TP 7bis : structures

Semaine du 8 Janvier 2024

Exercice A. Utilisation sur la pile de structures

À la place d'utiliser le complément à deux pour les entiers relatifs, on utilise une structure qui contient la valeur absolue du nombre ainsi que le signe du nombre.

Le type que l'on utilise est le suivant :

```
typedef struct _relatif {
    unsigned int absolue;
    bool signe;
} relatif_t;
```

Le booléen `signe` vaut vrai si le nombre est négatif.

1. Comment représenter le nombre 8 de cette manière ? −4 ?
2. Que représente le relatif suivant :

   ```
   relatif_t a = {12, true}
   ```

   On peut initialiser une structure avec un générateur. Cela ne peut pas être utilisé pour modifier une valeur déjà déclarée.
3. Quelles sont les représentations de 0 ?
4. Proposer une fonction de prototype `void afficher(relatif_t rel)` qui affiche un entier relatif en entrée.
5. Proposer une fonction de prototype `relatif_t ajouter(relatif_t a, relatif_t b)` qui calcule la somme de deux entiers relatifs.

Exercice B. Utilisation des structures pour renvoyer plus d'une valeur

On se donne le type suivant :

```
struct _resultat {
    int quotient ;
    int reste;
};
```

1. Comment utiliser `typedef` pour créer un type `resultat_t` qui soit équivalent à `struct _resultat` ?
2. Comment aurait-on pu faire cette définition en une seule instruction ?
3. En utilisant ce type, proposer une fonction de prototype `resultat_t division_euclidienne(int dividende, int diviseur)` qui renvoie le quotient et le reste de la division euclidienne du dividende et du diviseur à l'aide du type `resultat_t`.

Exercice C. Ordre de priorité des opérateurs

On se donne les type suivants :

```c
struct _a {
    int c1;
};

struct _b {
    int * c2;
};

typedef struct _a a_t;
typedef struct _b b_t;
```

1. On se donne `a_ptr` de type `a_t *`, et `b` de type `b_t`.
   Expliquez la différence entre les deux codes suivants :

```c
(*a_ptr).c1;
```

```c
*b.c2;
```

2. Dans quel cas aurions nous pu utiliser l'opérateur `->` ?

Exercice D. Allocation et modification d'une structure sur le tas

On se donne les types suivant :

```c
struct _somme {
    int total;
    int dernier_ajout;
};
typedef struct _somme somme_t;
```

L'objectif est de se servir de ce type pour pouvoir représenter un accumulateur dans lequel on ajoute des entiers. On veut pouvoir se servir du champs `dernier_ajout` pour pouvoir revenir d'une étape en arrière dans le calcul.

L'application est donc la suivante : initialement, les deux grandeurs sont nulles :

Total : 0	Dernier Ajout : 0

Je peux ajouter des entiers, par exemple 2 puis 3, ce qui donne successivement :

Total : 2	Dernier Ajout : 2

puis

Total : 5	Dernier Ajout : 3

Si je veux revenir en arrière, je peux le faire sur une étape :

Total : 2	Dernier Ajout : 0

Mais je ne peux pas revenir plus en amont à cause du fait que je ne me souvenais que de la dernière étape.

1. Proposer une fonction de prototype `somme_t * nouveau();` qui alloue sur le tas un objet de type `somme_t` vide (avec un total de 0 et un dernier ajout de 0) et qui renvoie un pointeur vers cette structure.

2. Proposer une fonction de prototype `void ajouter(somme_t * s_ptr, int n)` qui ajoute un entier dans la somme et mets à jour le dernier ajout.
3. Proposer une fonction de prototype `void retour_arriere(somme_t * s_ptr)` qui modifie une somme pour pouvoir revenir d'une étape en arrière. On modifiera le champs `dernier_ajout` pour obtenir 0.
4. Proposer une fonction de prototype `void detruire(somme_t * s_ptr)` qui libère la mémoire associée à une somme passée en argument.
5. Dans cette application, nous ne pouvons revenir que d'une étape en arrière. Comment aurait-on pu procéder pour pouvoir revenir d'un nombre arbitraire d'étapes en arrière ?

Exercice E. Taille d'une structure (🔧)

1. On se donne les type suivant :

```c
struct _a {
    int16_t c1;
    int8_t c2;
}
```

```c
struct _b {
    int8_t c1;
    int8_t c2;
    int8_t c3;
}
```

Quelle est la taille minimale qu'il faudrait en théorie pour stocker un élément de chacune de ces structures ? Quelle est la taille donnée par `sizeof` ?

2. À l'aide de la notion d'alignement, expliquer les raisons de cette différence.

L'alignement des données en pratique dépend de l'implémentation du compilateur.