TME 7 – GESTION MÉMOIRE

IMPLANTATION D'UNE GESTION DE TAS

Le tas est une zone mémoire réservée par le système pour permettre à un programme de faire de l'allocation dynamique (malloc, free). L'objectif de ce TME est de simuler une gestion simplifiée du tas.

On suppose ici que le tas est une zone de taille fixe égale à 128 octets.

On se propose de programmer les primitives `tas_malloc()` et `tas_free()` qui permettent respectivement d'allouer et de libérer une zone dans le tas :

`char *tas_malloc(unsigned int taille);`

réserve dans le tas une zone de `taille` octets. Cette fonction retourne l'adresse du début de la zone allouée. En cas d'erreur (si l'allocation est impossible), la fonction retourne `NULL`.

`int tas_free(char *ptr);`

libère la zone dont le début est désigné par `ptr`.

Pour gérer les espaces occupés et les espaces libres dans le tas, on utilise les structures de données suivante :

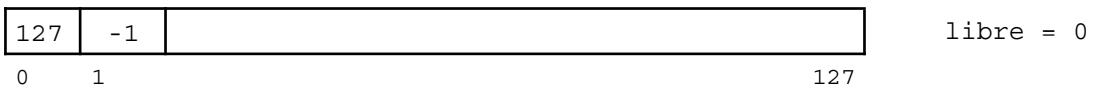
- une zone allouée contient 2 champs :
  - un octet donnant la taille `TD` de la donnée stockée,
  - la donnée elle-même.

  La taille de la zone est donc `TD+1`.
- une zone libre contient 2 champs :
  - un octet donnant la taille `TL` de la donnée pouvant être stockée dans la zone,
  - un octet donnant l'indice dans le tas du début de la zone libre suivante. Si la zone libre est la dernière, cet octet prend la valeur –1.

  La taille de la zone est donc `TL+1`.

On dispose en outre d'une variable `libre` contenant l'indice de début de la première zone libre du tas.

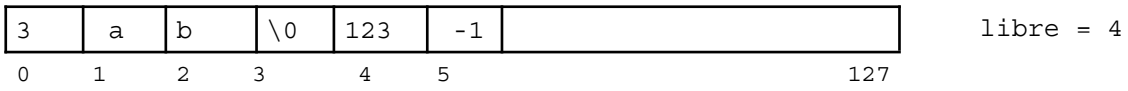
Initialement le tas est vide. On a donc l'image suivante :



Après exécution de l'opération

```
p1 = (char *) tas_malloc(3);
strcpy( p1, "ab" );
```

on obtiendrait l'image :



1. EXECUTION MANUELLE

1.1.

On suppose un tas initialement vide. Donnez l'apparence du tas après l'insertion d'une donnée de taille maximum.

1.2.

On suppose un tas initialement vide, et on utilise une stratégie d'allocation de type first-fit. Représentez l'apparence du tas après l'exécution des opérations suivantes, en précisant à chaque fois la valeur de la variable `libre` :

```
char *p1, *p2, *p3, *p4, *p5;
p1 = (char *) tas_malloc(10);
p2 = (char *) tas_malloc(9);
p3 = (char *) tas_malloc(5);
strcpy( p1, "tp 1" );
strcpy( p2, "tp 2" );
strcpy( p3, "tp 3" );
tas_free( p2 );
p4 = (char *) tas_malloc(8);
strcpy( p4, "systeme" );
```

2. PROGRAMMATION

L'allocation d'une zone dans le tas s'effectue en deux étapes :

- recherche suivant une stratégie prédéfinie (best-fit, worst-fit, first-fit) d'un emplacement libre d'une taille suffisante ;
- réservation de cet emplacement pour stocker la donnée : mise à jour de la variable `libre` et du chaînage des blocs.

Quelle que soit la stratégie utilisée, le prototype de la fonction de recherche est le suivant :

```
int strategie(int taille, int *pred);
```

recherche dans le tas une zone libre de `taille` octets. Cette fonction retourne l'adresse du début de la zone, -1 si aucune zone de taille suffisante n'existe. `*pred` est l'adresse dans le tas du début de la zone précédant la zone retournée.

Les fichiers utiles au TME se trouvent dans le répertoire :

`/Infos/licence/2003/sys/TME10`

Vous y trouverez une bibliothèque avec une fonction d'initialisation du tas et une fonction affichant le contenu du tas, ainsi que le Makefile correspondant.

2.1.

Programmez la fonction `first_fit()`.

2.2.

Programmez la fonction `tas_malloc()` en implantant une stratégie first-fit.

2.3.

Programmez la fonction `tas_free()`.

2.4.

Programmez le jeu d'essai de la question 1.2 et affichez l'apparence du tas. Vous pourrez pour cela utiliser la fonction `afficher_tas()` définie dans `affiche_tas.h`.