TP 1 Programmation – DUT 1 – Premières procédures

P.Courtieu

Septembre 2017

Résumé

On a appris à compiler un programme avec deux fichiers (dont un déjà fourni). On écrit des procédures simples sans puis avec paramètres. On teste les affichage produits par ces procédures.

On aborde la conditionnelle.

1 Procedures

1.1 Prise en main, Rappel

Comme au TD 1, effectuez les opérations suivantes :

— Démarrez une session linux et lancer un terminal (menu pricipal : `terminal/Konsole`).

— *Si nécessaire* créez le répertoire `tp1`.

```
mkdir tp1
```

— Placez vous dans le répertoire `Tp1` :

```
cd tp1
```

— Téléchargez les fichiers inout.h et inout.c et mettez les dans un répertoire `tp1`. Pour cela vous pouvez faire les commandes suivantes dans un terminal :

```
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.h"
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.c"
```

— Dans le même terminal, effectuez la commande suivante pour compiler le fichier inout.c :

```
gcc -c inout.c
```

— Ouvrez l'éditeur `kwrite` (menu principal `editeur/kwrite`) ou tout autre éditeur de votre choix, SAUF LES IDEs de type Eclipse/Netbeans.

— Ouvrez un nouveau fichier `tp1.c` dans le même répertoire `tp1`. **Vous écrirez toutes vos procédures dans ce fichier, chaque procédure devra être testée. N'effacez pas vos procédures une fois finies et testées.**

— Lorsque vous voulez compilez ce fichier :

```
gcc -c tp1.c
```

— Fabriquez un exécutable :

```
gcc inout.o tp1.o -o tp1
```

— Exécutez le le pogramme obtenu :

```
./tp1
```

1.2 Comment testez vos procédures ?

Programmez et tester les procédures *une par une* (testez une procédures dès que vous pensez qu'elle est finie). Les procédures de ce TP n'ont aucun effet à part les affichages, donc il faut vérifier sur l'écran que les messages qui s'affichent sont corrects. TESTEZ PLUSIEURS CAS, ce n'est pas parce-qu'un appel semble fonctionner que votre procédure est correcte.

Pour tester une procédure `proc(...)` écrivez une autre procédure `testproc()` dans laquelle vous effectuez un ou plusieurs appels à `proc`. Lorsque vous désirez lancer le test, ajoutez l'instruction `testproc();` dans la fonction `main`, compilez, lancez l'exécutable, vérifiez les affichages.

N'effacez pas vos procédures de test ! Elle peuvent servir à nouveau si vous devez corriger/améliorer vos procédures. En revanche vous pouvez effacer l'appel aux tests dans le `main`, il sera facile de le remettre plus tard. Dans l'exemple ci-dessous la procédure `testecritSommeDeuxInt` commence par afficher un message permettant de savoir quelle procédure est testée, puis chaque appel de la méthode est annoncé avant d'être appelée.

Exemple :

```
#include"inout.h"

void ecritSommeDeuxInt(int x, int y){
   ...
}

void testecritSommeDeuxInt(){
  ecrireString("** Début test ecritSommeDeuxInt:");
  ecrireString("** (2,5): ");
  ecritSommeDeuxInt(2,5); // devrait afficher 7
  ecrireSautDeLigne();
  ecrireString("** (-2,2): ");
  ecritSommeDeuxInt(-2,2); // devrait afficher 0
  ecrireString("** Fin test ecritSommeDeuxInt:");
}

void main(void){
  testecritSommeDeuxInt();
}
```

1.3 Procédures

1. **void** `ecritBonjour() { ... }` qui écrit à l'écran le texte ci-dessous :

   ```
   Bonjour
   ```

2. **void** `ecrit3Bonjour() { ... }` qui écrit à l'écran le texte ci-dessous :

   ```
   Bonjour
   Bonjour
   Bonjour
   ```

   Vous pouvez faire appel à la procédure précédente.

3. **void** `ecritSommeDeuxlireInt()` qui écrit à l'écran la somme de deux entiers lus au clavier (avec deux appels à `lireInt()`).

4. **void** `ecritDeuxFoislireInt()` qui écrit deux fois à l'écran un entiers lu au clavier (avec un seul appel à `lireInt()`). Il est nécessaire d'utiliser une variable.

5. **void** `ecritSommeDifflireInt()` qui écrit la somme puis la différence (sur deux lignes consécutives) de deux entier lus au clavier.

1.4 Procédures avec arguments

Vous testerez plusieurs appels à la fois de ces procédures dans votre `main`.

1. **`void`** `ecritDeuxInt(`**`int`** `x, `**`int`** `y)` qui écrit les deux entiers passés en paramètres.
2. **`void`** `ecritSommeDeuxInt(`**`int`** `x, `**`int`** `y)` qui écrit à l'écran la somme des deux entiers passés en paramètres.
3. **`void`** `ecritSommeTroisInt(`**`int`** `x, `**`int`** `y, `**`int`** `z)` qui écrit à l'écran la somme des trois entiers passés en paramètres.
4. **`void`** `ecritSommeDeuxIntBavard(`**`int`** `x, `**`int`** `y)` qui écrit les deux entiers passés en paramètres puis leur somme.

2 La conditionelle `if`

On peut dans un programme définir deux séquences d'instructions différentes en fonction du résultat d'un test à l'exécution. L'instruction conditionnelle comporte trois morceaux : le test, la partie à exécuter si le test est vrai à l'exécution (« then ») et la partie à exécuter si le test est fausse à l'exécution (« **`else`** »). Cette dernière partie (**`else`**) est facultative.

Voici comment on écrit une conditionnelle en C (et java).

```
if (test) {
  ... // instructions si test vrai
}
else {
  ... // instructions si test faux
}
```

3 Exemple d'utilisation

```
1 int x;
2 x = lireInt();
3 if (x >=0) {
4   ecrireString("nombre positif ou nul.");
5 }
6 else {
7   ecrireString("nombre strictement négatif.");
8 }
9 ecrireString("\n");
```

À l'exécution ce programme se comporte comme suit :

2. attend que l'utilisateur tape un entier au clavier et le stocke dans la variable `x` ; puis :

3 à 8. si `x` est plus grand ou égal à zéro (`x >= 0`) alors (l. 4) le texte « `nombre positif ou nul.` » est affiché à l'écran ; sinon (l. 7) le texte « `nombre strictement négatif.` » est affiché ; puis :

9. (ligne 9) : Enfin quoiqu'il arrive un saut de ligne est affiché.

3.1 Tests possibles

En attendant de savoir définir nos propres tests, nous utiliserons les expressions de tests suivantes, où `e1` et `e2` doivent être des expressions entière (`12`, `x`, `x+3`, etc) :

— `e1 == e2` et `e1 != e2`

— `e1 <= e2` et `e1 >= e2`
— `e1 < e2` et `e1 > e2`

3.2 Procédures avec conditionnelles

1. **`void`** `ecritMax(`**`int`** `x, `**`int`** `y)` qui écrit à l'écran le plus grand de deux entiers passés en paramètres (si ils sont égaux, on en écrit un).
2. **`void`** `ecritTestPlusGrandEq(`**`int`** `x, `**`int`** `y)` qui affiche le text « <x> `plus grand que` <y> » si le premier argument est plus grand ou égal que le deuxième et « <valeur de l'arg 1> `pas plus grand que` <valeur de l'arg 2> » sinon. <x> et <y> étant remplacés par les valeur réelles des arguments à l'exécution.
   Par exemple : `ecritTestPlusGrandEq(12,13)` doit afficher « `12 n'est pas plus grand que 13` ».
3. **`void`** `ecritTestPlusPetit(`**`int`** `x, `**`int`** `y)` qui se comporte comme `ecritTestPlusGrandEq` mais en inversant le test (premier argument strictement plus petit).
4. **`void`** `ecritTrie3(`**`int`** `x, `**`int`** `y, `**`int`** `z)` qui affiche « `<x>, <y>, <z> triés` » si les trois arguments sont ordonnés par ordre croissant. On tolère les arguments consécutifs égaux. Écrire « `<x>, <y>, <z> pas triés` »sinon.
5. **`void`** `ecritMax3(`**`int`** `x, `**`int`** `y, `**`int`** `z)` qui écrit à l'écran le plus grand des trois arguments.
6. **`void`** `ecritMax4(`**`int`** `x, `**`int`** `y, `**`int`** `z, `**`int`** `t)` qui écrit à l'écran le plus grand des quatre arguments. Contrainte : N'utilisez que 3 if **`if`**.
7. **`void`** `ecritTestDeuxEgaux(`**`int`** `x, `**`int`** `y, `**`int`** `z, `**`int`** `t)` si parmi les 4 arguments au moins deux sont égaux.
8. **`void`** `ecritPlusSommeProd(`**`int`** `z, `**`int`** `t)` écrit à l'écran le plus grand entier entre la somme et le produit des deux arguments.

3.3 Menu

Écrivez un programme dans la fonction `main` qui propose les différentes fonctionnalités des procédures de la section précédente.

Le déroulement du programme doit être le suivant :

1. Affichage des opérations disponibles avec un numéro pour chaque.
2. l'utilisateur tape un entier (+ « entrée »)
3. invitation à taper le premier argument de la procédure
4. invitation à taper le deuxième argument de la procédure
5. etc
6. lancement de la procédure
7. Saut de ligne et fin de programme.

Si vous avez le temps imaginez un moyen que le programme recommence à l'étape 1 après l'étape 7 plutôt que de s'arrêter.

3.4 Fonctions (si vous avez fini avant la fin)

Si vous connaissez l'instruction **`return`** et la notion de fonction, reprogrammez les procédures de la section 2 sous forme de fonctions retournant le résultat au lieu de l'afficher. Programmez des tests pour ces fonctions.