Les fonctions

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

Table des matières

Table des matières

Les modules/fonctions

Le langage C permet de découper un programme en plusieurs parties nommées souvent « fonctions/modules ». On dit aussi programmation modulaire. Elle permet :

- d'éviter la répétition des instructions.
- de paramétrer certains modules et de les réutiliser.

Comme nous l'avons déja dit :

- Une **fonction** est un ensemble d'instructions qu'on écrit une seule fois en leur attribuant un nom.
- Une fonction peut être paramétrée, de façon que son travail puisse s'adapter à des situations semblables, mais non identiques.

```
fonction nom   - - en-tête
{
   Instructions   - - corp de la fonction
}
```

Les modules/fonctions

- Comme nous l'avons vu, chaque programme C doit contenir une fonction applepée **main()**. Le fonction **main** peut appeler n'importe quel autres fonctions.

- Dans les fonctions **main()** écrites jusqu'à présent, nous avons fait appel a des fonctions tel que la fonction **printf()** et la fonction **scanf ()** qui sont des fonction du langage C.

- Nous allon voir comment :
  1. écrire nos propres fonctions,
  2. leur transmettre des données,
  3. traiter les données transmises
  4. renvoyer un résultat.

Déclaration d'une fonction en C

- Le but d'une fonction C, qu'elle provienne d'une bibliothèque (fonction interne) ou écrite par l'utilisateur, est de recevoir des données, opérer sur ces données et renvoyer directement au plus une valeur unique.

- Nous savons que l'appel de la fonction **printf()** se fait en donnant le nom de la fonction et en passant toutes les données (arguments) entre parenthèses.

```
printf("Bonjour");
```

- La fonction appelée doit pouvoir accepter les données qui lui sont transmises. Seulement après réception des données, elles peuvent être manipulées pour produire un résultat utile.

Déclaration d'une fonction en C

type de retour nom_fonction(liste de type des arguments);
Regardons l'exemple suivant :

```c
#include <stdio.h>
main( )
{
float Nbr_1, Nbr_2, max_Nbr;
/* Déclaration de la fonction */
float fonct_max (float, float); /* Le prototype de la function */
printf("Entrer le 1er nombre : ");
scanf ("%f", &Nbr_1);
printf("\n Entrer le 2nd nombre : ");
scanf ("%f", &Nbr_2);
max_Nbr = fonct_max(Nbr_1,Nbr_2);/*Appel de la function fonct_max*/
printf("\nLe max des deux nombres est %f", max_Nbr);
}
```

Ici nous avouns appelé la fonction **fonct_max** au sein de la fonction **main**.
Pour utiliser une fonction, il est obligatoire de la déclarer dans la fonction appelante.

Définition d'une fonction en C

- Comme nous l'avons cité précédemment, chaque fonction C se compose de deux parties, **l'entête** et le **corps**.
- L'**entête** de la fonction permet d'identifier le type de données de la valeur retournée par la fonction, de fournir à la fonction un nom et de spécifier le nombre et l'ordre et des arguments.
- Le **corps** de la fonction permet d'opérer sur les données transmises et renvoyer, au plus, une valeur à la fonction appelante.
- L'**entête** d'une fonction se compose d'une seule ligne qui contient le type de valeur renvoyée par la fonction, son nom et les noms et types de données de ses arguments.

```
float fonct_max (float N1, float N2)
```

Les noms d'arguments dans la ligne d'**entête** sont formellement appelés **paramètres** ou **arguments** formels, et nous les utiliserons d'une manière interchangeables.

Définition d'une fonction en C

- Le mot clé **void** est utilisé pour déclarer que la fonction ne renvoie aucune valeur. Par exemple, l'entête

```
void afficher (float N1, float N2)
```

declare une fonction **afficher** qui ne renvoie aucune valeur et qui prend deux paramètres de type **float**.

- Pour déclarer une fonction qui ne prend pas de paramètres, la fonction **afficher** peut être définie comme suit :

```
void afficher ()
```

Définition d'une fonction en C

- Pour définir le corps de la fonction **fonct_max** qui cherche le plus grand des deux nombres passés en arguments et le renvoie à la fonction appelante (main).

```
void afficher (float N1, float N2)
{ /* Début du coprs de la function */
float max_nbr;  /* déclaration d'une variable */
if (N1 >= N2) /* cherche le nombre maximum */
    {
        max_nbr = N1;
    }
else
    {
        max_nbr = N2;
    }
return (max_nbr) /* renvoie la valeur de max_nbr */
/* fin */
}
```

Table des matières

Transmission par valeur des arguments

Nous avons déjà parlé de la transmission par valeur des argument lors du cours de pseudocode. Voyons le fonctionnement de ce mode en C :

```c
#include <stdio.h>
main()
{ void echange (int a, int b) ;
int n=10, p=20 ;
printf ("avant appel : %d %d\n", n, p) ;
echange (n, p) ;
printf ("après appel : %d %d", n, p)
}
void echange (int a, int b)
{
int c ;
printf ("début echange : %d %d\n", a, b) ;
c = a ;
a = b ;
b = c ;
printf ("fin echange : %d %d\n", a, b) ;
}
```

Le programme affiche

```
avant appel : 10 20
début echange : 10 20
fin echange : 20 10
après appel : 10 20
```

Il y a eu une transmission par valeur, c.à.d les valeurs de n et p ont été recopiées localement dans la fonction **echange**.

Table des matières

Variables globales

- En C, nous appelons variable globale toute variable partagée ou commune entre plusieurs fonctions.

```c
#include <stdio.h>
int i;
main()
{ void afficher ();
for (i=1 ; i<=5 ; i++)
{
    afficher();
}
}
void afficher(void)
{
    printf ("j'affiche la valeur de i pour la %d fois\n", i);
}
```

Le programme affiche : j'affiche la valeur de i pour la 1 fois
j'affiche la valeur de i pour la 2 fois
j'affiche la valeur de i pour la 3 fois
j'affiche la valeur de i pour la 4 fois
j'affiche la valeur de i pour la 5 fois

Variables locales

- Nous appelons une variable locale toute variable déclarée au sein d'une fonction.
- La portée d'une variable locale est limitée à la fonction.

```
int M ;
main()
{
int N ;
....
}
fct1 ()
{
int N ;
int M ;
}
```

La variable N de main n'a aucun rapport avec la variable N de fct1. De même, la variable M de fct1 n'a aucun rapport avec la variable globale M.

Variables statiques

- Les variables locales ne sont pas définies de manière permanente comme ceux des variables globales. Un nouvel espace mémoire leur est alloué à chaque entrée dans la fonction et libéré à chaque sortie.
- En C, il est possible d'allouer un espace mémoire permanant à des variables locales en utilisant le mot-clé **static**.
- Les variables locales statiques sont, par défaut, initialisées à zéro.

```c
#include <stdio.h>
main()
{ void fct(void) ;
int n ;
for ( n=1 ; n<=5 ; n++)
fct() ;
}
void fct(void)
{ static int i ;
i++ ;
printf ("appel numéro : %d\n", i) ;
}
```

Le programme affiche

```
appel numéro : 1
appel numéro : 2
appel numéro : 3
appel numéro : 4
appel numéro : 5
```

Les fonctions récursives

- Le langage C autorise la récursivité des appels de fonctions.
- Pour la récursivité directe : une fonction comporte, dans sa définition, au moins un appel à elle-même,
- L'exemple suivant utilise la récursivité pour le calcul du factoriel :

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
        int fac (int n);
        int n ;
        n = fac(4);
        printf("%d",n);
}
int fac (int n)
{
if (n>1) return (fac(n-1)*n) ;
else return(1) ;
}
```

Table des matières

Utilisation

- Tous les programmeurs C ont accès à un ensemble de fonctions standards et préprogrammées.
- Ces fonctions préprogrammées sont stockées dans une bibliothèque.
- Avant d'utiliser ces fonctions, vous devez savoir:
  1. Le nom de chaque fonction disponible
  2. Les arguments requis par chaque fonction
  3. Le type de données du résultat (le cas échéant) renvoyé par chaque fonction
  4. Une description de ce que fait chaque fonction
- Par exemple la fonction **sqrt**:

```
double sqrt(double N)
```

son en-tête liste qu'elle attend un argument double et renvoie une valeur double.

- L'utilisation de ces bibliothèques nécessitent une déclaration pour un fonctionnement correct. Elle se fait avec l'inclusion du fichier entête (header .h).

```
#include <math.h>
```

Les arguments de type tableaux

- Nous pouvons passer un argument tableau à une fonction. Regardons l'exemple suivant :

```
#include <stdio.h>
main( )
{
int Tab[5] = {2, 18, 1, 27, 16};
int val_max(int [5]); /* Déclaration de la function */
printf("Le maximum est %d", val_max(Tab));
}
int val_max(int vals[5])
{
int i, max = vals[0];
for (i = 1; i <= 4; ++i)
{
    if (max < vals[i])
    {
        max = vals[i];
        return (max);
    }
}
}
```

Les arguments de type tableaux

- Notez que le prototype de la fonction **val_max()** dans le **main()** retournera un entier et attend un tableau de cinq entiers comme un argument.

- Il est également important de savoir qu'un seul tableau est créé, ce tableau est appelé **Tab**, et dans **val_max()** le tableau (array) est connu sous le nom de **vals**. Comme illustré dans le programme, les deux noms font référence au même tableau.

- La fonction **val_max()** a besoin de savoir que l'argument **vals** fait référence à un tableau d'entiers.

Table des matières

Les arguments variables en nombre

- En C, nous pouvons réaliser une fonction capable de recevoir un nombre d'arguments susceptible de varier d'un appel à un autre. Les fonctions **printf** et **scanf** sont un bon exemple de fonction où le nombre d'argument change à chaque appel.

- Les fonctions **va_start** et **va_arg** du fichier entête **stdarg.h** permettent de définir des fonctions avec un nombre variable d'arguments.

- La seule contrainte à respecter est que la fonction doit posséder certains arguments fixes (c'est-à-dire toujours présents).

- Le dernier argument fixe permet d'initialiser le parcours de la liste d'arguments.

Exemple : fonction avec un nombre variable d'arguments

```c
#include <stdio.h>
#include <stdarg.h>
void essai (int par1, char par2, ...)
{
va_list adpar ;
int parv ;
printf ("premier paramètre : %d\n", par1) ;
printf ("second paramètre : %c\n", par2) ;
va_start (adpar, par2) ;
while ((parv = va_arg (adpar, int) ) != -1)
printf ("argument variable : %d\n", parv) ;
}
main()
{
printf ("premier essai\n") ;
essai (125, 'a', 15, 30, 40, -1) ;
printf ("\ndeuxième essai\n") ;
essai (6264, 'S', -1) ;
}
```

Le programme affiche

premier essai
premier paramètre : 125
second paramètre : a
argument variable : 15
argument variable : 30
argument variable : 40
deuxième essai
premier paramètre : 6264
second paramètre : S

Exemple : fonction avec un nombre variable d'arguments

- La fonction **essai** accepte deux paramètres fixes **par1** et **par2**, les trois points servent à spécifier au compilateur l'existence de paramètres en nombre variable.
- **va_list adpar** précise que **adpar** nous servira à récupérer les différents arguments variables.
- La fonction **va_start** permet l'initialisation de **adpar** avec l'adresse du paramètre variable. Il faut noter que cette dernière est déterminée par **va_start** à partir de la connaissance du nom du dernier paramètre fixe (c.à.d **par2**).
- La fonction **va_arg** fournit comme résultat la valeur trouvée à l'adresse courante fournie par **adpar** (son premier argument), suivant le type indiqué par son second argument (ici int).
- Elle incrémente aussi l'adresse contenue dans **adpar**, de manière que celle-ci pointe alors sur l'argument variable suivant.
- La voucle **while** nous permet de récupérer les différents arguments variables, sachant que le dernier a pour valeur -1.

Références