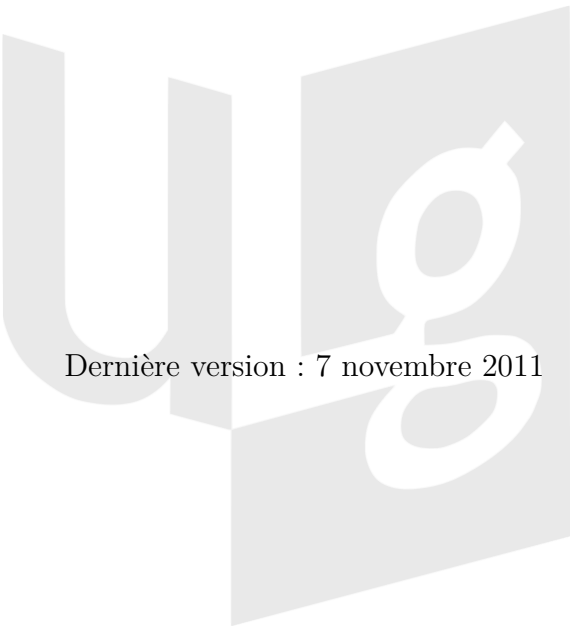
Résolution d'exercices avec Visual Studio

Document relatif au cours d'*Introduction à la Programmation*

Document rédigé par David Taralla
3e Bachelier en Sciences Informatiques
david.taralla@student.ulg.ac.be



Dernière version : 7 novembre 2011

Préambule

Ce petit guide a été écrit dans le but de vous faciliter l'utilisation du programme Microsoft Visual Studio 2010 dans le cadres des répétitions du cours d'Introduction à la Programmation. Lors de la première répétition, il y a eu quelques problèmes de démarrage en raison d'incompatibilités entre ce qui a été vu au cours théorique et ce qui se passe en pratique avec ce programme.

Vous avez vu au cours théorique le langage `C` dans sa norme *c99*, alors que le compilateur utilisé en répétition est en version *c89*. Certains problèmes de la compilation seront donc « normaux » (comme le problème que vous avez tous eu avec la fonction `scanf`). Dans tous les cas, si vous ne parvenez pas à compiler ou si vous avez des questions, n'hésitez pas à appeler un moniteur qui viendra vérifier votre code et vous aider.

Bon travail !

Table des matières

1 Style de programmation

Le style de programmation *(présentation du code)* est très important dans le monde de l'informatique. Il permet de relire le code beaucoup plus facilement, ce qui est aussi valable pour ceux qui le liront par après (comme le correcteur de votre examen).

Plus votre code sera propre et *auto-documenté* (présence de commentaires pour expliquer les passages compliqués, noms de variables exprimant bien le rôle de celles-ci,...), plus celui qui le lira sera prompt à avoir une bonne impression de votre travail.

Enfin, **soyez cohérents avec vous-même** ! Choisissez un style (celui qui est proposé en Section 1.1 par exemple) et **respectez-le tout au long de votre programme** ! !

Voici quelques règles d'hygiène de code qu'il vous faudrait à tout prix respecter.

1.1 Exemple de style (conseillé)

1.1.1 Opérateurs binaires

Tous les opérateurs binaires ont un espace de part et d'autre.

```
d=(((a+b-c)*5)%3)<<4;                    // TRÈS mauvais !
resultat = (((a + b - c) * 5) % 3) << 4; // Correct

---

if (a==(b/c)&&(d!=0||(d%3)==0)) {              // TRÈS mauvais !
if (a == (b / c) && (d != 0 || (d % 3) == 0)) { // Correct
```

1.1.2 Opérateurs unaires

Les opérateurs unaires ne sont pas entourés d'espaces.

```
int *i = (int *)malloc(5 * sizeof(int)); // Mauvais
int* i = (int*)malloc(5 * sizeof(int));  // Correct

---

int * a = & i; // Mauvais
int* a = &i;   // Correct

---
```

```
++ i; // Mauvais
++i;  // Correct
```

1.1.3 Branchement conditionnel

```
// Programme ...

if (condition1) {
  instruction1;
  instruction2;
  ...
}
else if (condition2) {
  instruction1;
  instruction2;
  ...
}
else {
  instruction1;
  instruction2;
  ...
}

// Programme ...
```

Différents points de style à remarquer :
- les point-virgules sont accolés à l'instruction qu'ils délimitent ;
- présence d'un espace entre les mots clés et les accolades ouvrantes ;
- les accolades fermantes sont seules sur leur ligne ;
- les parenthèses sont collées à leur contenu mais **pas** aux mots clés.

Astuce : lorsque vous ouvrez une `{`, une `[` ou une `(`, **fermez-la tout de suite puis écrivez à l'intérieur**. Cela vous évitera des oublis de parenthèse/accolade, et elle sera directement bien indentée par rapport au bloc qu'elle définit.

1.1.4 Boucle *for*

Vous veillerez à limiter au maximum le nombre d'instructions dans la ligne du `for`. Habituellement, il n'y en a que trois : L'initialisation d'un « compteur », la condition sur la continuité de la boucle et le pas de la boucle (ce qui fait avancer le « compteur »).

```
// Programme...

int i; // Pour les variables de boucles destinées à compter, i suffit
for (i = 0; i < qqch; i++) {
  instruction1;
  instruction2;
  ...

  if (condition1) {
    instruction3;
    instruction4;

    if (condition2) {
      instruction5;
      ...
    }
    else {
      instruction6;
      ...
    }

    instruction7;
  }

  instruction8;
}

// Programme...
```

Différents points de style à remarquer :
- chaque bloc est indenté par rapport à son bloc parent [1]. À chaque entrée dans un bloc, on augmente la tabulation (habituellement 2 ou 4 espaces) ;
- présence d'un espacement vertical entre les blocs : cela rend le code plus clair.

1.1.5 Boucle *while*

Le mot clé `while`, sa condition et son accolade ouvrante sont **seuls** sur une ligne ! Pas question de mettre une ou plusieurs instructions directement à la droite de l'accolade.

1. L'« imbrication » des blocs

```
// Programme...

while(condition) {
  instruction1;
  instruction2;
  ...
}

// Programme...
```

1.1.6 Boucle *do... while*

Le mot clé `do` et son accolade ouvrante sont **seuls** sur une ligne ! Pas question de mettre une ou plusieurs instruction(s) directement à la droite de l'accolade. De même, l'accolade fermante, le mot clé `while`, sa condition et le point virgule de fin d'instruction sont également seuls sur une ligne.

```
// Programme...

do {
  instruction1;
  instruction2;
  ...
} while(condition);

// Programme...
```

1.1.7 Noms des variables

Il est d'une importance cruciale de bien nommer vos variables **à l'avance**. Ainsi vous savez tout de suite, à n'importe quel endroit du programme, quel est son rôle et ce qu'elle représente.

Les noms des variables qui ne sont pas des constantes commencent toujours par une **minuscule** et se poursuit en *camel-case* (Ex. : « valeurRetour »).

1.1.8 Exemple

Voici un exemple de programme calculant la somme des N premiers nombres strictement positifs ($N > 0$) et affichant chacune de ses étapes. La première version est celle, horrible, qu'il ne faut surtout pas écrire. La seconde est une bonne manière de le faire.

```
// Exemple 1 : à éviter!
int main() {
int a,i,N=5;




for(i=1,a=0 ;i<=N;i++) {a=a+i;
 printf("%d",a);
    }
return 0;}


// Exemple 2 : correct!
int main() {
  // Initialisation des variables
  int N = 5;
  int i;
  int sommePartielle = 0;

  // Calcul de chaque terme et affichage
  for(i = 1; i <= N; i++) {
    sommePartielle = sommePartielle + i;
    printf("Terme %d : %d\n", i, sommePartielle);
  }

  // Fin du programme
  return 0;
}
```

2 Résoudre les TP avec Visual Studio 2010

Démarrez Visual Studio. L'écran d'accueil apparaît.

2.1 Passer à un nouvel exercice

Lorsque vous commencez un nouvel exercice, il est préférable de créer un nouveau *projet*. Pour ce faire, allez sous *Fichier* > *Nouveau* > *Projet...*. Un dialogue apparaît : Cliquez sur *Visual C++* en haut à gauche, et sélectionnez dans le colonne centrale *Application console Win32*. Entrez ensuite le *Nom* de votre projet

(Par exemple, `TP1_ex1`). Il doit être différent de ceux que vous avez déjà créés précédemment, la notation donnée en exemple est donc intéressante dans cette optique. Enfin, cliquez sur *OK*.

Un nouveau dialogue apparaît. Cliquez sur *Suivant*. Ensuite, sur ce nouvel écran, sélectionnez bien (même si cela devrait être les choix par défaut) :
- Application console
- Projet vide (**décoché**)
- En-tête précompilé (**coché**)

Cliquez finalement sur *Terminer*.

Vous avez presque terminé. Remplacez, pour plus de compatibilité avec le cours théorique, le code

```
int _tmain(int argc, _TCHAR* argv[])
```

par, comme vous l'avez appris,

```
int main()
```

Surtout, **surtout**, ne supprimez pas la ligne *#include "stdafx.h"*. Visual Studio en a besoin pour compiler, et elle doit rester la première ligne de votre programme. Maintenant, vous pouvez commencer votre vrai travail : répondre à la question !

N'oubliez pas, avant de créer tout nouveau projet, de ***sauvegarder*** *votre projet en cours (*Fichier > Enregistrer tout*).*

2.2 Ouvrir un projet existant

À l'examen, vous pourriez avoir envie d'aller revoir un de vos codes passés. Pour ce faire, allez sous *Fichier > Ouvrir > Projet, solution....* Cherchez le dossier qui porte le nom du code qui vous intéresse (`TP1_ex1` par exemple), ouvrez-le, puis sélectionnez le fichier solution (pour notre exemple : `TP1_ex.sln`) avant de cliquer sur *Ouvrir*.

2.3 Compiler son programme

Pour compiler votre programme, appuyez une première fois sur `F5`. Si votre code est correct (et s'il n'y a pas de souci de compatibilité), la fenêtre peut s'ouvrir et se refermer très vite alors qu'elle devrait vous afficher un résultat : c'est normal. Maintenant, faites `CTRL + F5` : le programme demandera une entrée quelconque avant de se terminer automatiquement. Vous aurez ainsi tout le loisir de voir si votre code a retourné un résultat correct... ou non !

3 Problèmes de compatibilité c89 & c99

Les problèmes suivants sont connus (la liste sera mise à jour si besoin est) :

- Il faut ignorer les avertissements du compilateur concernant *'scanf' : This function or variable may be unsafe.*.
- Une seconde utilisation de la fonction `scanf` ne redemande rien à l'utilisateur. À la place, celle-ci reprend ce qui existait déjà dans le *tampon* (i.e. la réponse précédente) et n'attend pas que l'utilisateur ait donné une information.
  Un moyen simple de résoudre ce problème : juste avant l'appel du second `scanf`, mettez la ligne `fflush (stdin);`. Vous devriez alors être en mesure d'observer le comportement attendu du programme.
- Vous devez impérativement déclarer toutes vos variables au début de chaque fonction (`main` comprise).
- Le type `bool` n'est pas utilisable, et les mots-clés `true` et `false` non plus. Utilisez plutôt le type `int`, la valeur `0` pour `false` et toute autre valeur positive pour `true`.
- Lors des allocations avec `malloc`, vous devez bien faire
  ```
  type* x = (type*)malloc(taille);
  ```
  Et non
  ```
  type* x = malloc(taille);
  ```