PARIS DESCARTES IUT

DUT Informatique - 1ère année

IAP – Introduction à l'Algorithmique et à la Programmation

TRAVAUX PRATIQUES – Semaine n°1

Thèmes

- Prise en main de Visual Studio 2017
- Expressions, types de donnés, algorithmes simples

Lors de première séance de TP, vous traiterez dans l'ordre les exercices du TP de la semaine.

Lors de la deuxième séance de TP et suivant votre avancement, vous devez continuer à passer sur machine les exercices de TP qui ne l'ont pas été en première séance. Une bonne pratique est de tester sur machine les exercices de TD. Cette semaine, vous passerez sur machine et testerez le code en langage C correspondant aux exercices 4, 5, 6 et 7 du TD.

Préambule. Prise en main de Visual Studio

Lancez Visual Studio 2017 (`démarrer -> Tous les programmes -> Environnement de développement -> Microsoft Visual 2017 -> Microsoft Visual 2017`).

Choisissez les paramètres d'environnement par défaut correspondant au `développement Visual C++`.

Créez un nouveau **projet** (`Fichier -> Nouveau -> Projet…`) :

- Modèle de projet : `Visual C++ -> Général`
- Type de projet : `Projet vide`
- Donnez un nom à votre projet. `TP1-Exo1 est` raisonnable.
- Changez son emplacement (les fichiers doivent être **sur votre dossier réseau** et `Z:\IAP` est la racine de ce dossier).
- Donnez un nom à la ***solution*** de votre projet (une *solution* permet de regrouper plusieurs projets). `IAP-Sem1 ou IAPs1` sont raisonnables.
- Laissez cochée la case « `Créer un nouveau répertoire pour la solution` ».

Lorsque vous créerez de nouveaux projets, Visual Studio vous donnera la possibilité de les ajouter à la solution courante. Nous vous recommandons de réunir tous les projets d'IAP de la même semaine dans une même solution.

Les programmes que vous écrirez seront composés (du moins les deux premières semaines) d'une unique fonction `main`. Vous pourrez au choix créer un projet par exercice ou bien faire un seul projet pour toute la séance et y ajouter un bloc

d'instruction entre accolades { } pour chaque exercice. Chaque bloc dispose de ses propres variables : vous pouvez donc utiliser le même nom de variables dans plusieurs blocs. Après avoir testé un bloc, vous pourrez le masquer par un commentaire (`/*…*/`) pour éviter une réexécution ultérieure.
Avant de pouvoir programmer, vous devez créer un fichier C au sein de votre projet (`Projet -> Ajouter un nouvel élément…`). Choisissez le type de fichier `Fichier C++` et donnez-lui un nom **finissant impérativement par «`.c`»**. Comme vous aurez pour le moment un seul fichier par projet vous pouvez lui donner le même nom que le projet avec **«`.c`»** à la fin.

Une fois le programme écrit, compilez-le (`Générer -> <votre nom de projet>` ou tapez la touche `F7` ce qui génère toute la solution) et corrigez (si nécessaire) toutes les erreurs. Exécutez votre programme (`Déboguez -> Exécutez sans débogage` ou frappez la touche `F5`).

Exercice 1. Premier programme

Écrivez un programme qui affiche « Bonjour a tous » sur une ligne puis « Ca va ? » sur la ligne suivante.

Exercice 2. Types – taille et domaine de variation des données

Affichez pour chaque type élémentaire du langage C, la taille mémoire (en nombre d'octets) prise par une variable de ce type et le domaine de validité de ce type. Le domaine est caractérisé par une borne minimale et une borne maximale. Ces bornes sont définies (par des macro-instructions) dans le fichier d'entête `"limits.h"`, ajoutez l'entête correspondantes `#include" limits.h"` avant la fonction `main()`. Les constantes que vous devez employer pour chacun des types sont données dans le tableau ci-dessous.

Pour afficher la taille des types, vous utiliserez l'opérateur `sizeof(T)` (pour le type `T`). La borne minimale des types `unsigned` n'est pas définie dans `limits.h`, pourquoi ? Vous afficherez la borne à partir du littéral entier correspondant (constante entière). Pour les type `float` et `double`, vous afficherez simplement leur taille.

Type	*Borne min.*	*Borne max.*
`char`	`CHAR_MIN`	`CHAR_MAX`
`unsigned char`	*Non définie*	`UCHAR_MAX`
`short`	`SHRT_MIN`	`SHRT_MAX`
`unsigned short`	*Non définie*	`USHRT_MAX`
`int`	`INT_MIN`	`INT_MAX`
`unsigned int`	*Non définie*	`UINT_MAX`
`long`	`LONG_MIN`	`LONG_MAX`
`unsigned long`	*Non définie*	`ULONG_MAX`
`float`	*Non définies*	
`double`	*Non définies*	

Votre programme devra provoquer l'affichage suivant :

```
Types : coût-mémoire (en octets) et domaine de variation
char : 1 - Domaine : -128 .. 127
unsigned char : 1 - Domaine : 0 .. 255
short : 2 - Domaine : -32768 .. 32767
unsigned short : 2 - Domaine : 0 .. 65535
int : 4 - Domaine : -2147483648 .. 2147483647
unsigned int : 4 - Domaine : 0 .. 4294967295
long : 4 - Domaine : -2147483648 .. 2147483647
unsigned long : 4 - Domaine : 0 .. 4294967295
float : 4
double : 8
```

Note : Pour un bon affichage des domaines de variation, vous utiliserez les formats : `%u` pour les `unsigned int`, `%ld` pour les `long`, `%lu` pour les `unsigned long`.

Exercice 3. Switch

Un médecin de campagne peut être selon le jour de la semaine être : présent (du lundi à vendredi), en congé (le dimanche) ou d'astreinte (le samedi).

Selon le numéro du jour dans la semaine, affichez le statut du médecin ou un message d'erreur si le numéro du jour est incorrect. Le lundi est le jour 1.

Utilisez l'instruction switch.

Exercice 4. double et float sont différents

Lisez deux nombres réels (un `double` et un `float`) et affichez leur moyenne.

Indiquez `%f` pour saisir un `float` avec `scanf` et `%lf` pour saisir un `double` sous peine de récupérer une valeur totalement différente !

Par contre pour l'affichage avec `printf`, `%f` correspond à un double (mais un `float` est également correctement affiché).

Exercice 5. Conditionnelles et tests sur des réels

Affichez le plus grand parmi trois nombres réels. Au choix vous les saisirez ou les initialiserez avec différentes valeurs (et dans ce dernier cas vous recompilerez votre projet) pour tester votre programme sur plusieurs cas.

Utilisation de l'assertion pour les tests

Au lieu d'afficher le résultat vous pouvez écrire une assertion de la forme

```
assert(max == 5.2) ; // vérifie que max vaut bien 5.2
```


L'inclusion <`assert.h`> devra alors être ajoutée avant la fonction `main()`.

Une assertion interrompt l'exécution du programme avec un message d'erreur si elle est fausse.

Hélas, pour comparer deux réels il est vivement conseillé de prévoir une marge d'erreur et d'écrire plutôt :

```
double marge = 0.0001 ; //par exemple
```

|max-5.2| < marge (1)

Rappel : |x|= -x si x <0 sinon x.

Exprimez (1) sous la forme d'une expression booléenne.

Implantation des exercices du TD et test des solutions

Implantez les exercices du TD en machine et testez-les dans la mesure du possible sur l'ensemble des valeurs (Jeu de Données de Test - JDT) représentant les différents cas des algorithmes.

Pour cela, plusieurs solutions :

a) Soit vous saisissez des valeurs différentes lors de plusieurs exécutions.

b) Les variables ne sont pas saisies mais initialisées (en dur) dans le code-source. Pour tester d'autres données de test, les variables seront initialisées différemment et le programme recompilé avant chaque exécution.

Dans ces deux premiers cas, la vérification se fera dans le code par un affichage des résultats aussi explicite que possible

c) La vérification pourra se faire par assertion (cf. exercice 5) sans besoin d'affichage. Ce type de test est appelé **test unitaire**.