Projets de Langage C

Romain Dubessy

Consignes

Cette année les projets se font à distance. Ils correspondent en principe à 8 séances de 4 heures (soit 32 heures de travail en binôme). Chaque groupe sera suivi par un enseignant avec il sera possible d'échanger « en direct » **après une prise de rendez-vous** (par mail, ...) ou de manière asynchrone par messages (mail, teams, ENT, ...). Ce document détaille les consignes générales pour chaque projet, une suggestion de planning ainsi que les modalités d'évaluation.

Liste des projets :

Quelques remarques générales

Évaluation des projets Les projets seront évalués selon trois critères :

- quantité et qualité du travail fourni : à la fin de chaque journée de travail il faut déposer sur l'ENT avant 20h un code qui compile et s'exécute à chaque séance. Bien entendu après les premières séances le programme ne fera pas « grand chose », mais il est très important d'avoir une progression (une amélioration à chaque séance). En terme d'organisation cela veut dire qu'il ne faut pas déposer un code avec une fonction incomplète (qui ne compile pas).
- clarté des explications : chaque code déposé doit être commenté : Noms, date et version du code, quelques phrases pour décrire l'état d'avancement du projet, une phrase de commentaire au début de chaque fonction implémentée.
- démonstration : le rapport final prendra la forme d'une vidéo commentée en voix *off* expliquant le projet (5 minutes) et faisant une démonstration du code (5 minutes), à déposer sur l'ENT.

Organisation des projets Le plan des séances sera approximativement :

- 1ère séance : analyser le problème, répertorier les diverses structures de données nécessaires et les divers outils (fonctions) pour les manipuler. Faire **valider les conclusions** par l'enseignant encadrant.
- séances 2 et 3 : fonctions d'affichage (`print`), de gestion des fichiers (`save`, `load`) et de gestion de liste (`append`, `remove`, ...). Ces fonctions sont indispensables pour tous les projets et doivent être testées rigoureusement.
- séances 4 à 6 : implémentation de la fonction principale (algorithme) et des fonctions secondaires nécessaires au bon fonctionnement du projet,
- séance 7 et 8 : implémentation des fonctions d'interface permettant l'interaction avec l'utilisateur.

Gestions des fichiers Lorsqu'il y aura besoin de sauvegarder des données dans un fichier texte (ASCII) on utilisera le format `csv`. Un fichier `csv` permettra de sauvegarder les informations contenues dans un type de **structure** en respectant les contraintes :

- chaque ligne du fichier correspond à un objet du type **structure**,
- les champs de la structure sont séparés par des ';'.

1 Carnet d'adresses

Position du problème Le but de ce projet est de réaliser un logiciel de gestion de carnet d'adresse permettant de gérer des contacts.

Structures de données Afin de pouvoir ajouter et supprimer des fiches de contact du carnet d'adresse on utilisera un stockage sous forme de liste chaînée, chaque élément de la liste contenant les données d'un contact. Pour faciliter la gestion des données d'un contact, on utilisera plusieurs structures permettant de stocker les informations suivantes :

- nom, prénom,
- date de naissance,
- adresse :
  - numéro, type et nom de rue,
  - code postal,
  - ville
- numéro de téléphone,
- e-mail.

L'ensemble des contacts pourra être stocké dans un format ASCII dans un fichier `contacts.csv` (un contact par ligne du fichier), qui devra pouvoir être relu plus tard. Les contacts seront stockés dans la liste par ordre alphabétique.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes, au moyen d'une interface simple :

- appel du programme avec la syntaxe suivante :
  `addressManager FILE`, où `FILE` est le fichier de base de données,
- ajouter un nouveau contact,
- rechercher un contact par nom et :
  - afficher sa fiche,
  - supprimer sa fiche,
  - mettre à jour ses données,
- sauvegarder la liste de contact dans un fichier,
- charger une liste de contact sauvegardée.

Enfin si le temps le permet on cherchera à générer à partir de la liste de contacts un ensemble de fichier `html` statiques comportant un index avec la liste des noms par ordre alphabétique et une page associée à chaque fiche.

2 Gestion de stock

Position du problème Le but de ce projet est de réaliser un logiciel de gestion de stock permettant d'organiser des produits en différentes catégories.

Structures de données On propose d'organiser les produits sous forme de liste chaînée de catégories, chacune contenant une liste chaînée de produits. Chaque fiche de produit devra contenir les informations suivantes :

- nom du produit,
- prix du produit,
- quantité disponible.

Chaque catégorie devra contenir :

- le nom de la catégorie,
- une liste de produit.

L'ensemble des produits pourra être stocké dans un fichier ASCII `produits.csv` sous la forme suivante :

```
[categorie 1]
nom_produit;prix_produit;quantite_disponible
nom_produit;prix_produit;quantite_disponible
[categorie 2]
nom_produit;prix_produit;quantite_disponible
...
```

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes, au moyen d'une interface simple :

- appel du programme avec la syntaxe suivante :
  `stock FILE`, où `FILE` est le fichier de base de donnée,
- ajout et suppression d'une catégorie,
- ajout et suppression d'un produit (dans une catégorie),
- mise à jour d'un produit (prix ou quantité disponible),
- affichage des produits d'une catégorie :
  - triés par ordre alphabétique,
  - triés par prix croissant ou décroissant,
  - triés par quantité disponible,
- sauvegarder la liste des catégories et produits dans un fichier `produits.csv`,
- charger une liste de catégories et produits sauvegardée.

Pour effectuer les différents tris on suggère de stocker les catégories et produits par ordre alphabétique puis, au besoin, de créer une copie triée avec un tri par insertion de la liste de produits, par exemple par prix décroissants.

3 Vérification de code

Position du problème Le but de ce projet est de réaliser un logiciel capable de lire un fichier source écrit en langage C, de vérifier que la syntaxe élémentaire du langage C est respectée et de produire un fichier source correctement mise en page.

Structures de données On propose d'adopter la stratégie suivante :

- on lit l'intégralité du fichier source et on le transfère dans une chaîne de caractères,
- on convertit cette chaîne de caractère en une liste d'éléments, qui peuvent être :
  - un mot clé du langage (if, else, for, switch, break, ...),
  - un symbole (;, {, (, [, +, *, <, &, ++, ...),
  - un identificateur ou une constante,
  - une chaîne de caractère.
- on lit ensuite la liste d'éléments, on vérifie les règles de syntaxe et on fait la mise en page.

Chaque élément contiendra les informations suivantes :

- un code permettant de connaître le type (mot clé, symbole, ...),
- une chaîne de caractère contenant l'élément.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- appel du programme avec la syntaxe suivante :
  `codeAnalyzer FILE`, où `FILE` est le fichier source à modifier,
- pour le développement : affichage des éléments de la liste avec leur type,
- production d'un fichier de sortie contenant le programme mis en page.

Pour la mise en page on respectera les règles suivantes :

- on remplacera les tabulations par des espaces,
- retour à la ligne après chaque instruction,
- retour à la ligne avant un accolade ouvrante / fermante et modification de l'indentation,
- si possible lignes d'au maximum 80 caractères.

4 Itinéraires de métro

Position du problème Le but de ce projet est de réaliser un logiciel capable de trouver un itinéraire permettant d'aller d'une station du réseau de lignes de métro de la RATP à une autre en un temps minimum.

Structures de données On suggère de procéder en deux étapes. Il faut d'abord arriver à lire le fichier `metro.csv` (fourni) qui contient la liste des stations de métro, organisées par lignes, ainsi que les horaires de premier et dernier métro permettant d'estimer le temps de trajet entre deux stations. Ces données peuvent être organisées sous la forme d'un tableau de lignes de métro, chaque ligne contenant une liste doublement chaînées de stations. Chaque station contient :

- le nom de la station,
- le temps qu'il faut pour aller à la suivante,
- le temps qu'il faut pour aller à la précédente.

Il faut ensuite adapter un algorithme de recherche d'itinéraire à ce problème, ce qui peut se faire directement en utilisant des listes.

Pour tester le programme il est conseillé de procéder par étapes en commençant par traiter le cas d'une ligne, puis d'un trajet sur deux lignes avec un changement avant de tester le cas général.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- appel du programme avec la syntaxe suivante :
  `pathFinder FILE`, où `FILE` est le fichier contenant la liste des stations,
- charger le fichier contenant les stations,
- demander une station de départ et une station d'arrivée,
- calculer l'itinéraire optimal et l'afficher :
  - en indiquant les correspondances,
  - et les temps de trajet pour chaque tronçon.

5 Résolution de Sudoku

Position du problème Le but de ce projet est de réaliser un logiciel capable de résoudre une grille de Sudoku.

Structures de données On propose d'adopter une méthode de solution par « essai–erreur » où on essaye successivement toutes les combinaisons possibles jusqu'à trouver la bonne. On utilisera des tableaux à deux dimensions pour représenter les grilles de sudoku et on stockera la solution dans une liste chaînée de type « pile » (LIFO : *last in first out*).

Les grilles de sudoku seront lues et stockées dans un fichier ASCII sous formes de tableaux de 9 lignes et 9 colonnes, séparées par une ligne vide.

La principale difficulté de ce projet réside dans la réalisation de l'algorithme de solution, qui doit éviter autant que possible les manipulations trop compliquées. On conseille de procéder par étapes et d'isoler dans des fonctions bien identifiées les tâches simples, comme par exemple tester si un chiffre est autorisé...

Dans un deuxième temps on se posera la question d'implémenter des raisonnements logiques permettant de déduire la valeur des cases manquantes « évidentes » (déductions directes ou indirectes...). On pourra alors estimer la difficulté des grilles suivant la complexité des raisonnements mis en jeux / ou le nombre d'essais requis.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- appel du programme avec la syntaxe suivante :
  `sudokuSolver FILE`, où `FILE` contient une ou plusieurs grilles de sudoku,
- lire des grilles de sudoku dans un fichier passé en argument,
- afficher une grille de sudoku,
- résoudre une grille de sudoku par la méthode « essai–erreur » (indiquer si la grille n'a pas de solution),
- améliorer la résolution en utilisant des déductions logiques,
- estimer la difficulté des grilles,
- sauvegarder la solution,
- proposer une interface simple de jeux.

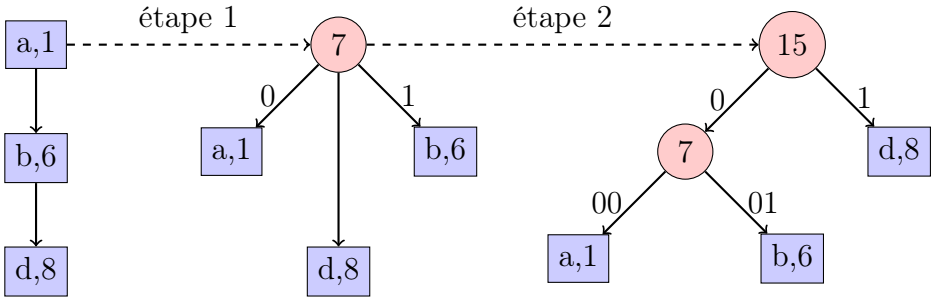
6 Compression de fichiers

Position du problème Le but de ce projet est de réaliser un logiciel capable de compresser sans pertes de données un fichier, au moyen de l'algorithme de Huffman.

Structures de données On suggère de suivre les étapes suivantes :

- ouvrir le fichier, le lire dans une chaîne de caractères et compter le nombre d'occurrences de chaque caractère,
- créer une liste des caractères, triés par nombre croissant d'occurrences,
- à partir de cette liste, créer un arbre contenant le code de Huffman,
- associer à chaque caractère son code,
- traduire le fichier de départ en fichier « codé ».

Le décodage s'obtient en effectuant les étapes en ordre inverse. Pour que cela soit possible il faut transmettre dans le fichier « codé » un moyen de reconstruire le dictionnaire (par exemple le nombre d'occurrences de chaque caractère. Les deux principales difficultés sont : la construction du code de Huffman (voir l'exemple ci-dessous) et l'écriture du code binaire *bit par bit*.



Remarque : à chaque étape on prend les deux premiers éléments de la liste triée, on les combine en un nouvel élément (on additionne les poids), qu'on réinsère dans la liste (ordre croissant). On s'arrête quand la liste ne contient plus qu'un élément. L'arbre permet de trouver le code de chaque caractère.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- appel du programme avec la syntaxe suivante :
  - `compress ACTION INFILE OUTFILE`, où `ACTION` est soit `encode`, soit `decode`, pour compresser ou décompresser un fichier,
  - `compress stat FILE`, pour afficher les statistiques d'un fichier, le taux de compression, ainsi que le « dictionnaire » utilisé.
- compression d'un fichier texte, et, éventuellement, d'un fichier binaire (image, vidéo, ...).

7 Correcteur d'orthographe

Position du problème Le but de ce projet est de réaliser un logiciel capable d'analyser un texte, de détecter les fautes d'orthographe et de proposer des corrections.

Structures de données Le programme utilisera un fichier « dictionnaire » fourni, qui contient tous les mots de la langue française (environ 630000 !), à raison de un mot par ligne. Afin de simplifier le problème on ne s'intéresse pas aux accents.

La première étape consiste à charger en mémoire le dictionnaire comme une liste simplement chaînée de mots, qui n'a pas besoin d'être triée par ordre alphabétique. Ensuite on lit le texte à corriger, et pour chaque mot, on cherche si il existe dans le dictionnaire. Si non, le mot est mal orthographié. *Remarque* : attention aux majuscules ! Le dictionnaire contient des mots écrits uniquement en minuscules.

Pour proposer une correction, on suggère de trouver les mots proches sur le critère suivant : on cherche les mots qui contiennent la plus longue séquence de lettres commune avec le mot à corriger, en tolérant au maximum trois différences. On détectera donc les fautes d'orthographe de type « faute de frappe ».

Étant donné la taille du dictionnaire (plusieurs méga-octets de données) on attachera une attention particulière à la gestion de la mémoire et à l'efficacité de la gestion des listes !

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

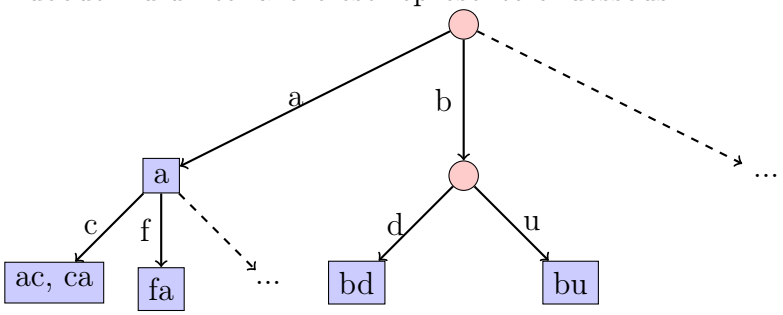
- appel du programme avec la syntaxe suivante :
  `spellcheck DICT FILE`, où `DICT` est le fichier dictionnaire et `FILE` le fichier texte à analyser,
- afficher les mots inconnus dans le texte,
- proposer une ou plusieurs corrections pour chaque mots inconnus.

8 Aide pour jeux de lettres

Position du problème Ce projet consiste à réaliser un logiciel donnant la liste des mots que l'on peut former à partir d'un ensemble de lettres.

Structures de données Le programme utilisera un fichier « dictionnaire » fourni, qui contient tous les mots de la langue française (environ 630000 !), à raison de un mot par ligne. Afin de simplifier le problème on ne s'intéresse pas aux accents.

Comme le dictionnaire est très grand il faut concevoir un moyen efficace de chercher un mot à partir des lettres qui le composent. Pour cela on propose de ranger une les mots dans une structure de type « arbre », en utilisant comme clé les lettres du mot rangées par ordre alphabétique. Par exemple la clé associée au mot « langage » est « aaeggln ». Chaque noeud de l'arbre représentera ainsi une lettre du mot et aura donc 26 descendants potentiels. Le « début » d'un tel arbre est représenté ci-dessous :



Attention à chaque clé peut correspondre plusieurs mots ! Il faut donc garder dans les noeuds de l'arbre la liste des mots correspondant à la clé.

On conseille fortement de se restreindre dans un premier temps à construire un arbre contenant uniquement des mots courts.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités :

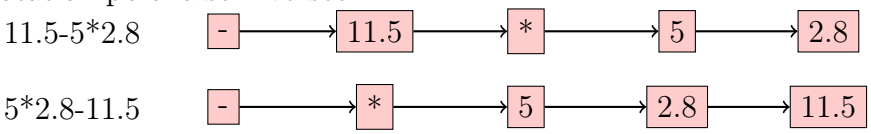
- appel du programme avec la syntaxe suivante :
  `help DICT`, où `DICT` est le nom du fichier dictionnaire,
- tirer aléatoirement quelques lettres,
- afficher la liste des mots qu'il est possible de construire avec ces lettres et les classer par score obtenu au « scrabble »,
- ajouter des contraintes : mots contenant certaines lettres à des endroits déterminés (placement sur une grille).

Remarque en changeant la clé par des chiffres de 0 à 9 on obtient le système de codage T9.

9 Analyseur d'expression

Position du problème Le but de ce projet est de réaliser un logiciel capable de faire un calcul mathématique à partir d'une expression écrite sous forme de chaîne de caractères.

Structures de données On conseille de convertir la chaîne de caractères en représentation sous forme de liste, comme dans l'exemple suivant, utilisant la notation polonaise inversée :



Le principe de la notation polonaise inversée est le suivant : lorsque l'on évalue l'expression, on prend les éléments dans l'ordre, avec la règle qu'un opérateur s'applique immédiatement aux deux éléments qui le suivent. Si deux opérateurs se suivent (comme dans le cas du deuxième exemple), il faut évaluer le second avant le premier : on aura donc intérêt à utiliser une fonction récursive pour évaluer l'expression.

La principale difficulté consistera à bien gérer les règles de priorité entre opérateurs, la présence de parenthèses, ainsi que les fonctions mathématiques standards.

On effectuera les calculs en précision « double ».

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- appel du programme avec la syntaxe suivante :
  `parser EXPRESSION`, où `EXPRESSION` est la chaîne de caractère à évaluer,
- capacités de calcul (par ordre de difficulté) :
  - gestion des opérateurs courants avec leur règles de priorité,
  - gestion des parenthèses,
  - gestion des opérateurs logiques,
  - fonctions mathématiques standard (celles de la librairie `math.h`),
  - gestion des nombres complexes,
- si une erreur d'entrée est détectée (parenthèses non refermées, ...) un message devra être affiché,
- éventuellement, manipulation d'expressions avec une ou plusieurs inconnues.

10 Intelligence artificielle

Position du problème Le but du projet est d'écrire un logiciel permettant de lire et reconnaître des chiffres dans des images, comme ci-dessous :

Pour cela on utilisera un réseau de neurones (une « IA ») que l'on entraînera à reconnaître les chiffres dans les images. Pour plus d'explications sur le principe voir ici.

Structures de données Les images sont stockées dans un format de 28×28 pixels en niveaux de gris (1 octet) par pixel. Le réseau de neurones est organisé en « couches » (*layers*), chaque couche contient un pointeur vers un buffer d'entrée, un pointeur vers un buffer de sortie et une liste de neurones. Chaque neurone contient une vecteur de coefficients $\{w_i\}$ (*weights*) et un biais b (*bias*). Un neurone agit sur le buffer d'entrée de son *layer* en calculant une sortie :

$$\text{output} = \frac{1}{1 + e^{-\text{input}\cdot\text{w} - b}}.$$

On commencera par étudier un réseau à deux *layers* : le premier contenant 300 neurones, le second contenant 10 neurones, chaque neurone estimant la probabilité que l'image corresponde à un chiffre donné.

Remarque : le buffer de sortie du *layer* N est le buffer d'entrée du *layer* $N + 1$.

Fonctionnalités souhaitées Le logiciel devra offrir les fonctionnalités suivantes :

- chargement et sauvegarde du réseaux de neurones dans un format csv :
  taille du layer 1
  bias ;w0 ;w1 ;.... ;wN (pour le neurone 1)
  bias ;w0 ;w1 ;.... ;wN (pour le neurone 2)
  ...
  taille du layer 2
  ...
- entraînement du réseau sur une base de donnée déjà classée,
- application du réseau à une base de donnée non classifiée.