Normes bàsiques de programació PRO1: C2

Professorat de Programació 1

15 de setembre de 2020

1 Preàmbul

Aquestes normes són la continuació del document 'Normes bàsiques de programació PRO1: C1'.

2 Estructura d'un programa

Recorda que 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
5. Programa principal (`main`)

2.1 Inclusions

- `<vector>`: Es poden declarar (amb o sense mida inicial i, en el primer cas, amb o sense valor inicial a cada posició) i assignar. A més, es pot consultar la seva mida i accedir-ne a cadascuna de les posicions.

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

// Pre: -
// Post: llegeix de l'entrada una seqüència que emmagatzema en v
void llegeix_vector(vector<double>& v) {
    int n;
    cin >> n;
    v = vector<double>(n);
    for (int i = 0; i < n; ++i) cin >> v[i];
}

// Pre: v és vàlid
// Post: el valor dels elements de v es dobla
```

```
void dobla(vector<int>& v) {
    for (int i = 0; i < v.size(); ++i) v[i] *= 2;
}

int main() {
    vector<double> v;
    llegeix_vector(v);
    vector<int> u(100, -4);
    dobla(u); // ara u és igual a un vector<int>(100, -8)
}
```

És penalitzable:

- Fer servir vectors quan no és estrictament necessari (o al menys, quan usar-los impliqui una solució més ineficient).

Està prohibit:

- Usar cap altra operació sobre vectors.

- `<string>`: Defineix operacions que permeten manipular els strings com a vectors. Es poden declarar donant la mida i el valor inicial a cada posició, es pot consultar la seva mida, i es pot accedir a cadascuna de les seves posicions.

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

int main() {
    string s;
    cin >> s;
    cout << s << " te " << s.size() << " caracter(s)" << endl;
    cout << "el primer caracter es " << s[0] << endl; // s no buit

    ...

    int n;
    char c;
    cin >> n >> c;
    string t(n, c);
    cout << t << " te " << n << " caracter(s)" << endl;
    cout << "tots iguals a " << c << endl;
}
```

Està prohibit:

- Usar cap altra operació sobre strings.

- `<algorithm>`: En els exercicis de final de curs que no diguin explícitament el contrari, si cal ordenar un vector es pot usar el procediment estàndard `sort()`.

```
vector<int> v(MAX);
for (int i = 0; i < MAX; ++i) cin >> v[i];
sort(v.begin(), v.end());
cout << "ordenats:";
for (int i = 0; i < MAX; ++i) cout << " " << v[i];
cout << endl;
```

Si es vol ordenar un vector amb algun criteri diferent de l'habitual (de petit a gran), es pot fer fàcilment usant una funció de comparació que indiqui quan un element és més petit que un altre.

```
// Volem ordenar el vector de gran a petit.
// Per fer-ho, implementem aquesta funció:
bool comp(int a, int b) {
    return a >= b;
}

int main() {
    vector<int> v;
    ...
    sort(v.begin(), v.end(), comp);
    ...
}
```

Està prohibit:

- Usar cap altra operació de la llibreria `<algorithm>`.

2.2 Procediments

Hi ha dos tipus de procediments:

- Funcions: en general, tenen un únic paràmetre de sortida (el valor que retorna la funció) i la resta són paràmetres d'entrada.

```
// Pre: n >= 0
// Post: retorna el número de dígits de n
int num_digits(int n) {
    ndigits = 1;
    while (n > 9) {
        ++ndigits;
        n /= 10;
    }
    return ndigits;
}
```

- Accions: en general, tenen més d'un paràmetre d'entrada i/o entrada/sortida. No retornen cap valor explícitament, cosa que es marca amb un `void`. Tampoc no apareixerà la instrucció return.

```
// Pre: n >= 0
// Post: calcula la suma dels dígits de n i el número de dígits de
    n
void suma_i_num_digits(int n, int& suma, int& ndigits) {
    suma = 0;
    ndigits = 1;
    while (n > 9) {
        ++ndigits;
        suma += n%10;
        n /= 10;
    }
    suma += n;       // sumem l'últim dígit
}
```

Com a excepció, si algun paràmetre de sortida és un booleà, llavors pot tenir sentit fer una funció booleana en comptes d'una acció. Per exemple,

```
// Pre: n >= 0
// Post: calcula la suma dels dígits de n i retorna true si
//       el nombre de dígits és parell, false en cas contrari
bool suma_i_parell(int n, int& suma) {
    suma = 0;
    int ndigits = 1;
    while (n > 9) {
        ++ndigits;
        suma += n%10;
        n /= 10;
    }
    suma += n;      // sumem l'últim dígit
    return ndigits%2 == 0;
}
```

en comptes de:

```
// Pre: n >= 0
// Post: calcula la suma dels dígits de n i retorna true si
//       el nombre de dígits és parell, false en cas contrari
void suma_i_parell(int n, int& suma, bool& parell) {
    suma = 0;
    int ndigits = 1;
    while (n > 9) {
        ++ndigits;
        suma += n%10;
        n /= 10;
    }
    suma += n;      // sumem l'últim dígit
    parell = ndigits%2 == 0;
}
```

Hi ha tres tipus de paràmetres:

- d'entrada:
  - Els tipus bàsics i strings[1] es passen per valor.
  - Altrament (vectors i structs), es passen per referència constant.
- d'entrada/sortida: es passen per referència.
- de sortida: es passen per referència.

Tingues en compte que:

- És una bona pràctica posar els paràmetres en aquest ordre: entrada, entrada/sortida, sortida.
- Per evitar que un nom n'amagui un altre és bo donar a les variables noms diferents dels noms dels procediments. Això és imprescindible en els procediments recursius.

Està prohibit:

× Usar referències fora del pas de paràmetres.

[1] Estem suposant que els strings usats en els exercicis del curs són 'prou petits'. Altrament, caldria passar-los per referència constant.

3 Noms del procediments

Els noms de procediments s'escriuen exclusivament amb lletres minúscules. Igual que els noms de constants i variables, si estan formats per més d'una paraula se separaran amb un '_' i han de ser significatius. En particular, no ser capaç de trobar un bon nom és senyal que el procediment no està prou ben meditat.

És convenient seguir les següents convencions:

- Els noms de les accions són (o inclouen) verbs en forma imperativa.

```
void intercanvia(int& a, int& b);
void elimina_digits(int d, int& n);
```

- Les funcions que retornen un booleà comencen habitualment amb `"es_"` o `"esta_"`.

```
bool es_primer(int n);
bool esta_contingut(int d, int n);
```

  Però també hi ha altres possibilitats:

```
bool pertany(int d, int n);
```

- El nom de les funcions que retornen tipus no booleans solen ser substantius que indiquen què es retorna.

```
int factorial(int n);
int nombre_de_repeticions(char c, string s);
```

  Però també:

```
char a_minuscula(char c);
```

4 Declaració i inicialització de variables

- Algunes situacions típiques en què no s'inicialitza una variable en la pròpia declaració són:
  - Quan s'usarà immediatament com a paràmetre de sortida.

Correcte:

```
int x;
cin >> x;
int s, d;
// segon i tercer paràmetre
// són de sortida
suma_i_num_digits(x, s, d);
```

Penalitzable:

```
int x;
cin >> x;
int s = 0;
int d = 0;
suma_i_num_digits(x, s, d);
```

5 Bucles

Sempre que no afecti a la llegibilitat del codi, les instruccions `while` i `for` poden contenir instruccions `return`. Hi ha alguns casos on posar un `return` dins d'un bucle és la manera més senzilla i fiable de programar, i produeix codi més eficient i llegible.

Aquest codi:

```
// Pre: 0 <= d <= 9 i n > 0
// Post: retorna true si d és un dígit de n, false en cas contrari
bool pertany(int d, int n) {
    while (n != 0) {
        if (n%10 == d) return true;
        n = n/10;
    }
    return false;
}
```

és una alternativa vàlida al següent codi:

```
// Pre: 0 <= d <= 9 i n > 0
// Post: retorna true si d és un dígit de n, false en cas contrari
bool pertany(int d, int n) {
    bool hi_es = false;
    while (not hi_es and n != 0) {
        if (n%10 == d) hi_es = true;
        n = n/10;
    }
    return hi_es;
}
```

Tingues en compte que:

- Només s'accepta aquest ús de `return` per interrompre un bucle quan sigui clarament avantatjós respecte a no usar-lo.

6 Estil

6.1 Espais

- El caràcter & (per indicar que un paràmetre es passa per referència) s'escriu immediatament a la dreta del tipus al qual està associat.

  ```
  void nom_accio(int& x) { // NO: void nom_accio(int  & x) {
  ```

- No hi ha espai en blanc entre el corxet esquerre i el nom del vector, ni al voltant del contingut dels corxets d'accés a un vector.

  ```
  v[i + 3];      //  NO: v [ i + 3 ];
  ```

6.2 Claus

La forma que nosaltres recomanem és:

```
void una_accio_qualsevol(...) {
    ...
}
```

```
int una_funcio_qualsevol(...) {
    ...
}
```

Recorda 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'una espai en blanc.

6.3 Parèntesis

- El parèntesi esquerre d'un procediment s'escriu enganxat al seu nom.

```
bool pertany(int d, int n) { // NO: bool pertany  (int d, int n) {
```

- La part interna del parèntesi no se separa de la llista de paràmetres.

```
bool pertany(int d, int n) { // NO: bool pertany( int d, int n ) {
```

- No es posen parèntesis al voltant d'allò que retorna una funció.

```
return trobat;              // millor que:  return (trobat);
```

6.4 Línies en blanc

S'han de separar amb (una, per exemple) línia en blanc cadascun dels procediments que es defineixin.

Correcte

```
int una_funcio_qualsevol(...) {
    ...
}

void una_accio_qualsevol(...) {
    ...
}

void main() {
    ...
}
```

Penalitzable

```
int una_funcio_qualsevol(...) {
    ...
}
void una_accio_qualsevol(...) {
    ...
}
void main() {
    ...
}
```

7 Comentaris i Documentació

Cal explicar *què* fa cada procediment. Es farà mitjantçant:

- Precondició: són les condicions que han de complir els paràmetres d'entrada i d'entrada/sortida.
- Postcondició: són les condicions que compleixen els paràmetres d'entrada/sortida i de sortida.

o, de forma alternativa,

- Amb una explicació en llenguatge natural.

És una bona pràctica comprovar que la documentació deixa clar el paper de cada paràmetre, i que és suficient per poder fer servir-lo sense haver de mirar-ne el cos.

Cal no confondre això amb *com* ho fa, que pot explicar-se opcionalment quan no sigui trivial o el mètode sigui diferent de l'habitual.

```
//  Pre: n >= 0.
//  Post: retorna n!
int factorial(int n) {
    if (n == 0) return 1;
    else return n*factorial(n - 1);
}

//  Intercanvia els valors d'a i b.
void intercanvia(int& a, int& b) {
    int aux = a;
    a = b;
    b = aux;
}

// Calcula el dígit màxim i mínim de n
// Pre: n >= 0
// Post: maxim i minim són el dígit màxim i mínim de n, respectivament
// Explicació:
// El màxim i mínim d'un nombre amb un sol dígit és ell mateix.
// Altrament, obtenim quin és el dígit màxim i mínim de n sense
// el seu últim dígit i llavors comprovem si aquest últim dígit és el
// nou màxim o mínim.
void maxim_minim(int n, int& maxim, int& minim) {
    if (n < 10) maxim = minim = n;
    else {
        maxim_minim(n/10, maxim, minim);
        int d = n%10;
        if (d > maxim) maxim = d;
        else if (d < minim) minim = d;
    }
}
```

Fixa't que:

- La documentació d'un procediment es posa just per sobre de la seva capçalera.

És penalitzable:

× Declarar un procediment sense documentar-lo d'aquesta manera.

8 Consideracions finals

Construccions no recomanades en un curs introductori:

× Els prototipus, és a dir, declarar els procediments abans de la seva definició. No calen perquè no hi ha recursivitat encreuada.

× Preincrements amb efectes laterals.

```
int i = -1;
while (++i < v.size() and v[i] != x);
if (i < v.size()) cout << x << " es troba a " << i << endl;
```

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