Aide mémoire de C

par A.Glad - mai 2011

Liste des types simples utilisés dans ce cours

Les entiers

- type `unsigned int`, pour les entiers naturels
- type `int`, pour les entiers relatifs

Les flottants

- type `float`, nombre flottant en simple précision
- type `double`, nombre flottant en double précision. Plus précis mais plus coûteux en mémoire.

Les booléens type `bool`. Peuvent prendre deux valeurs : `true` (vrai) ou `false` (faux). Nécessite l'inclusion de la librairie `<stdbool.h>`.

Les caractères type `char`. ex: `char c = 'a'` assigne le caractère a à la variable c.

Les constantes Les constantes se définissent dans le pré-processeur (avant les fonctions). On définira une constante comme suit: `#define PI 3.14159`

Déclaration de variables

```
unsigned int somme; //déclaration de la variable 'somme' de type entier naturel
char lettre, autreLettre; //déclaration de deux variables de type caractère : 'lettre' et 'autreLettre'
```

Structures de contrôle

conditionnelle

```
if(expression_booléenne)
   {
       /*une ou plusieurs instructions*/
   }
else
   {
       /*une ou plusieurs instructions, optionnel*/
   }
```

boucles

- pour $a \leq b$:

```
for(i=a; i<=b; i=i+1)
   {
      /*une ou plusieurs instructions*/
   }
```

- pour $a \geq b$:

```
for(i=a; i>=b; i=i-1)
   {
      /*une ou plusieurs instructions*/
   }
```

-
```
while(condition_d'arrêt) //avec condition_d'arrêt, une expression booléenne
   {
      /*une ou plusieurs instructions*/
   }
```

Les fonctions

```
type_de_retour nom_de_la_fonction(type_1 parametre_1, ... , type_n parametre_n)
{
    /*une ou plusieurs instructions*/
    return e; //e est une expression de type 'type_de_retour'
}
```

Affichage

Nécessite l'inclusion de la librairie `<stdio.h>`
`printf("%d / %d = %d", 8, 2, 8/2);` affiche `8 / 2 = 4`.
Le premier `'%d'` correspond au deuxième paramètre de la fonction (8 dans l'exemple), le deuxième `'%d'` au troisième paramètre (2 dans l'exemple), etc. On utilisera `'%d'` pour l'affichage des entiers, `'%f'` pour l'affichage des flottants, `'%c'` pour l'affichage des caractères et `'%s'` pour les chaînes de caractères.

Les enregistrements

La structure personne permet de stocker et d'accéder à l'âge, la taille et le sexe d'une personne.

```
struct Personne
   {
      unsigned int age;
      float taille;
      char sexe;
   };
typedef struct Personne personne;

personne nouvellePersonne()
   {
      age = AGE_PAR_DEFAUT;
      taille = TAILLE_PAR_DEFAUT;
      sexe = SEXE_PAR_DEFAUT;
   }
```

```
personne ecrire_age(int a, personne p)
   {
      p.age = a;
      return p;
   }

void exemple()
   {
      personne p;
      float taille;
      p = nouvellePersonne();
      p = ecrire_age(23, p); //écriture de l'âge
      taille = lire_taille(p); //lecture de la taille
   }
```

Les tableaux

`int tab[TAILLE_TABLEAU];` définit un tableau (non initialisé) d'entiers de taille `TAILLE_TABLEAU`. Les indice d'un tableau vont de `0` (premier élément) à `TAILLE_TABLEAU-1` (dernier élément). `tab[i]` permet d'accéder à l'élément d'indice i du tableau. Exemple : `tab[2] = 5;` assigne la valeur 5 au troisième élément du tableau `tab`.
Les chaînes de caractères sont des tableaux d'entiers. Exemple: `char nom[5]; nom = "toto";` On notera que la taille du tableau doit toujours être strictement supérieure à la taille de la chaîne. Dans ce cas, `nom` est un tableau de 5 caractères. `nom[0]` vaut '`t`', `nom[3]` vaut '`o`' et `nom[4]` vaut '`\0`' (caractère de fin de chaîne).

Les listes

```
struct Liste
   {
      int valeur;
      liste suivant;
   };
typedef struct Liste *liste;

liste l_vide()
   {
      return NULL;
   }

bool est_vide(liste L)
   {
      return suivant==NULL;
   }

liste reste (liste L)
   {
     return (*L).suivant;
   }
```

```
int prem(liste L)
   {
      if(est_vide(L))
        {
          printf("La liste est vide!");
          exit(0);
        }
      return (*L).valeur;
   }

liste cons (int x, liste L) //ajout d'un élément
   {
     liste M ;
     //réservation de la place mémoire nécessaire pour une cellule :
     M = malloc (sizeof (*M));
     (*M).valeur = x;
     //on peut aussi écrire " M->valeur = x; "
     (*M).suivant = L;
     return M;
   }
```