INF3172 : TP2

AZIZ SALAH

DATE LIMITE DE LA REMISE : LE LUNDI 9 DÉCEMBRE 2013 AVANT 23H59

1. Objectifs pédagogiques du travail

- Comprendre le fonctionnement de l'allocation contiguë
- Gestion de la mémoire par ramasse miettes et compactage de la mémoire (similaire à la gestion de la mémoire dans Java)

Ces objectifs sont concrétisés par la réalisation d'un gestionnaire d'allocation utilisant le ramasse miettes et le compactage de la mémoire.

2. Description de la réalisation

Ce travail consiste à réaliser un interpréteur pour un mini langage afin de maitriser la gestion de la mémoire et les enjeux autour. Notre mini langage est composé d'instructions permettant de définir un segment de mémoire courant et d'y allouer des tableaux d'entiers pointés par des variables. De même il offre la possibilité de lancer le ramasse miettes ainsi que le compactage du segment de mémoire en cours. Le tableau qui suit présente les détails des instructions du mini langage.

Date: A13.

`M`	affiche la valeur du registre limite (la taille) du segment de mémoire en cours -La taille initiale du segment de mémoire en cours est 32 (entiers) initialisés à 0.
`I <taille>`	libère le segment de mémoire en cours, supprime toute les variables et crée un nouveau segment de longueur <taille> en terme de la taille d'un int. -<taille> est un entier strictement positif -Toute les cases dans le nouveau segment sont initialisées à zéro.
`N <identif> <taille> <list>`	alloue dans le segment de mémoire en cours un tableau (référencé par la variable <identif>) de <taille> entiers et initialise ses cases avec les éléments de `<list>`. La variable <identif> peut être déjà existante. - `<list>` est une séquence d'entiers séparés par des blancs. - On suppose que `<taille>` représente toujours la longueur de `<list>`. `<taille>` ne peut être nulle. - Pour satisfaire cette requête une allocation dans le segment de mémoire courant par l'algorithme first fit est tentée. En cas d'échec le ramasse miettes est lancé. Celui-ci récupère les tableaux déréférencés (ceux qui ne sont référencés par aucune variable et fusionne les troues créées avec les troues voisins s'il en existe. Ensuite on tente une deuxième fois le first fit. En cas d'échec on calcule l'espace libre total. Si celui-ci suffit à la requête, le compactage est lancé puis par first fit est lancé pour l'allocation. En cas d'insuffisance de la mémoire, la variable `<identif>` n'est ni créée ni modifiée et `print_erreur(MEM_INSUFFISANTE)` le signale.
`D <identif>`	supprime la variable `<identif>` mais pas le tableau - Si cette variable n'existe pas, l'appel `print_erreur(NO_VAR)` permet de le signaler.
`R <identif1> <identif2>`	permet d'ajouter ou de modifier `<identif1>` comme une autre référence du tableau référencé par `<identif2>` - Si la variable `<identif2>` n'existe pas cette instruction est ignorée et l'appel `print_erreur(NO_VAR)` permet de le signaler.
`P <pos> <taille>`	affiche <taille> éléments entiers du segment de mémoire en cours à partir de la position <pos>. - Une erreur est signalée si les bornes du segment de mémoire courant sont dépassés en utilisant la fonction appropriée fournie.
`L`	affiche selon l'ordre alphabétique des noms, la liste des variables avec les positions de début des zones qu'elles réfèrent et leurs tailles respectives
`C`	lance le ramasse miettes et le compacteur la mémoire
`Q`	quitte `minimem`

– L'allocation se fait par l'algorithme first fit.
– Si un tableau est déréférencé, c'est le ramasse miettes qui s'en occupera lors de sont passage.
– Le compactage de la mémoire préserve l'ordre des tableaux en mémoire et vise à créer un troue à la fin du segment de mémoire en cours.

3. Exemple d'exécution

L'exécutable implémentant l'interpréteur est appelé `minimem`. Le symbole `#` représente le prompt de l'interpréteur.

Nous allons fournir plusieurs cas d'utilisation afin d'illustrer le comportement attendu qui pourront aussi servir de cas tests.

3.1. **Compilation et exemple simple.**

```
matl> gcc -Wall -std=c99 -o minimem minimem.c util.c
malt> ./minimem
#M
Registre limite : 32
#P 0 32
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0001 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0002 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0003 | [00000] [00000]
#L
Aucune variable
#N var1 5 23 16 888 777 33
#P 0 32
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00023] [00016] [00888] [00777] [00033] [00000] [00000] [00000] [00000] [00000]
0001 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0002 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0003 | [00000] [00000]
#P 28 40
Hors segment en cours
#L
var1(0000000000:0000000005)
#R beta var1
#L
beta(0000000000:0000000005)
var1(0000000000:0000000005)
#I 40
#P 0 40
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0001 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0002 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
0003 | [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000] [00000]
#L
Aucune variable
#I 10
#N var1 8 1 2 3 4 5 6 7 8
#N var2 3 10 20 30
Memoire insuffisante
#Q
malt>
```

3.2. **Cas d'utilisation : Allocation déclenchant ramasse miettes.**

```
#I 20
#N v1 10 1 2 3 4 5 6 7 8 9 10
#N v2 5 51 52 53 54 55
#N v3 3 31 32 33
```

```
#P 0 20
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00001] [00002] [00003] [00004] [00005] [00006] [00007] [00008] [00009] [00010]
0001 | [00051] [00052] [00053] [00054] [00055] [00031] [00032] [00033] [00000] [00000]
#D v2
#L
v1(0000000000:0000000010)
v3(0000000015:0000000003)
#N v0 3 4 44 444
#L
v0(0000000010:0000000003)
v1(0000000000:0000000010)
v3(0000000015:0000000003)
#P 10 10
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0001 | [00004] [00044] [00444] [00054] [00055] [00031] [00032] [00033] [00000] [00000]
```

3.3. **Cas d'utilisation : Allocation déclenchant ramasse miettes et compactage.**

```
#I 20
#N v1 10 1 2 3 4 5 6 7 8 9 10
#N v2 4 51 52 53 54
#N v3 3 31 32 33
#D v2
#P 0 20
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00001] [00002] [00003] [00004] [00005] [00006] [00007] [00008] [00009] [00010]
0001 | [00051] [00052] [00053] [00054] [00031] [00032] [00033] [00000] [00000] [00000]
#L
v1(0000000000:0000000010)
v3(0000000014:0000000003)
#N v8 5 6 66 666 6666 66666
#L
v1(0000000000:0000000010)
v3(0000000010:0000000003)
v8(0000000013:0000000005)
#P 0 20
----------0---|---1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|
0000 | [00001] [00002] [00003] [00004] [00005] [00006] [00007] [00008] [00009] [00010]
0001 | [00031] [00032] [00033] [00006] [00066] [00666] [06666] [66666] [00000] [00000]
```

4. Exigences non fonctionnelles

4.1. **Gestion de l'allocation dynamique.** Le programme doit s'adapter dynamiquement avec la taille des données en utilisant l'allocation dynamique. Tout espace alloué dynamiquement doit être libéré dès qu'il n'est plus requis.

4.2. **Gestion des erreurs.** Seules les erreurs considérées par la fonction `print_erreur` sont à traiter dans votre programme.

4.3. **Composition du programme.**

- `«util.h»` : définit les prototypes de fonctions utilitaires à utiliser pour l'affichage ainsi que les structures de données utilisées. Le fichier `«util.h»` ne doit pas être modifié.

- «`util.c`» : est une implémentation complète qui vous fournit les fonctions utilitaires pour l'affichage. Les fonctions fournies dans ce fichier ne doivent pas être modifiées.
- «`minimem.c`» : comporte la fonction `main` et vos fonctions selon votre conception.
- La compilation de votre programme «`minimem.c`» se fait par la commande :
  `gcc -Wall -std=c99 minimem.c util.c -o minimem`
- La compilation de votre programme ne devrait donner aucun avertissement (warnning).
- Votre fichier «`minimem.c`» doit contenir les noms des auteurs et leurs codes permanents en commentaire.
- L'affichage doit respecter les exemples dans les cas d'utilisation fournis.
- Tout affichage doit se faire seulement avec les fonctions fournies :
  - `print_limit`
  - `print_var`
  - `print_zone_contenu`
  - `print_prompt`
  - `print_erreur`

4.4. **Portabilité du programme.** Afin de s'assurer de la probabilité de votre programme, celui-çi devrait être compilé et testé sur les serveurs malt et rayon1 (ou rayon2). Le premier serveur roule sur Linux alors que les deux derniers sont des machines Sun.

5. Ce que vous devez remettre

En équipe d'au plus deux personnes, un membre de l'équipe seulement remettra votre fichier «`minimem.c`» électroniquement dans Moodle en suivant le lien approprié. Pas de remise en double svp.

6. Pondération

- Tests de fonctionnement : 70%
- Exigences non fonctionnelles : 20%
- Structure du programme, commentaires, indentation ... : 10%

Votre travail sera corrigé en le soumettant à une batterie de tests. Si jamais aucun test ne marche la note maximale sera 20%

Remarques importantes :

- Un programme ne compilant pas se verra attribuer la note 0.
- Des pénalités sont à prévoir si le programme ne respecte pas l'affichage exigé.
- Aucun programme reçu par courriel ne sera accepté. En cas de panne des serveurs, un délai supplémentaire vous sera accordé sans pénalité de retard.
- Les règlements sur le plagiat seront strictement appliqués.
- 10% comme pénalité de retard par journée entamée. Après cinq jours, le travail ne sera pas accepté.
- La remise d'un fichier "zippé" donne lieu à 10% de pénalité.