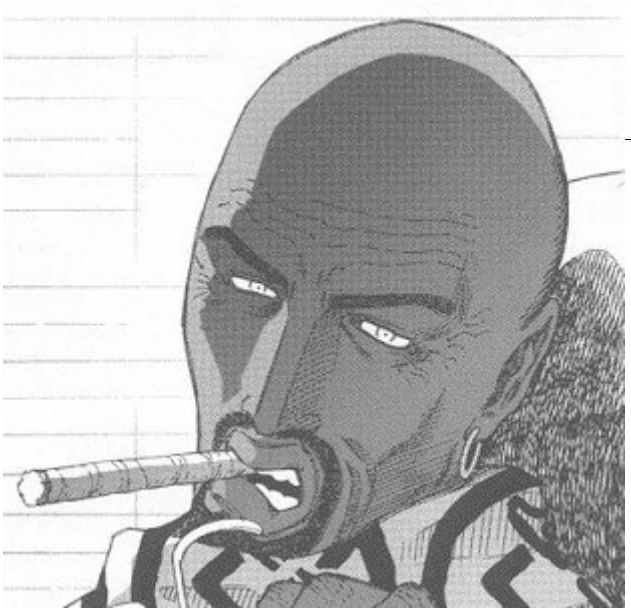

Vector

Parait qu'il peut vous envoyer sur Zalem.
Encore faut il vous stocker...

Fonctions autorisées :
malloc, calloc, realloc, free

Indices de fonctions à reproduire :
memcpy

Un vecteur est un **tableau** supervisé par un ensemble de fonction et qui propose des possibilités de modification de longueur. En C++, le conteneur std::vector<type> est l'incarnation de cette construction, c'est une collection capable de stocker de manière contiguë des éléments. Il s'agit aussi d'un type de conteneur dynamique, ce qui signifie que sa taille peut-être modifiée pendant l'exécution du programme. Le vecteur peut comporter une stratégie d'allocation permettant d'améliorer ses performances : c'est pour cela que *taille* et *capacité* sont **découplés**. L'insertion ou la suppression d'un élément peut entraîner la réallocation de la mémoire pour agrandir ou réduire la taille du tableau sous-jacent.

Considérez la structure suivante :

```
typedef struct      s_vector
{
        void        *data_array;        // Un pointeur vers un espace mémoire de stockage
        size_t      sizeof_data;        // La taille d'un unique élément géré par vecteur
        size_t      array_capacity;     // La capacité de stockage actuelle du vecteur
        size_t      data_count;         // Le nombre d'élément actuellement stocké dans le vecteur
}               t_vector;
```

Écrivez la fonction suivante, associé à la macro suivante :

```
t_vector        *_efvector_new(size_t        sizeof_data,
                               size_t        initial_capacity);
#define         efvector_new(type, initial_capacity)
```

La fonction _efvector_new créée et renvoi un t_vector initialisé, prêt à l'emploi. Les données stockés dans le vector ne **sont pas des pointeurs mais des valeurs copiées.**

La macro devra appeler **_efvector_new** en transformant le type passé en paramètre en taille de donnée.

Écrivez la fonction suivante, qui détruit le vector et les données stockées :

```
void            efvector_delete(t_vector        *vector);
```

Écrivez maintenant les deux fonctions suivantes :

```
void            *efvector_push(t_vector       *vector,
                               const void     *data);
```

Cette fonction ajoute un élément à la fin du vector, en effectuant une « deep copy », c'est à dire en copiant tous les octets pointés par data dans l'espace de stockage du vector. Un pointeur vers l'espace ou est stocké l'élément est renvoyé. NULL est renvoyé en cas d'erreur.

```
void            efvector_pop(t_vector       *vector);
```

Cette fonction retire le dernier élément depuis la fin du vector.

Écrivez enfin, la macro suivante :

```
#define         efvector_at(vector, position, type)
```

Cette macro renvoi un **pointeur** vers la donnée stockée à la case position dans vector, et lui attribut le type **type**.

Pour aller plus loin :

```
bool        efvector_capacity(t_vector *vector,
                              size_t   newsize);
```

Qui étend la capacité maximale actuelle du vecteur.

```
void        efvector_clear(t_vector *vector);
```

Qui vide le vecteur de son contenu – mais ne réinitialise pas la capacité, évidemment.

```
t_vector    *efvector_view(t_vector *vector,
                           size_t   start,
                           size_t   len);
```

Qui renvoi un vecteur contenant une **collection de pointeurs pointant sur les données du vecteur original**, données considérées à partir de start et sur len cases.

```
void        efvector_swap(t_vector *vector,
                          size_t   a,
                          size_t   b);
```

Qui inverse l'élément situé à la position a avec l'élément situé à la position b.