T. Grandpierre

TP1 – IGI 2001 – 1h

Premiers pas en C

Rappel commandes Unix

- **ls** permet de lister le contenu du répertoire courant, pour plus d'informations vous pouvez utiliser l'option « -a » pour afficher tous les fichiers, ainsi il faut taper « **ls -a** »
- **mkdir NomRep** : crée un sous répertoire qui s'appellera **NomRep** dans cet exemple
- **cd NomRep** : on se déplace dans **NomRep** qui devient le répertoire courant
- .. (deux points successifs) signifie « le répertoire qui précède le répertoire courant » et **.** (le point) signifie « le répertoire courant ». Ainsi « **.** » et « **..** » peuvent être des arguments pour toutes les commandes Unix. Par exemple « **ls ..** » affichera le contenu du répertoire précédent, et « **ls . »** est identique à ls, ils affichent le contenu du répertoire courant.
- **cp fichier1 fichier2** : effectue une copie (duplication) de fichier1, cette copie s'appellera fichier2
- **cp –r Rep1 Rep2** : effectue une duplication de tout le répertoire Rep1 en un répertoire Rep2

Exercice 1 : premier programme

a) Créer un sous répertoire IGI2101 à l'intérieur duquel vous créerez le sous répertoire TP1

b) A l'aide d'un éditeur de texte standard qui ne fait pas de mis en forme (nedit, gedit, notepad++ **etc mais pas word et autre traitement de texte**) saisir votre premier programme en C, c'est-à-dire copier les lignes suivantes et sauvegarder le tout sous le nom pgm1.c (vous pouvez taper directement « nedit pgm1.c »)

```
int main( ) {
    int z=0 ;
    z=z+1 ;
    return 0 ;
}
```

c) Quittez l'éditeur et faites ls –a : vous devez voir le fichier que vous avez créé. Vous pouvez même afficher son contenu avec la commande Unix « more » en faisant « more pgm1.c »

d) Compilez ce programme source écrit en C afin de générer un programme exécutable (appelé aussi binaire) : « gcc pgm1.c »

e) Faites à nouveau « **ls –a** » : vous devez voir qu'un nouveau fichier appelé « **a.out** » a été créé, c'est le programme exécutable

f) Exécutez ce programme avec la commande suivante : « **./a.out** » (le ./ signifie le répertoire courant)

g) Il s'exécute alors mais n'affiche rien, **c'est normal !** Il déclare un entier nommé z qui est initialisé avec la valeur 0, puis incrémente z, puis "return 0" indique que le programme se termine en retournant 0 au système Unix (ce qui indique à Unix que le programme s'est terminé sans erreur)

Exercice 2 : affichage simple

1) Dupliquez le programme précédent en le nommant pgm2.c (cp pgm1.c pgm2.c)
2) Editez pgm2.c et modifiez le code pour saisir le programme suivant :

```
#include <stdio.h>
int main ( ) {
  printf("Bonjour , voici mon second programme !");
```

```
  return 0;
}
```

3) Comme précédemment compilez (gcc pgm2.c) puis exécutez (./a.out) le programme produit.

   Important : pour afficher vous venez d'utiliser printf dont la syntaxe d'utilisation n'est pas connue du compilateur. Sa syntaxe d'utilisation (le type de ses arguments et le type de variable qu'elle retourne éventuellement) est définie dans le fichier « stdio.h ». Il faut donc charger ce fichier à l'aide de « **#include<stdio.h>** » avant d'utiliser printf.

4) Dans le terminal, tapez la commande « **ls –a** » (un espace entre le s et le tiret du 6) : on constate que le fichier **a.out** correspondant à l'exercice 1 a été écrasé par ce nouveau programme.
5) Pour éviter cela, on peut indiquer au compilateur le nom du fichier exécutable qu'il va produire grâce à l'option « **-o** » qui doit être suivie du nom à donner. Ainsi pour que le nom du programme binaire soit par exemple « **pgm2** », la commande de compilation devient « **gcc pgm2.c –o pgm2** ». Essayez cette commande puis exécutez le programme avec la commande "**./pgm2**"
6) Modifiez ce programme de façon à afficher plusieurs messages de votre choix en utilisant plusieurs fois la fonction printf. Notez que pour effectuer un retour à la ligne, il suffit d'insérer le code d'un retour chariot « **\n »** (qui se dit Cariage Return en anglais, abrégé par les lettres CR) n'importe où dans la chaîne de caractère entre guillemets du printf. Une tabulation s'insère avec le code \t

Exercice 3 : affichage d'une variable

1) Dupliquez le premier programme en le nommant **pgm3.c**
2) On souhaite maintenant afficher la valeur de la variable « **z »** de l'exercice 1 en ajoutant les lignes suivantes :

```
#include <stdio.h>
int main () {
    int z=0 ;
    z=z+1 ;
    printf("La valeur de z est ");
    printf ("%d", z);
   return 0 ;
}
```

3) Compilez (**gcc pgm3.c –o pgm3**) et exécutez (**./pgm3**) ce programme : vous constaterez que les caractères « **%d »** du second printf ont été remplacé par la valeur de **z**
4) Regroupez les deux printf en un seul, compilez et vérifiez que cela fonctionne. Vous pouvez aussi vérifier qu'il est possible d'ajouter du texte derrière %d : par exemple en saisissant

```
printf("La valeur de z est %d , le programme est fini", z);
```

Exercice 4 : affichage de plusieurs variables

1) Dupliquez le programme précédent et nommé le pgm4.c
2) Saisissez et compilez le programme suivant :

```
#include<stdio.h>
int main ( ) {
  int x=10,y=20,z=0;
  z=x+y;
  printf("La somme de %d et %d est %d", x,y,z);
  return 0;
}
```

3) modifiez-le pour bien comprendre les mécanismes du printf

Exercice 4 : dessiner avec printf

1) A partie de l'un des programmes précédents, créer un programme "**pgm4-1**" qui dessine un carré de n x n étoiles (avec par exemple n=10) à l'aide **d'un seul printf("*");** (il faut donc utiliser deux boucles imbriquées et dans l'une d'elle ajouter un printf("\n"); Les boucles s'écrivent comme en JAVA, il faut prendre garde à déclarer l'itérateur en dehors du for (cela fonctionnera mais vous sera expliqué en cours),
   Int i ;
   for (i=0 ;i<10,i++) puis le code à répéter entre accolades si il y a plusieurs lignes.

   Exemple d'exécution souhaitée avec n = 5 :

   ./pgm4-1

   *****
   *****
   *****
   *****
   *****
2) A partir de pgm4-1, créez pgm4-2 qui affiche successivement tous les carrés en faisant varier n de 1 à 80 (il suffit de rajouter une boucle externe qui fait varier n de 1 à 80).

Exercice optionnel, si il vous reste du temps : Visual Studio

Cet exercice s'adresse aux élèves qui souhaitent savoir comment compiler un programme C avec Microsoft Visual Studio sous windows. Faites-le uniquement si les exercices précédents ont été effectués !!

Sachez que ça n'est pas le seul compilateur disponible sous Windows : le compilateur gcc existe aussi et est préférable dans le cadre de cette unité, voir consigne d'installation ici : http://www.esiee.fr/~grandpit/mingw-install.pdf

1) Rebootez votre PC sous windows

2) Lancez Microsoft Visual Studio 2010 et choisissez C++ s'il vous le demande
3) Cliquez sur Nouveau Projet, dans la fenêtre de gauche sélectionnez Visual C++, puis Win32
4) Dans la fenêtre centrale sélectionnez "Application Console Win32" et en bas de cette même page donnez un nom à votre projet, par exemple TP1IGI2001 (et changez éventuellement le répertoire)
5) Un assistant s'ouvre alors :
    a. Cliquez sur suivant
    b. Garder "Application console" et "En tête précompilé"
    c. Cliquez sur Terminer
6) Un projet complet est alors créé automatiquement. Tous les fichiers qui le constituent sont affichés dans la partie de gauche, le plus important étant TP1IGI2001.cpp. En double cliquant sur ce dernier, il s'affiche dans l'éditeur (partie de centrale)
7) Vous constatez qu'un programme d'exemple a été créé automatiquement, pour le compiler, allez dans le menu "Générer" puis sélectionnez "Générer la solution". La fenêtre du bas affiche alors la sortie du compilateur qui doit se terminer par "Génération : 1 a réussi, 0 a échoué …"
8) Vous pouvez alors exécuter votre programme de deux façons :
    a. Depuis Visual Studio en cliquant dans le menu "Déboguer" sur " Démarrer le débogage"
    b. ou depuis un explorateur en double cliquant sur le programme TPIGI2001.exe qui a été produit (par défaut il doit ce trouver dans "VotreCompte\Documents\Visual Studio 2010\Projects\TP1IGI2001\Debug"
9) Ce programme ouvre une fenêtre console (noire) puis la referme aussitôt : c'est normal le programme d'exemple ne fait rien..il s'agit donc maintenant de le modifier comme vu dans les précédents exercices mais avant cela :
    a) Notez que dans VisualStudio, la ligne "int main () { " est remplacée par `int _tmain(int argc, _TCHAR* argv[])` : Ne la modifiez pas, de même insérez le #include<stdio.h> après le `#include "stdafx.h"` qui est spécifique à Visual Studio.
    b) Ajouter getchar(); avant le return 0 afin que le programme attendent qu'on tape un caractère avant de fermer la fenêtre.

    ⇨ Ce sont les deux principales modifications à effectuer pour que cela fonctionne. Cependant au cours des prochains TPs, d'autres différences apparaitront et rendrons Visual Studio moins intéressant pour cette unité.