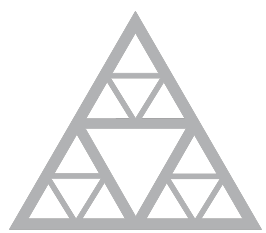
Bonnes pratiques de programmation en C++

Nicolas Audebert

Mercredi 15 février 2017



Bien travailler en C++

Compiler souvent

- Compiler permet de s'assurer qu'il n'y a pas d'**erreur de type** (exemple : `std::string` au lieu d'un `float`)
- Compiler permet de s'assurer qu'il n'y a pas d'**erreur de syntaxe** (exemple : oubli d'un `;`)
- Compiler permet de lever des **avertissements** : variables déclarées mais non utilisées, transtypages douteux, etc.

Dans son environnement de développement

QtCreator et **VisualStudio** permettent de compiler en un clic. Abusez-en.

Style de programmation

Accolades

Choisir un style et s'y **tenir**.

```
// Style K&R
void a_function(int x, int y){
    if (x == y) {
        something1();
        something2();
    } else {
        somethingelse1();
        somethingelse2();
    }
    finalthing();
}
```

```
// Style Allman
void a_function(int x, int y)
{
    if (x == y)
    {
        something1();
        something2();
    }
    else
    {
        somethingelse1();
        somethingelse2();
    }
    finalthing();
}
```

Style de programmation

Noms de fonctions, variables, classes

On préférera utiliser la `CamelCase` pour les classes et la `snake_case` pour tout le reste.

Exemple

```
class Cluster {
  private:
    int r, g, b;
    int nb_points;
  public:
    get_average_color();
};

Color Cluster::get_average_color(){
  return Color(r,g,b);
}
```

Style de programmation

Écrire du code clair et concis

Éviter les parenthèses superflues, les variables inutiles, les lignes à rallonge...

```
// NON ! Code peu élégant
bool Cluster::operator==(Cluster C){
    bool est_egal=true;
    if (n!=C.n || r!=C.r || g!=C.g || b!=C.b){
        est_egal=false;
    }
    return est_egal;
}
```

```
// Version plus claire
bool Cluster::operator==(Cluster C){
    bool egal = n == C.n && r == C.r && g == C.g && b
     ↪    == C.b;
    return egal;
}

// On peut même directement écrire
bool Cluster::operator==(Cluster C){
    return (n == C.n && r == C.r && g == C.g && b ==
     ↪    C.b);
}
```

Style de programmation

Indentation

Chaque bloc doit être **indenté** et si possible mis en valeur par des accolades.

```
// NON
double maximum(double x, double y){
double res;
if (x>y)
res=x;
else
    res=y;
    return y;}
```

```
// Oui
double maximum(double x, double y){
    if (x > y) {
        return x;
    } else {
        return y;
    }
}
```

Style de programmation

Écrire du code lisible

- Indentation correcte (pas de bloc mal indenté)
- Noms de variables et de fonctions clairs
- Du code **commenté**

```
bool Vector::nul const (){
    if (size()==0)
        return (true);
    int n = size();
    int res = 0;
    for(int i=0; i<n; i++){
        res = (res + tab[i]);
    }
    if (res == 0)
        return true;
    return false;
}
```

```
// Détermine si la somme du vecteur est nulle
// Ne modifie pas le vecteur
bool Vector::somme_nulle() const {
    // On vérifie que le vecteur a une taille > 0 pour
     ↪    accéder aux valeurs
    int taille = size();
    if taille == 0
        return true;
    int somme = 0;
    // Calcul de la somme des éléments de tab
    for(int i=0; i<taille; i++){
        somme += tab[i];
    }
    return (somme == 0);
}
```