TP5 – Arguments d'un programme - Pointeurs

Langage C (LC4)

semaine du 1^{er} mars

1 Arguments d'un programme

Quand on exécute un programme en ligne de commande, on peut faire suivre le nom du programme de plusieurs arguments, par exemple `./programme truc bidule 18`. La fonction `int main(int argc, char *argv[])` peut récupérer ces arguments, de la manière suivante :
– `argc` contient le nombre d'arguments ;
– `argv` est un tableau de chaînes de caractères contenant ces arguments ;
– `argv[argc]` est le pointeur nul `NULL`.

Attention : `./programme` est considéré comme un argument, il apparaît donc dans `argv[0]`. De plus tous les arguments sont récupérés sous la forme de chaînes de caractères (y compris le `18` ci-dessus).

Question 1. Dans un fichier `affiche_args.c`, écrivez une fonction `int main(int argc, char *argv[])` qui affiche dans la sortie standard le message suivant : « Vous avez rentré ... arguments. Voici les arguments que vous avez rentrés : ... » . Vous séparerez les arguments par des tabulations, et vous mettrez un retour à la ligne à la fin. Faites-le tourner sur plusieurs exemples.

1.1 Conversions

Vu que tous les arguments sont des chaînes de caractères, il va falloir les convertir selon ce qu'on veut obtenir. Pour cela on peut utiliser les fonctions `atoi` et `atof` dans la librairie standard qui convertissent une chaine de caractère en entier ou en flottants.

Question 2. Dans un fichier `somme_reels.c`, écrivez une fonction `main` qui affiche la somme des nombres réels rentrés par l'utilisateur sous la forme « La somme des nombres rentrés est ... », et en arrondissant le résultat au centième. Que se passe-t-il si l'utilisateur ne rentre aucun argument ? Modifiez votre programme pour qu'il affiche 0 si l'utilisateur ne rentre aucun argument. Faites tourner votre programme sur plusieurs exemples, en essayant notamment de rentrer autre chose que des réels pour voir ce qui se passe.

Question 3. Dans un fichier `somme_entiers.c`, écrivez une fonction `main` qui calcule la somme des nombres entiers rentrés par l'utilisateur. Que se passe-t-il si l'utilisateur rentre autre chose que des entiers ? Modifiez votre programme pour que, si l'utilisateur rentre autre chose que des entiers, il affiche un message d'erreur indiquant le numéro du premier argument erroné. N'oubliez pas qu'il y a des entiers négatifs.

2 Polynômes

On représente toujours un polynôme par un tableau de **double**. Mais on regroupe le pointeur vers les coefficients et le degré du polynôme dans une structure (*Rappel* : un polynôme de degré d a $d+1$ coefficients) :

```
typedef struct { int degre; double *coefficients; } polynome;
```

Question 4. Écrire une fonction `double valeur_polynome(int n, polynome *P)` qui évalue la valeur du polynôme P en n.

Question 5. Écrire une fonction `polynome *somme_polynome (polynome *P, polynome *Q)` qui calcule la somme des polynômes P et Q.

Question 6. Écrire une fonction `polynome *produit_polynome (polynome* P, polynome* Q)` qui calcule le produit des polynômes P et Q.

3 Matrices

Question 7. Écrire une fonction `int **alloue_matrice (int lignes, int colonnes, int val)` qui créé une matrice de taille lignes $\times$ colonnes dont toutes les cases ont été initialisées à la valeur val. Écrire ensuite une fonction `void affiche_matrice (int lignes, int colonnes, int **M)` qui permet d'afficher la matrice M de taille lignes $\times$ colonnes.

Question 8. Écrire une fonction `void libere_matrice(int lignes, int colonnes, int **M)` qui libère la matrice M de taille lignes $\times$ colonnes. A-t-on vraiment besoin des deux premiers arguments ?

Question 9. Écrire une fonction `int **genere_identite(int n)` qui génère la matrice identité de taille n.

Question 10. Écrire une fonction `int **produit_matriciel(int l1, int c1, int **M1, int l2, int c2, int **M2)` qui calcule le produit matriciel de la matrice M1 (de taille l1 $\times$ c1) par la matrice M2 (de taille l2 $\times$ c2) si leurs tailles sont compatibles, et renvoit NULL sinon.