TP2 – IGI 2101 – 2h

Type de variable en C, mémoire, débogueur

BIEN COMPILER LES PROGRAMMES AVEC L'OPTION «gcc -Wall »

Exercice 1 : taille des variables

a) Créer un sous répertoire IGI2101 à l'intérieur duquel vous créerez le sous répertoire TP2

b) Ecrire un programme ***exercice1*** qui déclare et initialise (avec des valeurs quelconques) des variables de chaque type du tableau suivant, puis affiche le nom (il faut le mettre « en dur » dans le printf, pas moyen de le récupérer avec un % quelque chose), la valeur, la taille (grâce à sizeof) et l'adresse (voir cours) des types de données suivant :

Type	format de printf	Nom Var	Valeur	Taille en octet	Taille en bits	Adresse
char	%d					
unsigned char	%u					
short	%d					
unsigned short	%u					
int	%d					
Unsigned int	%u					

Note 1 : en C ANSI, les variables ne peuvent pas être déclarées n'importe où. Les déclarations des variables doivent être regroupées au tout début la fonction à partir de l'accolade ouvrante et avant la première instruction ou appel de fonction. C'est une bonne habitude à prendre car certain compilateurs C pour microcontrôleurs refuseront de compiler.

c) En observant les adresses, que constatez-vous ?
   a. Sont-elles réservées dans l'ordre croissant ou décroissant des adresses mémoire ?
   b. Sont-elles toujours contiguës ou y a-t-il parfois des octets non utilisés entre les variables ?

Exercice 2 : affichage de l'alphabet

1) Ecrire un programme **exercice3** qui affiche les 26 premières lettres de l'alphabet (une par ligne), en majuscules et en minuscule. Pour cela vous partirez de l'exemple donné en cours :
   a. une boucle qui incrémente un char initialisé avec la valeur 'A' jusqu'à la valeur 'Z'.
   b. Puis une deuxième boucle de 'a' à 'z'
2) Modifiez votre programme pour qu'il affiche en plus le code ASCII en **décimal** à la suite de chaque lettre
3) Modifiez votre programme pour qu'il affiche en plus le code ASCII en **hexadécimal** à la suite du code ASCII en décimal

4) Ecrire un autre programme ASCII (à partir du programme précédent) qui affiche tous les caractères ASCII en partant de la valeur 32 jusqu'à la valeur 127. Pourquoi ne démarrons nous pas à partir de la valeur 0 ? (Cf. cours et table ASCII).

Exercice 3 : calcul d'une factorielle

1) Ecrire un programme **factorielle** qui calcule **à l'aide d'une boucle (pas récursivement puisque pas encore abordé en cours)** la factorielle d'une variable entière x (exemple 5 !=5*4*3*2*1) puis l'affiche (par convention 0 !=1).
2) Essayez votre programme avec 0,1, 3, 4 pour vérifier votre programme
3) Ajouter une boucle externe qui englobe ce que vous venez de valider afin que votre programme calcul et affiche la factorielle des nombre 0 à 15,
4) Comparez les résultats affichés avec une calculatrice (celle du PC par exemple), vous devriez être surpris. Comment expliquez-vous cela ?
5) Donnez des modifications qui permettent d'améliorer le problème précédent. Jusqu'à combien pouvez-vous aller maintenant ?

Exercice 4: utilisation du debugger GDB (**optionnel** pour ceux qui ont le temps et qui souhaitent découvrir des outils utiles et avancés)

1) Saisir, compiler et exécuter le programme suivant que vous appellerez ex2.c :

```
1  #include<stdio.h>
2  int main() {
3          int compteur=0, somme=0;
4          do {
5                  compteur++;
6                  somme=somme+compteur;
7           } while (somme < 10) ;
8            printf("compteur=%d, somme=%d \n", compteur, somme);
9          return 0;
10 }
```

2) Nous allons maintenant exécuter ce programme en mode pas avec pas à l'aide d'un debugger en ligne de commande appelé GDB :
   a. il faut commencer par recompiler le programme avec l'option supplémentaire "-g" ce qui ajoute des informations au programme produit afin qu'il soit utilisable par gdb : "gcc ex2.c –o ex2 **–g**". Le programme peut toujours être exécuté comme précédemment (./ex2), mais il est un peu plus gros (il contient des informations supplémentaires) et aussi plus lent (des optimisations sont désactivées)
   b. lancer le logiciel gdb en lui passant en argument le programme à débugger : "**gdb ex2**", vous obtenez alors le prompt du debugger : (gdb)

c. <u>vous pouvez maintenant exécuter votre programme en mode pas à pas</u> :
    i. **taper "start"** : gdb va exécuter le programme jusqu'à la première ligne qui suit la déclaration de la fonction main, puis suspend l'exécution : il vous affiche le code source de cette ligne
    ii. vous pouvez ensuite observer le déroulement du programme, ligne après ligne en tapant **s (pour step), n'oubliez pas de valider avec la touche entrée**. Vous pouvez répéter cette commande autant de fois que vous voulez pour étudier le déroulement du programme
        - le plus intéressant est qu'à tout moment vous pouvez regarder le contenu des variables du programme avec la commande "**print NomDeLaVariable**" : "**print somme"** ou "**print compteur"**
        - vous pouvez exécuter plusieurs itérations d'un coup en ajoutant un argument à s, ainsi : **s 10** signifie faire 10 lignes puis s'arrêter
d. vous pouvez aussi ajouter des "points d'arrêt" (breakpoint en anglais) sur les lignes que vous souhaitez : pour arrêter le programme avant le printf qui est ligne 8, on écrit **"b 8"** (b pour breakpoint, 8 pour le numéro de la ligne), et ensuite on tape **run** : gdb exécute alors le programme jusqu'à la ligne 8 puis rend la main : on peut le faire fonctionner à nouveau en mode pas à pas.