Programmation 1

Devoir maison à rendre le 6 novembre 2023 sur eCampus

Organisation

Ce devoir est **individuel**. Il comprend trois exercices indépendants qui vous demandent de coder en C, LC3 et OCAML. Vous devez **rendre trois fichiers avec le code de chaque exercice** sur eCampus au plus tard le 6 novembre 2023 à 22h.

La notation tiendra compte de la pertinence des réponses et des commentaires contenus dans le code, de la lisibilité du code, de son élégance, de son originalité, et de son efficacité en terme de temps et mémoire utilisés pour l'exécution.

Vous pouvez répondre à une question en supposant avoir une réponse aux questions la précédant. Le barème est indicatif.

Exercice 1 Programmation en C (10 points)

On considère un gestionnaire de mémoire pour des noeuds d'arbres binaire ayant une donnée entière et un pointeur vers le parent. Le code de cet exercice doit être rendu dans un fichier `btree.c`

A. Définissez un type `btree_t` qui encode les valeurs des noeuds de l'arbre. Codez trois macro-définitions pour les valeurs gauches permettant d'obtenir le parent et les fils d'une adresse de type `btree_t`.

```
#define BTREE_PARENT(n) ....
#define BTREE_LSON(n)   ....
#define BTREE_RSON(n)   ....
```

Le gestionnaire de mémoire gère ces noeuds dans un tableau déclaré par :

```
#define MAX_TAB 20000
static btree_t mem[MAX_TAB];
```

L'état (disponible ou réservé) d'une entrée du tableau est mémorisé comme le bit de poids faible du pointeur vers le parent : 1 si entrée disponible, 0 sinon.

B. Codez une macro-définition `BTREE_ISFREE` qui a comme argument une valeur de type `btree_t` et qui renvoie `true` (de type `bool`) si la cellule est disponible et `false` sinon.

C. Codez une fonction `btree_t* btree_alloc(void)` qui réserve un noeud de l'arbre dans `mem` et renvoie son adresse ou `NULL` s'il n'y a pas de noeud libre.

Un **glaneur de cellules** (GC) collecte les noeuds réservés dans `mem` mais non utilisés (accessibles à partir de) par les variables du programme. Pour cela, le GC reçoit en entrée le tableau des adresses des variables du programme [1] et il parcourt la forêt accessible à partir de ces noeuds en marquant les noeuds visités. Ensuite, il parcourt le tableau `mem` et marque comme libres tous les noeuds non disponibles et non visités.

D. Codez une fonction `unsigned int btree_gc(btree* vars[])` qui effectue les opérations d'un GC ; le résultat renvoyé est le nombre de noeuds glanés (réservés mais inaccessibles, donc libérés).

E. Soit le programme ci-dessous. Que affiche-t-il ?

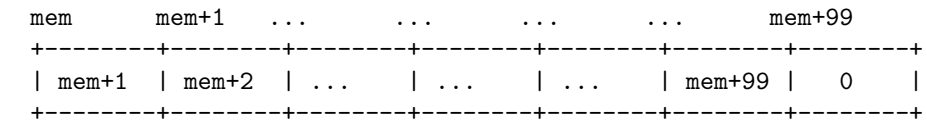
```c
int main(void) {
  btree_t* vars[2];
  vars[0] = btree_alloc();
  vars[1] = btree_alloc();
  BTREE_PARENT(vars[0]) = vars[1];
  BTREE_LSON(vars[1]) = vars[0];
  BTREE_RSON(vars[1]) = BTREE_LSON(vars[0]) = BTREE_RSON(vars[0]) = NULL;
  vars[1] = NULL;
  printf("%d collected nodes\n", btree_gc(vars));
  return 0;
}
```

Exercice 2 Programmation LC3 (6 points)

Le code de cet exercice doit être rendu dans un fichier `list.asm`.

A. Déclarez en LC3 une zone de mémoire étiquetée par `mem` qui contient 100 mots de 16 bits.

B. Ecrivez à l'adresse `x3000` une séquence d'instructions qui permet d'initialiser les premières 99 valeurs à l'adresse `mem` par les adresses entre `mem+1` et `mem+99` ; la dernière valeur est mise à 0.



1. Ceci est une simplification, car un GC général regarde les variables statiques et celle sur la pile.

Avec cette initialisation, la zone `mem` peut être vue comme une liste chainée, la valeur à l'adresse `mem+i` indique l'adresse de la cellule suivante dans la liste ; le debut de la liste est la cellule à l'adresse `mem`.

C. On suppose que les 100 valeurs de la zone étiquetée `mem` ont été permutées. Codez un programme qui calcule en `R0` la longueur de la nouvelle liste, c'est-à-dire le nombre de cellules de liste visitées en utilisant la relation « successeur d'une cellule » `mem+i` est la cellule à l'adresse donnée par la valeur de la cellule `mem+i`.

Exercice 3 Programmation OCaml (4 points)

Le code de cet exercice doit être écrit dans un fichier `modules.ml`

A. Définissez une signature `BTREE` pour les arbres binaires qui contient deux types abstraits `btree` et `node` et les valeurs suivantes :

- `is_empty` pour tester si l'arbre en argument est vide,
- `get_root` pour renvoyer le noeud racine de l'arbre en argument,
- `get_sons` pour renvoyer la liste des noeuds fils d'un noeud en argument,
- `get_label` pour renvoyer le texte qui étiquette le noeud en argument.

B. Définissez un foncteur `SBtree` qui a comme paramètre un module de signature `BTREE` et qui produit une structure qui implémente dans la fonction `post_visit` l'algorithme de parcours post-fixe d'un arbre binaire étiqueté `btree` donnée en argument et qui affiche les étiquettes des noeuds.