Travaux Dirigés

Site: Espadon
Cours: ALGORITHMIQUE ET PROGRAMMATION NIVEAU 2
Livre: Travaux Dirigés
Imprimé par: Thomas Stegen
Date: mercredi 23 mars 2016, 16:22

Table des matières

1 Calcul d'un âge en jours

2 Traitement de deux listes d'entiers

3 Moyennes simples

4 Conversion en base

5 Le crible d'Eratosthène

6 Décomposition

7 Nombre parfait

8 Tri simple

9 Le jeu du Yatzee

1 Calcul d'un âge en jours

Le problème

Ce TD a pour but de calculer l'âge, exprimé en nombre de jours, d'une personne à partir de sa date de naissance.

Programmation

- La méthode la plus simple consiste à prendre une date de référence (1er Janvier 1901 par exemple) et à calculer le nombre de jours écoulés entre cette date référence et la date considérée. En effectuant ce traitement deux fois, une simple soustraction répond à la question,
- la date du jour courant sera entrée "en dur" dans le programme,
- la date de naissance sera passée en argument au programme sous la forme de 3 entiers (jour, mois, année) séparés par des caractères blancs,
- les plus avancés pourront chercher dans la librairie C une méthode permettant de récupérer la date du jour à partir de l'horloge de la machine.

2 Traitement de deux listes d'entiers

Le problème

Il s'agit d'écrire un certain nombre de procédures ou fonctions permettant de manipuler des listes d'entiers.

Programmation

Ecrire les procédures ou fonctions suivantes :

Traitement d'une liste d'entiers

- initialisation d'une liste de n entiers (n est entré au clavier par l'utilisateur),
- initialisation AU HASARD d'une liste de n entiers entre 0 et 100 (n est entré au clavier par l'utilisateur),
- afficher une liste d'entiers,
- inversion en place du sens de la liste,
- inversion en copie du sens de la liste,
- insertion en place d'un élément à une position donnée,
- insertion en copie d'un élément à une position donnée,
- suppression en place d'un élément à une position donnée,
- suppression en copie d'un élément à une position donnée,
- écrire une procédure ou fonction de recherche de la place du maximum dans une liste,
- écrire une procédure ou fonction de recherche **simple** d'un élément dans une liste. Il s'agit de donner une liste et un entier est de savoir si cet entier fait partie de la liste et si oui en quelle position. Votre procédure ou fonction devra donc rendre la position de l'entier dans la liste si l'entier est bien dedans et -1 s'il n'est pas dans la liste.

- suppression en copie de tous les entiers pairs de la liste,
- suppression en place de tous les entiers pairs de la liste,

Traitement de deux listes d'entiers

- calculer l'union de deux listes,
- calculer l'intersection de deux listes,
- calculer la différence (éléments propres à chaque liste) de deux listes.

Recherche dichotomique dans une liste d'entiers

Ecrire une procédure ou fonction de recherche **dichotomique** d'un élément x dans une liste. Le principe de recherche dichotomique ne peut s'appliquer que sur une liste triée (dans l'ordre croissant par exemple). Le mécanisme est le suivant

- considérer l'élément médian *med* de la liste (le milieu de la liste à un prés),
- comparer cet élément médian *med* à l'élément x à rechercher.
- si $x>med$ alors recommencer la recherche dans la partie de la liste située après *med*,
- si $x<med$ alors recommencer la recherche dans la partie de la liste située avant *med.*
- la recherche s'arrête lorsque l'élément médian est égal à x (on a alors trouvé x dans la liste) ou lorsque l'on est arrivé en début ou en fin de liste (on sait alors que x n'est pas dans la liste).

Choix des opérations

Vous écrirez une procédure menu permettant à l'utilisateur de choisir l'opération qu'il désire effectuer parmi toutes celles programmées.

3 Moyennes simples

Le problème

Ecrire un petit programme simple de calcul de moyennes de différents étudiants. Chaque étudiant se verra attribuer un nombre arbitraire de notes (comprises entre 0 et 20), le programme en fera la moyenne (non pondérée dans un premier temps), stockera cette moyenne et affichera un tableau récapitulatif de l'ensemble des moyennes.

Programmation

J'attire votre attention sur les points suivants :

- Le programme demandera le nombre d'étudiants à traiter puis, pour chacun d'entre eux, son nom et ses différentes notes,
- le nombre de notes étant variable, la liste des notes de chaque étudiant se terminera par une note négative (notion informatique de drapeau ou flag),
- les notes seront exploitées mais non stockées,
- deux tableaux à une dimension serviront au stockage d'une part des noms, d'autre part des moyennes,
- après saisie des notes du dernier étudiant, il y a impression d'un tableau récapitulatif,
- le cas où un étudiant n'a aucune note doit pouvoir être traité. Sa moyenne est alors symbolisée par un tiret ou une étoile.

Aller plus loin ...

- introduire des coefficients pour chacune des notes,
- ne plus utiliser deux tableaux (noms et moyennes) mais définir une structure pour représenter un étudiant et un tableau de structures pour représenter une classe

4 Conversion en base

Le problème

Nous nous proposons de réaliser un petit programme de conversion d'un nombre décimal donné vers sa représentation dans une base quelconque. Dans un premier temps nous nous limiterons à des bases comprises entre 2 et 10 mais il serait intéressant de pouvoir gérer jusqu'à la base 16 (hexadécimal).

Sur le plan théorique, il y a peu à dire, j'espère ...

Programmation

Si l'on se limite à la base 10, la représentation peut se faire à l'aide des 10 chiffres décimaux usuels. La représentation hexadécimale, requiert, quant à elle, l'utilisation des lettres A, B, C, D, E et F ce qui implique d'avoir recours à des tableaux de caractères et des tableaux d'entiers. Le passage des entiers aux caractères étant l'une des difficultés de ce TD.
Vous traiterez successivement les 2 problèmes suivants :

1. passage base 10 vers base n <10 puis vers base n <=16
2. passage base n<10 puis <=16 vers base 10

5 Le crible d'Eratosthène

Le problème

Cette très ancienne méthode permet la détermination de tous les nombres premiers inférieurs à une valeur donnée. La méthode manuelle consiste à dresser la liste de tous les entiers entre 1 et **n** et à y "rayer" tous les nombres multiples d'autres entiers.

Plus précisément, l'algorithme peut être décrit de la manière suivante :

1. rayer le 1,
2. rechercher, à partir du dernier nombre premier considéré, le premier nombre non rayé qui est forcement premier (demander la démonstration à votre professeur de maths préféré mais ce n'est pas bien compliqué). Ce nombre devient alors le dernier nombre premier et on raye tous ses multiples,
3. recommencer l'opération 2 jusqu'à ce que le nombre premier considéré soit supérieur à la **racine carrée de n.** On peut alors montrer que tous les nombres non premiers ont été rayés de la liste.

   Démonstration simple :
   - Si n n'est pas premier alors il existe p et q strictement compris entre 1 et n tels que n = p * q
   - l'un des deux est forcément plus petit que l'autre, disons p < q,
   - multiplions à droite et à gauche par p, nous obtenons p^2 < p*q donc p^2 < n donc **p < racine(n)**
   - donc si n n'est pas premier alors ces diviseurs sont < racine(n)

Programmation

Quelques indications :

- l'énoncé semble suggérer l'utilisation d'un tableau d'entiers mais ce tableau est inutile puisque les entiers de 1 à n sont bien connus. Par contre, il semble raisonnable de disposer d'un tableau indiquant pour chaque entier s'il a été rayé ou non. Un tableau de **n** "booléens" (entiers en C) s'impose,
- le nombre n est demandé à 1'utilisateur mais ne devra pas excéder la taille prévue pour le tableau de "booléens",
- l'algorithme informatique est le suivant
  1. le tableau de booléens sera initialisé à "false"**,**
  2. commencer l'itération avec une variable *prem* valant 1,
  3. rechercher à partir *de prem* le premier entier non rayé sans toutefois dépasser la valeur limite **n.**
  4. rayer tous les multiples de *prem* dans le tableau,
  5. répéter l'opération jusqu'à ce que la valeur de *prem,* soit supérieure à la racine carrée de **n.**
- Vous écrirez 3 procédures ou fonctions
  - une d'initialisation du tableau de "booléens",
  - une de passage du crible,
  - une d'affichage du résultat.

6 Décomposition

Le problème

Écrire une procédure de décomposition d'un entier en facteurs premiers. Vous utiliserez, bien entendu, ce que nous avons fait sur les nombres premiers dans les précédents TDs (crible d'ératosthene).

Programmation

Nous allons mettre en pratique la compilation séparée de fichiers sources puis l'édition de liens afin de relier un nouveau programme à une bibliothèque de procédures ou fonctions déjà écrites et compilées.

La démarche est la suivante :

1. reprendre votre fichier source du crible d'ératosthene (eratos.cpp) et ajouter une procédure de transformation d'une liste de "booléens" (0 et 1 en C) en une liste d'entiers ;
2. enregistrer ce nouveau source sous le nom eratoslib.cpp ;
3. **définir un nouveau projet** en indiquant à la création du projet que vous souhaitez construire une "librairie statique" (static library) comme type de projet ("Fichier - nouveau projet" et choisir "librairie statique");
4. ajouter "eratoslib.cpp" à ce nouveau projet
5. construire un fichier "eratoslib.h" contenant les "#include <...>", les définitions de constantes, les définitions de types et les prototypes de vos fonctions ;
6. inclure le fichier "eratoslib.h" dans "eratoslib.cpp" (commande #include "eratoslib.h");
7. retirer toute procédure "main" de eratoslib.cpp ;
8. compiler "eratoslib.cpp" pour obtenir un fichier "eratoslib.a"
9. **définir un nouveau projet** de type "Console application"
10. écrire votre ou vos procédures de décomposition en facteurs premiers dans un fichier "decom.cpp" en utilisant toute les procédures nécessaires situées dans "eratoslib" ;
11. inclure "eratoslib.h" dans "decomp.cpp" afin que le compilateur trouve les prototypes des fonctions de "eratoslib" utilisées dans "decomp.cpp" ;
12. établir un lien entre "decomp.cpp" et "eratoslib.obj" afin qu'à l'exécution le programme trouve le code compilé des procédures de "eratoslib". En Dev Cpp, aller dans "Projet - Options du projet " puis dans l'onglet "Paramètres" et inclure le fichier "eratoslib.a" dans la partie "Editeur de liens"
13. compiler et exécuter votre programme de décomposition.

7 Nombre parfait

Le problème

Un nombre *parfait* est un nombre présentant la particularité d'être égal **à la somme de tous ses diviseurs excepté lui-même**. Le premier nombre parfait est le nombre **6** qui est bien égal à 1+2+3.

Deux nombres *a* et *b* sont dits nombres *amis* si la somme des diviseurs de *a* est égale à *b* et la somme des diviseurs de *b* est égale à *a*. Par exemple 220 et 284 sont des nombres amis.

Programmation

Nombres parfaits

Ecrire une procédure *nombres_parfaits* qui retourne la liste des nombres parfaits inférieurs à un entier *nmax* entré au clavier par l'utilisateur dans le programme principal. Pour ce faire, il peut être utile d'écrire la fonction *somme_div* qui retourne la somme des diviseurs d'un nombre entier passé en paramètre.

```
Entrez un entier : 3000
Les nombres parfaits inférieurs à 3000 sont :
6 28 496
```

Nombres amis

Ecrire une procédure permettant de trouver au moins une autre paire de nombres amis. Le programme commence comme celui des nombres parfaits. Dans le cas où **n** n'est pas parfait, il convient de calculer la somme des diviseurs de la somme des diviseurs de **n** et de tester si cette somme est égale à **n**. Dans un second temps, vous pourriez réfléchir à la question : comment éviter la *double détection* des paires de nombres amis ?

```
Entrez un entier : 3000
284 et 220 sont amis
220 et 284 sont amis
1210 et 1184 sont amis
1184 et 1210 sont amis
2924 et 2620 sont amis
2620 et 2924 sont amis
```

8 Tri simple

Le problème

Le but du TD est simple, il s'agit de passer en revue les méthodes de tri les plus simples. Vous avez probablement déjà programmé certains tris, en ce cas, votre tâche n'en sera que plus aisée. Chacun de vos tris sera programmé comme une procédure ou une fonction et le programme principal donnera le choix entre les différentes méthodes.

Il existe de nombreux algorithmes visant à trier un ensemble de données selon un certain prédicat. Nous programmerons deux de ces algorithmes dans ce TD. Pour des raisons de commodité, nous ne traiterons que des listes de n entiers et les rangerons dans un ordre croissant mais le principe reste le même pour n'importe quel type de données et n'importe quel prédicat de comparaison.

Les méthodes de tri les plus connues sont, pratiquement dans l'ordre d'efficacité croissante, les suivantes

- permutation ou *tri bulle,*
- extraction simple et dichotomique,
- insertion simple et dichotomique,
- tri rapide ou *quick sort.*

Programmation

Nous ne programmerons dans ce TD que les tris par permutation et insertion simple et dichotomique, le tri rapide sera traité dans un TD ultérieur.

Tri par permutation

C'est probablement la méthode la plus simple. Cette méthode consiste à ramener, par le biais de permutations, l'élément le plus petit en début de liste (comme une bulle). Puis, ceci fait, il suffit de réitérer le processus en ne considérant plus que les n-1 éléments restant dans la liste.

Cet algorithme modifie la liste initiale et se programme à l'aide de deux boucles imbriquées. L'une parcoure la liste en partant de la tête et contient une seconde boucle partant de la fin, comparant deux éléments consécutifs et les permutant si besoin est.

Ex:

[3, 1, 5, 7, -3, 2] liste initiale

[-3, 3, 1, 5, 7, 2] -3 est mis en tête de liste

[-3, 1, 3, 5, 7, 2] 1 est mis en tête de la liste restante

[-3, 1, 2, 3, 5, 7] 2 est mis en tête de la liste restante

[-3, 1, 2, 3. 5. 71

Tri par insertion

Cette méthode consiste, quant à elle, à construire, élément après élément, une liste triée. Il suffit de partir d'une liste vide et d'y insérer successivement tous les éléments de la liste initiale. Le problème se résume donc à écrire une procédure d'insertion d'un élément dans une liste triée puis de rappeler cette procédure n fois pour obtenir ce que l'on désire.

Reprenons l'exemple précédant

[]

[3]

[1, 3]

[1, 3, 5]

[1, 3, 5, 7]

[-3, 1, 3, 5, 7]

[-3, 1, 2, 3, 5, 7]

Les algorithmes de tri par insertion simple et insertion dichotomique présentent rigoureusement la même structure si ce n'est que :

- insertion simple : la recherche de l'endroit d'insertion se fait en parcourant la liste depuis le début et en comparant l'élément à insérer à l'élément courant,
- insertion dichotomique : la recherche de l'endroit d'insertion se fait par dichotomie (séparation de l'intervalle en deux...).

Procédures complémentaires et structures de données

Pour pouvoir tester nos algorithmes, il est indispensable de disposer d'une procédure d'entrée des éléments d'une liste et d'une procédure d'affichage de listes (probablement déjà écrites lors d'un précédant TD).

De plus, tous les algorithmes de tri nécessitent d'avoir accès à la taille de la liste. Pour ce faire, il y a généralement deux stratégies :

- calculer, lors de l'entrée, le nombre d'éléments et passer celui-ci en argument de toutes les procédures ou fonctions,
- manipuler une structure contenant la taille de la liste et la liste elle même en lieu et place d'une simple liste.

9 Le jeu du Yatzee

Le problème

Certains d'entre vous doivent connaître un jeu de dés dénommé YATZEE. Ce jeu se joue avec 5 dés standards non pipés. Un jet de dés consiste en trois lancés maximum. Le premier lancé se fait avec les 5 dés et les deux lancés suivants se font en choisissant de 0 à 5 des dés à rejouer en fonction de la stratégie choisie (rubrique de score à remplir). Une fois la combinaison de dés établie, le joueur doit choisir dans quelle rubrique de score comptabiliser ce jet.

Le score comporte 10 rubriques :

- *les uns, les deux, les trois. les quatre, les cinq* et les *six :* on additionne les dés portant respectivement 1, 2, ...
- *le brelan :* il faut avoir 3 dés identiques, le score est alors le total obtenu grâce aux 3 dés en question ou 0 s'il n'y a pas de *brelan,*
- *le carré* : même chose que le *brelan* mais il faut 4 dés identiques,
- le *full :* il faut avoir un *brelan* et une *paire,* le score est alors le total des 5 dés ou 0,
- *le yatzee* enfin où tous les dés sont identiques, le score est alors le total de tous les dés ou 0.

Une rubrique de score déjà choisie ne peut être réutilisée dans la même partie. Le jeu se termine lorsque toutes les rubriques de score ont été remplies. Le but final consiste, bien sûr, à obtenir le score le plus élevé possible.

Programmation

Vous êtes libres de choisir les structures de données qui vous semblent le plus adaptées au problème tant pour la tenue du score que pour les jets de dés. Je ne peux que vous suggérer de faire appel aux tableaux d'entiers.

Les différents points à traiter sont les suivants :

- tirage aléatoire*,*
- choix des dès à rejouer : il y a plusieurs solutions possibles mais je pense que vous pouvez choisir parmi les trois suivantes :

1. demande interactive du genre *: Voulez-vous rejouer ce dé (oln)*
2. une chaîne de cinq booléens avec o pour rejouer le dé et n pour le conserver (Ex : onnoo pour rejouer les dés 1, 4 et 5),
3. une chaîne de caractères donnant les dés à rejouer (Exs : ACD, A, ABD, ...).

- tenue du score : il est nécessaire de maintenir à jour une structure indiquant

1. quelles rubriques de score ont déjà été choisies (déclencher une erreur en cas de nouvelle sélection),
2. de manière constante, le score total actuel,

- affichage du jet et du score : cet affichage sera réduit à quelque chose de très simple dans un premier temps et pourra être amélioré par la suite (il y a du temps à passer ...).

L'opération la plus délicate est très probablement la tenue du score et tout particulièrement la reconnaissance des *brelans, carrés et fulls.* Pour ce faire, il peut être utile de disposer d'une fonction comptant le nombre d'occurrences d'un entier dans une liste. C'est à partir de cette fonction que l'on programmera une grande partie de la gestion du score.

Le bel affichage des dès et du score n'est pas du tout le but du TD mais peut donner lieu, après écriture du noyau du programme bien sûr, à utilisation éventuelle de procédures graphiques plus ou moins sophistiquées (pour les plus avancés d'entre vous).