RABILLER Donatien

**Projet LO22**

**Conversion chaine littérale – nombre**

**Introduction**

L’objectif de ce projet est de programmer un convertisseur permettant de traduire des nombres en lettres et inversement. Le programme est réalisé en langage C et compilé à l’aide de gcc. L’ensemble du processus de compilation est quant à lui organisé via un makefile. Le programme est séparé en 3 fichiers : converter.c, converter.h et main.c.

**I/ Description des fonctions**

Le programme présente deux fonctions principales de conversion : letter2int(char\* chaineNombre) et int2letter(int value), ainsi que plusieurs fonctions utilitaires.

1. Fonction de conversion

* Entier vers lettres :

La première fonction de conversion permet de traduire un nombre entier en toutes lettres. Elle a pour prototype : *char*\* int2letter(unsigned *int* value) ; Cette fonction prend en paramètre un entier positif et retourne un pointeur sur un tableau de caractère. Elle gère les nombres de 0 à 4 294 967 295 (quatre milliards deux cent quatre-vingt-quatorze millions neuf cent soixante-sept mille deux cent quatre-vingt-quinze).

* Lettres vers entier :

La seconde fonction de conversion permet de traduire un nombre littéral en nombre. Elle a pour prototype : *unsigned int* letter2int(*char\** chaineNombre) ; Cette fonction prend en paramètre une chaine de caractère représentant un nombre et retourne un nombre positif. Elle gère les nombres de 0 à 4 294 967 295. Cette fonction s’appuie sur une sous-fonction: *int* letter2intPacket(*char\** chaineNombre) ; qui prend en paramètre une chaine de caractère représentant sous forme littérale un nombre de 0 à 999 et retourne sa représentation sous forme d’un entier.

1. Fonction utilitaire

Pour réaliser le projet, je me suis appuyé sur plusieurs fonctions utilitaires afin de gérer les chaînes de caractères.

*void* insertAtStart(const *char\** source, *char\** destination) ; Cette fonction insère la chaîne de caractères **source** au début de la chaîne **destination**. Attention : **destination** doit avoir suffisamment d’espace alloué pour pouvoir stocker **source** en plus de son propre contenu.

*int* getFirstOccurenceInString(const *char\** str1, const *char\** str2); Cette fonction trouve la première occurrence de la chaîne str2 dans la chaîne str1 et retourne l’index du premier caractère de str2 dans str1. La valeur de retour prend alors 0 si rien n’a été trouvé, ou un nombre supérieur à 0 si str2 est trouvé.

*char\** cutStringFromStart(const *char\** str, *int* n) ; Cette fonction alloue et retourne un pointeur sur un tableau de caractère contenant les n premier caractères contenus dans str en ajoutant le caractère EOF a la fin de la chaîne retournée.

*void* trim(*char\** s) ; Cette fonction réalise un trimmage sur la chaîne s en enlevant les caractères espace présents au début et à la fin de la chaîne. Il faut faire attention avec cette fonction car le pointeur s est modifié. Il faut donc garder une sauvegarde de son état avant l’application de trim pour pouvoir libérer l’espace mémoire correctement.

*Int* contains(*char\** arr[], *int* arraySize, *char\** toCompare) ; Cette fonction cherche la chaîne de caractère **toCompare** dans le tableau de chaîne **arr**. Elle retourne -1 si toCompare n’a pas été trouvé et l’index du premier caractère auquel se trouve toCompare sinon.

*Unsigned int* getPowerWithI(*int* i) ; Cette fonction retourne un entier positif qui correspond à 10^(i\*3) pour i=1, 2 ou 3.

**II/ Variables globales**

En plus des fonctions de conversion, le fichier converter.c déclare aussi plusieurs variables globales. C’est variables sont des tableaux de chaîne de caractère.

*char\** units[] = { … } Cette variable définit la correspondance entre une unité en chiffre et en littéral, par exemple units[2] = ‘’deux’’ et units[9] = ‘’neuf’’.

*char\** dizaine[] = { … } Cette variable définit la correspondance entre une dizaine en chiffres et sa représentation littérale à un index près, par exemple dizaine[0]= ‘’dix’’, dizaine[4] = ‘’cinquante’’, par contre la différentiation quatre-vingt, quatre-vingt-dix d’une part, et soixante, soixante-dix n’est pas faite.

*char\** specials[] = { … } Cette variable définit la correspondance entre un nombre spécial en chiffres et sa représentation littérale a 11 près, par exemple specials[0] = ‘’onze’’, specials[3] = ‘’quatorze’’.

*char\** puissancedix[]= { … } Cette variable définit la correspondance entre les puissance de dix en littéral et en nombre : puissancedix[0] = ‘’cent’’, puissancedix[1] = ‘’mille’’. On peut retrouver la puissance par rapport à l’index en multipliant l’indice par 3. 1000 = 10^3 = 3\*1.