

0.1 Création de nouveaux types

Bien que C soit assez riche, il y'a certains types que vous retrouvez dans d'autres langages, que vous ne retrouvez pas en C. Quelques exemples : Types restreints, intervalles d'entiers, booléens, chaînes de caractères, tableaux dynamiques, tableaux non contraints ... Néanmoins, il vous est offert la possibilité de "créer" de nouveaux types, il seront évidemment locaux au exécution de votre programme.

0.1.1 1. Quelques nouveaux types :

Pour fixer les idées, j'associerai chaque concept à un exemple.

0.1.2 1.1 Créer un type booléen :

Pour créer un type booléen vous devez créer un type énuméré, ceci peut se faire par : **typedef enum.**

– **Voici un petit programme qui affiche vrai si votre booléen est vrai et faux sinon.**

```
#include <stdio.h>
typedef enum { true, false } boolean;
void vrai(boolean a)
{if (a==true) printf("\nvrai\n");
else printf("\nfaux\n");}
int main (void)
{vrai(true); return(1);}
```

Ce programme affiche vrai, mais il aurait pu afficher faux si j'avais appelé vrai() avec false.

0.1.3 1.2 Créer un n-uplets :

Supposons que nous souhaitions gérer les comptes d'une boulangerie, pour cela, à chaque facture nous associons un numéro et la somme à prélever. Ceci se réalise très facilement grâce au constructeur "**struct**".

– **Voici un exemple de code :**

```
#include <stdio.h>
typedef struct {int NumFact; float Somme;} Facture;
int main (void)
{
Facture fact1;
fact1.NumFact=1;
fact1.Somme=2000;
printf("\nLa facture %d indique de prélever %f FF\n", fact1.NumFact, fact1.Somme);
return(1);
}
```

– **Qui affiche :**

La facture 1 indique de prélever 2000.000000 FF.

Notez que pour accéder à un champ j'utilise le sélecteur "." suivi du nom du champ.

0.1.4 1.3 Création du type chaîne de caractère :

Pour représenter une chaîne de caractère en C, on utilise un tableau, comportant plusieurs données de type char, dont le dernier élément est le caractère nul '\0', c'est-à-dire le premier caractère du code ASCII (dont la valeur est 0). Ce caractère est un caractère de contrôle (donc non affichable) qui permet d'indiquer une fin de chaîne de caractères. Ainsi une chaîne composée de n éléments sera en fait un tableau de n+1 éléments de type char.

– **Voici un exemple mettant en oeuvre cela :**

```
#include <stdio.h>
#include <string.h>
typedef char ChaîneCar[20];
int main(void)
{
    ChaîneCar Chaîne1;
    ChaîneCar Chaîne2 = {'B','o','n','j','o','u','r','2'};
    Chaîne1[0]= 'B' ;
    Chaîne1[1]= 'o' ;
    Chaîne1[2]= 'n' ;
    Chaîne1[3]= 'j' ;
    Chaîne1[4]= 'o' ;
    Chaîne1[5]= 'u' ;
    Chaîne1[6]= 'r' ;
    Chaîne1[7]= '1' ;
    Chaîne1[8]= '\0' ;
    printf("\n la chaîne de caractère 1 a %d caractères",
        strlen (Chaîne1)) ;
    printf("\n la chaîne de caractère 2 a %d caractères\n",
        strlen (Chaîne2)) ;
    return(1) ;
}
```

Comme vous le voyez, vous pouvez utiliser l'ensemble des fonctions propres aux chaînes de caractères, d'où l'"#include <string.h>"

– **L'exécution de ce programme vous affichera :**

```
la chaîne de caractère 1 a 8 caractères
la chaîne de caractère 2 a 8 caractères
```

Le caractère '\0' étant ignoré. Notez, que l'agrégat de tableau (remplissage du tableau Chaîne2) n'est possible qu'à la déclaration du tableau.

0.1.5 2. Les listes chaînées :

Une liste chaînée correspond à une structure dont au moins un des champs contient un pointeur vers une structure de même type. De cette façon on crée des éléments (appelés parfois noeuds ou liens) contenant des données, mais, contrairement à un tableau, celles-ci

peuvent être éparpillées en mémoire et reliées entre-elles par des liens logiques (des pointeurs). Lorsqu'une structure contient un pointeur vers son successeur ou précédent, l'ensemble des structures forme une liste chaînée. Si une structure contient à la fois un pointeur vers le précédent et le suivant, on parle de liste chaînée double. Si la structure contient deux pointeurs vers 2 structures suivantes (un pour chacune) on parle d'arbre binaire. Enfin, si la structure contient n pointeurs vers n suivants on parle d'arbre n-aire.

– **Voici un programme qui manipule une liste chaînée :**

```
#include <stdlib.h>
#include <stdio.h>
typedef struct _unDoublet {
    int rang;
    struct _unDoublet *suc;
} Doublet;
typedef Doublet *adDoublet;
int main(void){
    adDoublet premier,milieu,milieu2,dernier,courant,suivant;
    printf("construction de la liste ...");
    premier = (adDoublet) malloc (sizeof (Doublet));
    premier->rang = 1;
    milieu = (adDoublet) malloc (sizeof (Doublet));
    milieu->rang = 2;
    premier->suc = milieu;
    milieu2 = (adDoublet) malloc (sizeof (Doublet));
    milieu2->rang = 3;
    milieu->suc = milieu2;
    dernier = (adDoublet) malloc (sizeof (Doublet));
    dernier->rang = 4;
    milieu2->suc = dernier;
    dernier->suc = NULL;
    printf("\naffichage de la liste ...");
    for (courant = premier; courant->suc != NULL;courant =
        courant->suc)
        {printf("\n'element no %d pointe vers le no %d",
            courant->rang, courant->suc->rang);}
    printf("\n'element no %d n'a pas de successeur", dernier->rang);
    printf("\nliberation de la memoire ... \n");
    courant = premier;
    while (courant != NULL)
        {suivant = courant->suc;
        free (courant);
        courant = suivant;}
    return(1);
}
```

– **Le programme affiche :**

construction de la liste ...
affichage de la liste ...
l'élément no 1 pointe vers le no 2
l'élément no 2 pointe vers le no 3
l'élément no 3 pointe vers le no 4
l'élément no 4 n'a pas de successeur
libération de la mémoire ...

Je manipule ma liste par pointeur, c'est pourquoi je définis 4 pointeurs de doublet (ad-Doublet), qui sont : **premier, milieu, milieu2 et dernier**. Ils désignent respectivement les éléments 1,2,3 et 4 de ma liste. **dernier(4)** pointant vers NULL. Ceci permet lors d'un parcours de liste d'en connaître la fin. Pour chaque nouveau maillon de ma liste, je dois allouer dynamiquement de la mémoire grâce à la fonction **malloc()**, l'allocation se fait en fonction de l'objet pointé d'où le **sizeof**, qui renvoie ici la taille d'un Doublet. Pour utiliser ces fonctions j'ai besoin d'inclure la **stdlib.h**. J'accède à l'un des champs de la structure pointé par une "**flèche**" au bout de laquelle j'indique le champ qui m'intéresse. Après usage je dois restituer la mémoire que j'ai utilisée, ceci se fait à l'aide de **free()** qui prend en paramètre le pointeur à libérer.

0.1.6 Conclusion :

Bien que **struct** et **typedef** aient souvent été utilisés ensemble, il est bien-sûr possible de les utiliser séparément

– **quelques exemples :**

- `typedef char T[100]; /*définit le type T comme un tableau de 100 caractères (donc indicé de 0 à 99)*/`
- `struct {int i; double f; } S /* déclare S, une structure à 2 champs le premier étant un entier le second un double */`

«« Précédent¹ Suivant »»²

¹<http://www.trustonme.net/didactels/155.html>

²<http://www.trustonme.net/didactels/157.html>