TD1

Langage C (LC4)

Semaine du 28 janvier 2013

1 Structures de contrôle

1.1 Boucles

Question 1. Écrivez une fonction `int somme(int tab[], int taille)` qui prend en argument un tableau d'entiers `tab` et sa taille (`taille`) et qui renvoie la somme des nombres qu'il contient. Résolvez ce problème :
- en utilisant **`while`**,
- en utilisant **`do-while`**,
- en utilisant **`for`**.

Question 2. Écrivez une fonction **`void`** `initialise(`**`int`** `tab[],` **`int`** `taille,` **`int`** `val)` qui met la valeur `val` dans les `taille` cases du tableau `tab`.

Question 3. Écrivez une fonction **`int`** `main()` dans laquelle vous déclarez un tableau de 100 entiers et où au moyen des fonctions précédentes, vous initialisez à 1 les 100 entiers puis affichez leur somme (par la fonction `printf()`).

1.2 Boucles et tests

Question 4. Écrivez une fonction **`void`** `seuil(`**`int`** `tab[],` **`int`** `taille,` **`int`** `min)` qui met à `min` les cases du tableau qui sont inférieures.

1.3 Boucles et tests pour les plus courageux

Question 5. Écrivez une fonction **`int`** `inverse(`**`int`** `nb)` qui vérifie que `nb` est strictement compris entre 0 et 10000 et, le cas échéant, affiche `nb` à rebours. Par exemple, si `nb`=1234, la fonction affichera 4321 (pensez à utiliser la fonction `modulo`).

Question 6. Écrivez une fonction **`int`** `factorielle_boucle(`**`int`** `n)` qui calcule `n!` en utilisant une boucle. Écrivez ensuite une fonction **`int`** `factorielle_recursive(`**`int`** `n)` qui fait la même chose récursivement.

Question 7. Écrivez une fonction `tasse(`**`int`** `n,` **`int`** `tab[])` qui efface toutes les occurrences du chiffre 0 dans le tableau `tab` et tasse les éléments restants (on remplira les cases vides en fin de tableau par des zéros).

1.4 Tableaux

Question 8. Écrivez une fonction `void inverse_tab(int n,int tab[])` qui range les éléments du tableau `tab` dans l'ordre inverse sans utiliser de tableau supplémentaire.

Question 9. Écrivez une fonction `void fusion(int n, int tabA[], int m, int tabB[], int fus[])` qui prend en argument les tableaux `tabA` (de taille `n`) et `tabB` (de taille `m`) triés par ordre croissant et remplit le tableau `fus` (de taille `n + m`) avec les éléments de `tabA` et `tabB` triés par ordre croissant.

2 gcc et l'option -D

2.1 Le code le plus court (ou presque)

Soit le code `court.c` :

```
#include <stdio.h>

P
```

Question 10. Quel est le code `courtcar.i` généré par la commande suivante ?

```
cpp -DP="int main() { X }" -DX="putchar(65);R" -DR="return 0;"
court.c courtcar.i
```

Après compilation que fait ce code à l'execution ?

Question 11. Quel est le code `courtstring.i` généré par la commande suivante ?

```
cpp -DP="int main() { X }" -DX="puts("pouet");R" -DR="return 0;"
court.c courtstring.i
```

Peut-on compiler le code contenu dans `courtstring.i`. Pourquoi ? Comment peut-on améliorer la commande ci-dessus pour générer un code compilable ?

Question 12. Soit `bouclechar.c` le programme suivant :

```
#include <stdio.h>

int main() {
  int c;
  c = getchar();
  while (c != EOF) {
    putchar(c);
    c = getchar();
  }
  return 1;
}
```

Écrivez une commande permettant de générer `bouclechar.i` le précompilé de `bouclechar.c` à partir du programme `court.c`.