TP3 : chaînes de caractères

12 octobre

On rappelle qu'une chaîne de caractères est un tableau de `char`. Elles ont la particularité de toujours se terminer par le caractère nul que l'on appelle sentinelle. On utilisera ici la librairie `string.h` qui contient notamment les fonctions suivantes au programme : `strlen`, `strcpy` et `strcat`.

Vous trouverez ici un morceau de code qui vous permettra de comprendre le rôle de ces fonctions.

```
int main(){
   char nom[20];
   strcpy(nom,"bienvenu");
   printf("%s\n",nom);
   printf("%ld\n", strlen(nom));
   char prenom[20];
   strcpy(prenom,"jean");
   printf("%s\n",prenom);
   strcat(prenom,nom);
   printf("%s\n",prenom);
   printf("%s\n",nom);
   return 0;
}
```

1 Echauffement

▷ **Question 1.** En utilisant la sentinelle, écrire une fonction `longueur` qui prend une chaîne de caractères en entrée et renvoie sa taille. ◁

▷ **Question 2.** Ecrire une fonction `affiche_chaine` qui prend en entrée une chaîne de caractères et l'affiche. ◁

2 Le code César

▷ **Question 3.** Ecrire une fonction `encode` qui prend en entrée une chaîne de caractères `message` et un entier `i` et qui renvoie une chaîne de caractères qui correspond au message encodé à l'aide du code César c'est-à-dire où on a appliqué un décalage de i lettres à chacune des lettres du message. Les chaînes de caractères seront encodées avec des pointeurs afin de pouvoir être une valeur de retour.

Par exemple `encode("bonjour", 3)` renvoie `"erqmrxu"`.

La signature de la fonction sera : `char* encode(char* message, int i)`. ◁

3 Recherche de motifs

▷ **Question 4.** Ecrire une fonction `est_fact_gauche` qui prend en entrée deux chaînes de caractères $s1$ et $s2$ et détermine si $s1$ est un facteur gauche de $s2$. Par exemple `"mp"` est un facteur gauche de `"mp2i"`. La fonction renverra un booléen et pour cela on utilisera la librairie `stdbool`. ◁

▷ **Question 5.** Ecrire une fonction `est_fact` qui prend en entrée deux chaînes de caractères $s1$ et $s2$ et détermine si $s1$ est un facteur de $s2$. Par exemple `"bell"` est un facteur de `"libellule"`. La fonction renverra un booléen. On pourra essayer d'utiliser la commande `break` qui permet de sortir de la boucle dans laquelle on se trouve (d'une seule boucle en cas d'imbrication). ◁

▷ **Question 6.** On attend de vous que vous sachiez écrire des tests. Vous testerez (en utilisant la librairie `assert.h`) la fonction précédente en exécutant la commande suivante dans votre fonction `main` :

```
char* s2 = "bonjour";
char* s1 = "onj";
char* s11 = "journ";
char* s111 = "bonjoura";
char* s1111="bonjou";

assert (est_fact(s1,s2)==true);
assert (est_fact(s11,s2)==false);
assert (est_fact(s111,s2)==false);
assert (est_fact(s1111,s2)==true);
```

Créer des tests pour la fonction `est_fact_gauche`. ◁

4 Un langage

On définit le type `typedef char* mot` ainsi que le type `typedef mot* langage`.

▷ **Question 7.** Ecrire une fonction `affiche` qui prend en entrée un langage et sa taille et affiche tous les mots qui le composent. ◁

▷ **Question 8.** Ecrire une fonction `appartient` qui prend en entrée un mot et un langage et sa taille et vérifie si le mot appartient ou non au langage. On commencera par écrire une fonction qui prend en entrée deux chaînes et décide si elles sont ou non égales. ◁

▷ **Question 9.** Ecrire une fonction `ajoute` qui prend en entrée un mot et un langage et sa taille et renvoie un nouveau langage construit à partir de celui passé en entrée où on a ajouté le nouveau mot s'il n'y était pas déjà présent. ◁

5 Automates cellulaires

Un automate cellulaire est un tableau dans lequel les cases peuvent avoir différents états. On considère une situation initiale de l'automate et à chaque étape chacune des cases va changer d'état en fonction de la configuration générale de l'automate cellulaire.

Nous allons ici considérer un cas simple de tableau unidimensionnel dont chaque case est soit dans l'état 0 soit dans l'état 1 (pour avoir ensuite une représentation graphique, nous considérerons que les états sont des caractères : `' '` pour 0 et `'*'` pour 1).

Chaque case de l'automate cellulaire est appelée cellule. A chaque étape, la cellule va changer d'état en fonction de son propre état ainsi que de celui des cases qui lui sont adjacentes (celle à sa gauche et celle à sa droite) suivant les règles ci-dessous (on trouve à gauche de la flêche la suite correspondant aux états respectifs de la cellule gauche, la cellule du milieu et la cellule droite et à droite de la flêche le nouvel état de la cellule droite) :

1. `000->0`
2. `001->1`
3. `010->1`
4. `011->1`
5. `100->1`
6. `110->0`
7. `111->0`
8. `101->0`

On considérera que toutes les cases ont bien un voisin gauche et un voisin droit en considérant que les cases qui sont en dehors du tableau sont à 0 (contiennent `' '`).

▷ **Question 10.** Ecrire une fonction `regle` qui prend en entrée un triplet d'entiers `g`, `c`, `d` et qui renvoie l'état de la case `c` après application de la règle décrite ci-dessus.

La signature de la fonction sera : `int regle(int g, int c, int d).` ◁

▷ **Question 11.** On remarque qu'il y a huit configurations différentes. Si on considère les trois cases comme l'écriture en base 2 d'un entier alors on peut naturellement numéroter les configurations. Par exemples : 000 correspond à 0 et 100 à 4. Ecrire une nouvelle fonction `regle_binaire` qui prend en entrée un numéro de configuration correspondant à cet encodage binaire et renvoie un entier qui correspond à la nouvelle valeur de la case du milieu.

La signature de la fonction sera : `int regle_binaire(int config).` ◁

▷ **Question 12.** Ecrire une fonction `suivant` qui prend en entrée une chaîne de caractères `in` de cellulles contenant des `' '` et des `'*'` ainsi qu'une chaîne de caractères `out` de même longueur dans laquelle on va stocker le résultat de l'application de la règle décrite précédemment sur chacune des cellules de `in`.

La signature de la fonction sera : `void suivant(char* in, char* out).` ◁

▷ **Question 13.** Tester votre fonction `suivant` à l'aide du test suivant :

```
char in[11]="**  ***  *";
char out[11]="          ";
suivant(in,out);
assert(strcmp(out,"* ***  ***")==0);
```

◁

▷ **Question 14.** Ecrire une fonction `evolution` qui prend en entrée un entier `iterations` et une chaîne de caractères `init`, applique `iterations` fois la règle en démarrant de la configuration `init` et renvoie la chaîne obtenue à l'issue des `iterations` étapes.

La signature de la fonction sera : `char* evolution(int iterations, char* init).` ◁

On pourra terminer en testant le code suivant :

```
char init[81];
for (int i=0;i<80;i++){
   init[i]=' ';
}
init[80]='\0';
char* sortie=(char*)malloc(81*(sizeof(char)));
sortie[80]='\0';
strcpy(sortie,init);
init[40]='*';
for (int j=0;j<30;j++){
   sortie=evolution(j,init);
   affiche_chaine(sortie);
   printf("\n");
}
```