Université Bordeaux 1 Licence **INF 255**
Examen seconde session juin 2008 Durée : 1h30 – Sans document

Projet de programmation 2

On utilisera le langage C pour répondre aux questions.

Implémentation des exceptions en C

1. Rappeler le prototype et le rôle des deux fonctions sur lesquelles se base le mécanisme d'exception en C vu en cours.
2. Quelle structure de données est nécessaire à toute implémentation du mécanisme d'exception ?

Le problème du sac à dos*. Les questions 3 – 4 - 5 sont indépendantes.*

On dispose d'un sac capable de supporter un poids (en gramme) maximal et d'un ensemble d'objets ayant chacun un poids et une valeur (en euro). L'objectif est de remplir le sac d'objets sans dépasser le poids maximal tout en maximisant la somme des valeurs des objets qu'il contient.

Exemple : on dispose d'un sac supportant 3 kg et des objets suivants :

Bague	12g	3400€
Tournevis	270g	7€
Ordinateur	2500g	1234€
Baguette	250g	0,95€

Ne pouvant emporter tous les objets, on sélectionnera les objets « Bague », « Tournevis » et « Ordinateur » pour un montant de 4641€ et un poids de 2782g.

1. Définir un type *objet* afin de mémoriser le nom, le poids et la valeur d'un objet.

2. Définir un type *sac_a_dos* capable de mémoriser l'ensemble initial d'objets (un vecteur d'objets), le poids maximal du sac à dos et de décrire la sélection retenue (sous ensemble des objets, poids et montant du sac).

NB. Pour mémoriser l'ensemble des objets retenus on pourra associer un vecteur de booléens au vecteur d'objets. Ainsi, la solution de l'exemple précédent peut être représentée par le vecteur {true,true,true,false} en référence au tableau d'objets ci-dessus.

Il s'agit de résoudre le problème du sac à dos en testant systématiquement tous les sous ensembles de l'ensemble de départ. Dans ce devoir nous allons programmer deux techniques : une itérative, l'autre récursive.

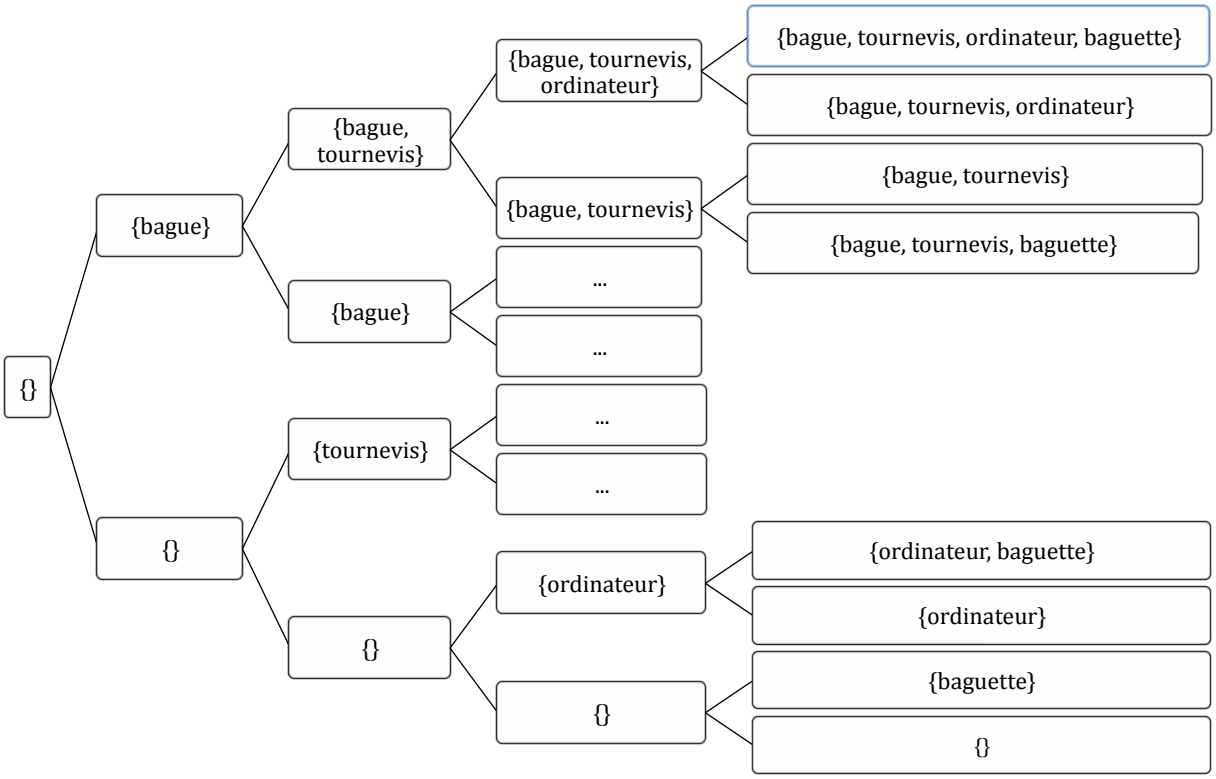
La méthode itérative se base sur une bijection entre les parties d'un ensemble de n éléments et les entiers naturels inférieurs à 2^n : il suffit d'identifier la présence du $i^{ème}$ objet au $i^{ème}$ bit de l'écriture binaire. Ainsi le vecteur {true,true,true,false} de l'exemple précédent correspond à l'entier 1110_2 c'est à dire 14_{10}. Ainsi pour énumérer (passer en revue) tous les sous ensembles, on peut utiliser un indice de boucle parcourant les entiers compris entre 0 (correspondant à l'ensemble vide) et 2^n-1 (l'ensemble initial). Mais plutôt que traduire à chaque tour de boucle l'entier en ensemble, on peut directement réaliser l'opération d'incrémentation sur le vecteur de booléens. Par exemple, pour passer de 1 à 4 (soit 0001 → 0010 → 0011 → 0100) :

{*false, false, false, true*}→{*false, false, true, false* }→{*false, false, true, true*}→{*false, true, false, false}*

3. Écrire la fonction bool incrementer(bool *bit, int nombre_de_bits) qui transforme le tableau bit en utilisant l'algorithme d'incrémentation binaire et retourne true si et seulement si la configuration obtenue contient au moins une fois la valeur true.

4. En utilisant la fonction incrémenter, écrire la fonction résoudre_sac_a_dos() qui prend en paramètre un sac à dos et qui calcule une (la) meilleure solution possible.

La méthode récursive utilise un schéma de résolution récursif analogue à celui employé en cours pour résoudre le problème des n-reines. Ici, il s'agit d'explorer (sans le construire) un arbre binaire complet où à chaque niveau on choisit de sélectionner ou non un objet (ici suivant l'ordre bague – tournevis – ordinateur – baguette) :



5. Écrire la fonction *resoudre_sac_a_dos*() en utilisant une stratégie récursive. On pourra s'aider en définissant des fonctions auxiliaires.

On souhaite accélérer la vitesse de traitement en supprimant le plus possible les calculs inutiles. Par exemple, pour le voyageur de commerce on peut arrêter une exploration dès lors que la longueur de la tournée en cours d'exploration est plus longue que la plus petite tournée calculée jusqu'à lors.

6. Proposer une optimisation basée sur la valeur des objets restant à prendre. Montrer comment cette optimisation peut s'intégrer à la version récursive. Peut-on optimiser de la sorte la version itérative ?