Convention d'écriture des programmes en langage C++

I.U.T. informatique

Christine JULIEN

TABLE DES MATIERES

Dans ce qui suit, nous explicitons la convention d'écriture des programmes adoptée en C++. Elle s'inspire en grande partie de celle utilisée en langage algorithmique et du guide francophone des conventions de codage pour la programmation en Java (http://www.cyberzoide.net/java/javastyle/.).
Cette convention devra être impérativement respectée dans l'écriture de tout programme C++.

Tout non respect d'une de ces règles entraînera le rejet systématique du programme.

1. Fichiers

1.1. Suffixes

Le suffixe qui suit le nom d'un fichier indique explicitement le type des informations qu'il contient .
On utilise les suffixes suivant :

Suffixe	Type du fichier
.C	fichier source C++
.o	fichier objet C++
.exe	fichier exécutable C++
.txt	fichier de données

1.2. Noms de fichiers

Les noms de fichiers doivent être explicites et être suivis de leur suffixe.

```
etudiants1ereAnnee.txt
jeuDuMorpion.C
```

2. Identificateurs

En règle générale, les identificateurs doivent avoir des noms explicites. La seule exception autorisée à cette règle est d'utiliser des variables muettes *i*, *j*, *k*, ... pour les indices de tableaux et les répétitions dont on connaît à l'avance le nombre d'itérations.

Identificateur de type
Un identificateur de type s'écrit en minuscules et commence par une majuscule. Si l'identificateur est composé de plusieurs mots, chaque mot commence par une majuscule.

```
struct Date : ... ;
typedef int MatriceCreuse[100];
```

Identificateur de variable ou de sous-programme
Un identificateur de variable, ou de sous-programme s'écrit en minuscules. Si l'identificateur est composé de plusieurs mots, chaque mot autre que le premier commence par une majuscule. Un identificateur de sous-programme C++ se comportant comme une procédure algorithmique est un verbe à l'infinitif (*action*). Un identificateur de sous-programme C++ se comportant comme une fonction algorithmique est un nom ou un adjectif (*expression*).

```
float moyenne;
```

```
Date dateDuJour;
void chercherMaximum();
int max(const TabEntiers tab);
bool estVide(const Liste l);
```

Identificateur de constante

Un identificateur de constante s'écrit en majuscules. Si l'identificateur est composé de plusieurs mots, chaque mot est séparé du suivant par le symbole souligné.

```
const int MAX = 100;
const int LG_MAX = 10;
const int NB_ELEVES = 50;
```

3. Lignes blanches

Les lignes blanches permettent d'isoler des paragraphes du programme et contribuent à clarifier sa structuration.

On introduit deux lignes blanches :

- après le cartouche de commentaires de début du programme.

On introduit une ligne blanche :

- après chaque partie du programme correspondant à l'inclusion des bibliothèques, à la déclaration des constantes, à la définition des types enregistrement, à la définition des types tableau, à la spécification des sous-programmes, à la définition du programme principal et à la définition des sous-programmes,
- après la déclaration des variables d'un sous-programme,
- après le corps d'un sous-programme.

Pour identifier aisément les différents paragraphes d'un programme, les blocs d'instructions { } sont alignés.

4. Espaces

Les espaces isolent les unités syntaxiques d'une expression ou d'une action.

On introduit un espace :

- après une virgule,
- après tout mot réservé du langage,
- avant et après le symbole d'affectation =, les opérateurs arithmétiques binaires et les opérateurs relationnels.

```
lire(nbValeurs, valeurs);
f = factorielle(n) * k ;
int premier(const Liste l);
int max;
```

5. Coupures

Lorsqu'une expression ne contient pas sur une seule ligne (80 caractères), appliquer les principes de mise en page suivants :

- couper après une virgule,
- couper avant un opérateur,
- introduire une indentation de 4 espaces par rapport à la ligne précédente s'il s'agit d'une sous-expression,
- aligner les sous-expressions de même niveau.

```
void rechercherOccurrence
    (const TabEntiers tab, const int n, const int x,
    bool &trouve, int &rang);

a = plusGrandEntier(alpha, beta, gamma, delta, omega)
    - alpha – beta - gamma - delta - omega;

return
    (unMonome
        (coefficient (m1) + coefficient (m2),
        exposant (m1))) ;
```

6. Déclaration des variables

Les déclarations de variables se font en début de bloc, c'est-à-dire juste après une accolade ouvrante, et non pas à l'intérieur d'un bloc d'instructions, excepté pour les variables qui contrôlent une instruction **for**. Il n'est autorisé qu'une seule déclaration de variable par ligne, même si plusieurs variables sont de même type.

```
int i;
int j;
bool termine;
```

7. Commentaires

On distingue cinq types de commentaires : le commentaire d'en-tête du programme, le commentaire d'introduction d'une partie du programme, le commentaire d'en-tête d'un sous-programme, le commentaire de déclaration de variable et le commentaire de trace d'affinage.

En-tête du programme

Un programme est précédé d'un cartouche de commentaire fournissant différentes informations concernant le rôle du programme, le nom du fichier source précédé de son chemin d'accès, le langage utilisé, le nom de l'utilisateur et des auteurs ainsi que la date de création et de dernière mise à jour du fichier.

```
//*************************************************************************//
// ROLE DU PROGRAMME      | jouer au morpion contre la machine              //
//------------------------|------------------------------------------------//
// FICHIER SOURCE         | /usr/public/mesProgrammes/jouerAuMorpion.C      //
//------------------------|------------------------------------------------//
// LANGAGE                | C++                                             //
//------------------------|------------------------------------------------//
// UTILISATEUR            | 1a01algo                                        //
//------------------------|------------------------------------------------//
// AUTEUR(S)              | Emile TOTO                                      //
```

```
//                        | Josiane TITI                                          //
//------------------------|-------------------------------------------------------//
// DATE CREATION          | 10/10/2006                                            //
// DATE MISE A JOUR       | 12/10/2006                                            //
//************************************************************************//
```

Titre de début d'une partie du programme

Chaque partie du programme (inclusion des bibliothèques, déclaration des constantes, définition des types enregistrement, définition des types tableau, spécification des sous-programmes, définition du programme-principal et définition des sous-programmes) est introduite par un titre.

```
//-------------------------------------------------------------------------
// IMPORTATION DES BIBLIOTHEQUES UTILISEES
//-------------------------------------------------------------------------
…

//-------------------------------------------------------------------------
// DECLARATION DES CONSTANTES
//-------------------------------------------------------------------------
…
```

En-tête d'un sous-programme

L'en-tête d'un sous-programme est précédé d'un cartouche explicitant le plus exactement possible le rôle du sous-programme. Ce cartouche mentionne systématiquement les paramètres formels du sous-programme, ainsi que le résultat retourné s'il s'agit d'une fonction. Signaler aussi les exceptions levées par le sous-programme ainsi que les post et pré-conditions s'il y a lieu.

```
// recherche la position de la première occurrence de l'élément x
// dans la tranche de tableau tab [1..n] ;
// si l'élément existe, trouvé = VRAI et rang désigne la place
// de l'élément dans le tableau tab,
// sinon trouvé = FAUX et rang est indéfini
void rechercherOccurrence
    (const TabEntiers tab, const int n, const int x,
    bool &trouve, int &rang);

// calcule n!
int factorielle(const int n);
```

Déclaration de variable

Le rôle d'une variable qui vient d'être déclarée doit aider à la compréhension de l'algorithme et indiquer les liens avec les paramètres, les autres variables ... Le commentaire relatif au rôle de la variable est placé à droite sur la même ligne que la déclaration de celle-ci.

Par souci de lisibilité, on essayera d'aligner les commentaires des déclarations de variables d'un même sous-programme.

```
int i;           // indice de parcours du tableau tab

bool termine;    // pour stopper la recherche de l'élément x
```

Trace d'un affinage

Le corps de tout sous-programme C++ y compris la fonction principale *main()* inclut la trace de son affinage en commentaires. Le commentaire doit refléter la conception du sous-programme, sans paraphraser le code.

Instructions et commentaires sont alignés ; ils ne sont pas mélangés sur la même ligne.

Une mise en page correcte du code C++ doit respecter la règle d'écriture dite *règle des paragraphes*. Elle est la même que celle adoptée en langage algorithmique et est rappelée ci-dessous :
- les paragraphes s'enchaînent et s'imbriquent selon les structures de contrôle utilisées dans le code. Tout paragraphe est composé d'un commentaire spécifiant une action de l'algorithme (*quoi* ou titre du paragraphe) immédiatement suivi de la mise en œuvre de l'action (*comment* ou contenu du paragraphe).
- par définition, un paragraphe débute par un commentaire. Il se termine soit au début du paragraphe suivant de même niveau d'indentation, soit au prochain niveau d'indentation inférieur au niveau courant (caractérisé par un mot-clé ou le début du paragraphe suivant), soit à la fin du texte.
- un programme ne respectant pas cette mise en page est mal documenté, la relation *quoi-comment* souhaitée en termes de traces d'affinage y étant caduque. On notera que la simple recopie, au sein d'un programme, de l'affinage d'une action ne permet pas toujours d'obtenir une mise en page correcte ; il conviendra alors de modifier le texte en conséquence.

```
...
void rechercherOccurrence ...
{
    int i;       // indice de parcours du tableau tab

    // se positionner sur le premier élément du tableau tab
    i = 1;
    while (i <= n && tab[i] != x)
    {
        // passer à l'élément suivant du tableau tab
        i++ ;
    }
    //  fournir le résultat de la recherche
    if (i <= n)
    {
        trouve = true;
        rang = i;
    }
    else
    {
        trouve = false;
    }
}
```

8. Instructions

Une ligne contient au plus une instruction.

Séquence

Les actions d'une séquence sont alignées.

```
i = 1;
j = 1;
termine = false;
rechercherOccurrence(tab, N, x, trouve, rang);
```

Sélection

Les mots clés **if**, **else** sont alignés. La condition **if (…)** et le mot clé **else** occupent chacun une ligne. Les actions d'une sélection sont alignées avec une indentation de 4 espaces par rapport aux mots clés **if** et **else**. Même si l'action associée au *alors* ou au *sinon* du **if** n'est composée que d'une seule instruction, elle sera encadrée par un bloc { } aligné avec le **if**.

```
if (condition)
{
    action1;
}
else
{
    action2;
}
```

Pour la sélection réduite, adopter le format suivant :

```
if (condition)
{
    action;
}
```

Sélection multiple

Les expression **switch** (…), **case** et **default** occupent chacun une ligne. Les mots clés **case** et **default** sont alignés avec une indentation de 4 espaces par rapport au mot clé **switch**. Les actions associées à une sélection sont alignées avec une indentation de 4 espaces par rapport au mot clé **case** ou **default**. La clause **default** n'est pas mentionnée si cette dernière ne comporte pas d'action.

```
switch (expression)
{
    case constante1 :
        action1;
    case constante2 :
        action2;
    …
    case constanteN :
        actionN;
}
```

Répétition *while*

La condition **while** (…) occupe une ligne. Les actions d'une répétition sont alignées avec une indentation de 4 espaces par rapport au mot clé **while**. Même si l'action associée au **while** n'est composée que d'une seule instruction, elle sera encadrée par un bloc { } aligné avec le **while**.

```
while (condition)
{
    action;
}
```

Répétition *do … while*

Les mots clés **do** et **while** (condition) sont alignés. Le mot clé **do** et la condition **while** (…) occupent chacun une ligne. Les actions d'une répétition sont alignées

avec une indentation de 4 espaces par rapport au mot clé **do**. Même si l'action associée au **do** n'est composée que d'une seule instruction, elle sera encadrée par un bloc { } aligné avec le **do**.

```
do
{
    action;
}
while (condition);
```

Répétition *for*

La condition **for** (…) occupe une ligne. Les actions d'une répétition sont alignées avec une indentation de 4 espaces par rapport au mot clé **for**. Même si l'action associée au **for** n'est composée que d'une seule instruction, elle sera encadrée par un bloc { } aligné avec le **for**. Si l'expression du **for** ne peut pas contenir sur une même ligne (80 caractères), appliquer les règles de coupure fournies au §4.

```
for (expression1 ; expression2 ; expression3)
{
    action;
}
```

9. Recommandations

9.1. Taille des unités de programme

Le corps d'un sous-programme ne doit pas, en règle générale, excéder 40 lignes. Décomposer le traitement en sous-programmes sinon.

9.2. Taille des lignes

Un programme doit pouvoir se lire de la même façon sur écran et sur papier (listing). De ce fait, une ligne de programme ne doit pas excéder 80 caractères (*cf. §5 coupures*).

9.3. Constantes

Les constantes numériques n'apparaissent pas directement dans le code et sont remplacées par un identificateur de constante. Ainsi, la déclaration :

```
typedef int TabEntiers[10];
```

doit s'écrire :

```
const int N = 10 ;
typedef int TabEntiers[N];
```

De même, écrire en utilisant la constante N :

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

9.4. Sélections imbriquées

Lorsque plusieurs conditions portent sur une même variable de choix et que ces conditions s'excluent mutuellement, préférer l'utilisation d'un **switch** à des sélections imbriquées.

Par exemple, préférer :

```
switch (c)
{
    case 'A' :
        …
    case 'B' :
        …
    case 'C' :
        …
}
```

à :

```
if (c == 'A')
{
    ...
}
else
{
    if (c == 'B')
    {
        ...
    }
    else
    {
        if (c == 'C')
        {
            ...
        }
        else
        {
            // autres valeurs de c
            ...
        }
    }
}
```

9.5. Retour de fonction

Lorsque c'est possible, n'écrire qu'un seul **return** par fonction, en général placé en tant que dernière action de la fonction. Sinon, s'assurer que toute branche d'exécution de la fonction se termine par un **return**.

Abréger certaines écritures en exprimant directement la valeur retournée par la fonction. Ainsi, transformer les deux actions :

```
c = a + b;
return (c);
```

en un seul **return** :

```
return (a + b);
```

De même, l'écriture :

```
if (a == b)
{
    return (true);
}
else
{
    return (false);
}
```

doit se simplifier en :

```
return (a == b);
```

Ne jamais mentionner de **return** à l'intérieur d'une répétition. C'est la condition du tantque qui doit exprimer la sortie de la boucle. Ainsi, l'écriture :

```
while (!trouve)
{
    if (i == NB)
    {
        return (false);
    }
    else
    {
        …
    }
}
```

doit se transformer en :

```
while (i < NB && !trouve)
{
    …
}
if (i == NB)
{
    return (false);
}
else
{
    …
}
```

9.6. Utilisation des booléens

Abréger certaines écritures lorsque des booléens sont utilisés dans une condition.

Préférer :

```
if (trouve)
```

à :

```
if (trouve == true)
```

Préférer :

```
if (!trouve)
```

à :

```
if (trouve == false)
```

9.7. Boucles contrôlées par une variable

Lorsqu'une boucle est contrôlée par une ou plusieurs variables de contrôle, préférer systématiquement l'utilisation d'un **for** à celle d'un **while**.

Par exemple, préférer :

```
for (int i = 0 ; i < n ; i++)
{
    for (int j = 0 ; j <n ; j++)
    {
        ecrire(matrice[i][j]);
    }
}
```

à :

```
// se positionner sur la première ligne
i = 0;
while (i < n)
{
    // se positionner sur la première colonne de la ligne
    j = 0;
    while (j < 0)
    {
        ecrire(matrice[i][j]);
        // passer à la colonne suivante de la ligne
        j++;
    }
    // passer à la ligne suivante
    i++;
}
```

Ne pas forcer la condition de sortie d'une répétition en affectant intempestivement sa ou ses variables de contrôle. Réécrire la condition de sortie de la répétition, en introduisant si besoin une condition composée.

9.8. Déclenchement d'exception

Autant que possible, détecter les exceptions le plus tôt possible et les intercepter au travers d'une sélection réduite, le traitement standard continuant en séquence.

```
// retourne la racine carrée de l'entier a ;
// lève l'exception "IMPOSSIBLE" si a < 0
int racineCarrée(const int a) throw (Chaine)
{
    int racine;      // racine carrée de a
    if (a < 0)
    {
        throw uneChaine("IMPOSSIBLE");
    }
    // calculer la racine carrée
    ...
    return (racine);
}
```

10. Exemple

Le code qui suit est un extrait d'un programme de jeu de morpion, rédigé en respectant les règles précédemment énoncées. Tous les sous-programmes n'ont pas été présentés pour alléger le texte.

```
//*************************************************************************//
// ROLE DU PROGRAMME      | jouer au morpion contre la machine              //
//------------------------|-------------------------------------------------//
// FICHIER SOURCE         | /usr/public/mesProgrammes/jouerAuMorpion.C      //
//------------------------|-------------------------------------------------//
// LANGAGE                | C++                                             //
//------------------------|-------------------------------------------------//
// UTILISATEUR            | 1a01algo                                        //
//------------------------|-------------------------------------------------//
// AUTEUR(S)              | Emile TOTO                                      //
//                        | Josiane TITI                                    //
//------------------------|-------------------------------------------------//
// DATE CREATION          | 10/10/1999                                      //
// DATE MISE A JOUR       | 12/10/2009                                      //
//*************************************************************************//


//-------------------------------------------------------------------------
// IMPORTATION DES BIBLIOTHEQUES UTILISEES
//-------------------------------------------------------------------------
#include "/usr/local/public/BIBLIOC++/entreeSortie.h"
#include "/usr/local/public/BIBLIOC++/chaine.h"


//-------------------------------------------------------------------------
// DECLARATION DES CONSTANTES
//-------------------------------------------------------------------------
const int N = 10;               // ordre de la matrice carrée

//-------------------------------------------------------------------------
// DEFINITION DES TYPES TABLEAU
//-------------------------------------------------------------------------
typedef char Carre[N][N];       // définition d'une matrice carrée d'ordre N

.....



//-------------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMES
//-------------------------------------------------------------------------
// retourne VRAI si tous les caractères d'une rangée du tableau damier
// d'ordre n sont égaux au caractère c, et FAUX sinon ;
// une rangée est soit une ligne i, soit une colonne j, soit la 1ère diagonale,
// soit la 2ème diagonale
bool rangeeGagnante
    (const Carre damier, const int n, const int i,
    const int j, const char c);

// affecte une case du damier d'ordre n par le caractère c,
// en demandant au joueur (autant de fois que nécessaire)
// les numéros de ligne et de colonne d'une case ;
// déclenche l'exception "DEBORDEMENT" si le numéro de ligne ou de colonne
// n'est pas compris entre 1 et n ;
// fournit l'indice i de la ligne et l'indice j de la colonne
// de la case affectée
void choisirCase
    (Carre damier, const int n, const char c,
    int &i, int &j) throw (Chaine);
```

```
……

//-----------------------------------------------------------------------------
// PROGRAMME PRINCIPAL
//-----------------------------------------------------------------------------
void main()
{
    Carre damier;       // damier de jeu d'ordre N
    int i;              // indice d'une ligne du damier
    int j;              // indice d'une colonne du damier
    int joueur;         // joueur courant (=0 si machine, =1 si joueur)
    int nbCases;        // nombre de cases occupées à un instant du jeu
    bool gagne;         // indique si un des 2 joueurs a gagné

    try
    {
        // préparer le jeu
        initialiserDamier(damier, N);
        afficherDamier(damier, N);
        // choisir comme premier joueur la machine
        joueur = 0;
        nbCases = 0;
        gagne = false;
        while (nbCases < N * N && !gagne)
        {
            if (joueur == 0)
            {
                // chercher une case libre dans le damier
                chercherCase(damier, N, 'X', i, j);
                gagne = rangeeGagnante(damier, N, i, j, 'X');
            }
            else
            {
                // choisir une case libre du damier
                choisirCase(damier, N, 'O', i, j);
                gagne = rangeeGagnante (damier, N, i, j, 'O');
            }
            afficherDamier(damier, N);
            // changer de joueur
            if (!gagne)
            {
                joueur = (joueur + 1) % 2;
            }
            nbCases++;
        }
    }
    catch (const Chaine exception)
    {
        ecrireNL(exception);
    }
}

//-----------------------------------------------------------------------------
// SOUS-PROGRAMMES
//-----------------------------------------------------------------------------
bool rangeeGagnante
    (const Carre damier, const int n, const int i,
    const int j, const char c)
{
    return
        (ligneGagnante(damier, n, i, c)
        || colonneGagnante(damier, n, j, c)
        || premiereDiagonaleGagnante(damier, n, c)
        || deuxièmeDiagonaleGagnante(damier, n, c));
}

void choisirCase
    (Carre damier, const int n, const char c,
    int & i, int & j) throw (Chaine)
{
```

```
    bool trouve;   // pour stopper le dialogue

    trouve = false;
    while (!trouve)
    {
        // lire les indices ligne et colonne de la case
        // et s'assurer de leur validité
        lire(i);
        lire(j);
        if (i < 1 || i > n || j < 1 || j > n)
        {
            throw uneChaine ("DEBORDEMENT");
        }
        // examiner si la case est libre
        if (damier[i][j] == ' ')
        {
            damier[i][j] = c;
            trouve = true;
        }
    }
}

……
```