||`, `!`. En lloc seu, cal usar els moderns `and`, `or`, `not`.

[1] Algunes d'aquestes construccions no presenten cap inconvenient per als programadors experts.

- ✗ La conversió de tipus de C. En els pocs casos en què s'hagi de convertir d'un tipus a un altre, convé fer-ho explícitament, i usant la notació moderna de C++.

  Conversió C++:

  ```
  int i = int(M_PI);
  cout << char('a' + i);
  ```

  Conversió C:

  ```
  int i = (int)M_PI;
  cout << (char)('a' + i);
  ```

- ✗ La comparació implícita amb el valor nul del tipus (excepte amb els booleans).

  Versió preferida:

  ```
  int x, y;
  cin >> x >> y;
  if (x == 0) ...;
  if (y != 0) ...;
  ```

  Versió a evitar:

  ```
  int x, y;
  cin >> x >> y;
  if (!x) ...;
  if (y) ...;
  ```

Està prohibit:

- ✗ Usar punters.
- ✗ Usar les paraules clau `new`, `delete`, `goto`, `continue`, `break`, `switch`, `try`, `throw`, `catch`, `template` i `operator`.
- ✗ Fer crides a sistema.
- ✗ En general, fer servir qualsevol instrucció que no s'hagi vist a classe.

Referències

- http://www.chris-lott.org/resources/cstyle/CppCodingStandard.html
- http://www.research.att.com/~bs/JSF-AV-rules.pdf
- http://gcc.gnu.org/onlinedocs/libstdc++/17_intro/C++STYLE
- http://www.horstmann.com/bigcpp/styleguide.html
- http://www.spelman.edu/~anderson/teaching/resources/style/
- http://geosoft.no/development/cppstyle.html
- http://geosoft.no/development/cpppractice.html