Les fonctions à nombre variable d?arguments

Merci 0

Nous avons souvent utilisé les fonctions type printf(), scanf(),...

Ces fonctions sont particulières, elles acceptent un nombre de paramètres variable ! Pouvons nous aussi créer de telles fonctions ? Présentation d'une bibliothéque du langage C "stdarg.h". Explication avec des exemples et des exercices.

LES FONCTIONS À NOMBRE VARIABLE D'ARGUMENTS

Le langage C permet de définir des fonctions dont le nombre d'arguments n'est pas fixé et peut varier d'un appel à un autre. Nous avons souvent utilisé les fonctions suivantes: - printf() - scanf() Ces fonctions sont particulières: elles acceptent un nombre de paramètres variable!

```
2  printf(" Bonjour !\n");
3
4  /* 2 paramètres */
5  printf(" le cube de %ld est :\n", valeur);
6
7
8  /* 4 paramètres */
9  printf("%ld * %ld = %ld\n", i, valeur, produit);
10
```

Pouvons nous aussi créer de telles fonctions ?

DÉCLARATION ET SYNTAXE

La déclaration se fait de la manière suivante:

```
#include <stdarg.h>
type fonction(type1 arg1, type2 arg2, ...)
{
}
```

Appel à la fonction :

```
fonction(arg1,arg2);
```

Règles à respecter Règle 1: Si le prototype contient P paramètres formels, alors l'appel à la fonction doit se faire avec au moins P paramètres. Exemple : On considère la déclaration suivante :

```
type fonction (type1 arg1, type2 arg2 ,type3 arg3, ...);
```

Lors de l'appel à cette fonction on doit fournir au moins 3 arguments :

```
fonction( arg1,arg2,arg3) //appel correct
fonction( arg1,arg2) //appel incorrect
fonction( arg1,arg2,arg3,arg4) //appel correct
```

Règle 2: Une fonction avec un nombre variable de paramètres doit comporter au moins un paramètre fixe. Exemple:

```
int somme(int a,...); // déclaration correcte

```

Règle 3: La notation ... (obligatoirement à la fin de la liste des paramètres) spécifie que la fonction possède un nombre variable de paramètres. Cette liste variable de paramètres doit toujours figurer en dernière position parmi les paramètres formels. Exemple:

```
void erreur(int n, char *msg, ...);
```

Cette déclaration dit que la fonction erreur() est définie de telle manière à ce que les appels doivent fournir au moins deux arguments, un de type int et un de type char*, mais qu'ils peuvent fournir des arguments supplémentaires. Exemple d'appel :

```
erreur( 3, "appel correct");
```

ACCÈS AUX ARGUMENTS

Pour accéder aux arguments situés après le dernier argument fixe, il faut utiliser certaines fonctions (ou plutôt macros) du fichier **stdarg.h** Cette bibliothèque contient toutes les fonctions dont on a besoin pour accéder aux arguments. Elle définit un type à utiliser pour traiter les listes variables de paramètres, ***va_list*** tableau contenant les informations sur les arguments ainsi que 3 macros pour récupérer la valeur des paramètres.

```
void va_start(va_list ap, last);
// fait pointer ap sur le premier argument variable fourni à la fonction.

type va_arg(va_list ap, type);
// renvoie le premier argument variable et fait pointer ap sur l'argument suivant.
// type est le type de l'argument qui va être lu et Va_arg génère une expression de ce void

va_end (va_list ap);
// remet tout en normal avant le retour &#224; la fonction appelante.
```

Ordre d'appel va_start doit être appliquée avant va_arg ou va_end. va_end doit être appelée une fois que va_arg a lu tous les arguments, sinon le comportement du programme sera imprévisible. Progression dans la liste d'arguments Le premier appel à va_arg fait renvoyer le premier argument de la liste Chaque appel suivant à va_arg permet de récupérer un argument. Valeurs renvoyées: - va_start et va_end ne renvoient pas de valeur. - va_arg renvoie l'argument courant de la liste (celui sur lequel ap pointe). Remarques Les types char, unsigned char ou flottants sont interdits avec va_arg. Règles de passage de paramètres:

- Pour les paramètres de type entier

- char, short sont convertis en int. - long reste au format long.

- Pour les paramètres de type réel

- Ils sont convertis en double ==Application=== Dans cette partie on va appliquer tous ce qu'on vu dans les précédentes : Je vais vous montrer un petit exemple d'application puis un petit exercice avec solution. On va réaliser une fonction qui calcule la somme des tout ses arguments, le nombre d'arguments est variable d'un appel à un autre. Comme je l'ai déjà dit : "Une fonction avec un nombre variable de paramètres doit comporter

au moins un paramètre fixe". Puisque les paramètres sont de même type, la déclaration de notre fonction somme peut être :
- double somme(double a,...) ; - int somme(int a,...) ; Dans la définition, on va utiliser tout les macros (fonction) que l'on a vues. - va_list : pour créer le tableau contenant les informations sur les arguments. - va_arg : pour récupérer l'argument courant et fait passer le pointeur (va_list) au suivant. - va_end : remet tout en normal avant le retour à la fonction appelante. Un problème :
Le nombre de paramètres est variable donc on ne sait pas combien de fois on doit utiliser la fonction va_arg ();
Solution : 1-Soit on va ajouter un paramètre indiquant le nombre de paramètre. 2-Soit on choisi un marqueur de fin pour qu'on puisse arrêter notre boucle (exemple : 0,-1). **Pour la 1er solution**

```
#include<stdio.h>
#include <stdarg.h>

int somme(int a,...)//la déclaration avec un paramètre (le nombre des paramètres passés à la
{
  int som=0,i=0,j=a;
  va_list ap; //création du pointeur
  va_start(ap,a); //initialisation sur le premier paramètre
  int c=0;
  while(i<j) //boucle pour récupérer tout les paramètres
  {
    c=va_arg(ap,int); //va-arg permet de retourner la valeur du paramètre courant et fait po
    som+=c;
    i++;
  }
  va_end(ap);
  return som;
}

int main()
{
  printf("la somme est : %d \n",somme(4,2,3,4,6));
  return 0 ;
  system("pause");
}
```

Résultat du programme

```
la somme est : 15
Press ENTER to continue
```

Et maintenant pour la 2ème solution
Comme marqueur de fin, je vais choisir 0 puisque x+0=x;

```c
#include <stdio.h>
#include <stdarg.h>

int somme(int a,...)
{
  int som=a,i=0;
  va_list ap;
  va_start(ap,a);
  int c=1;
  while(c!=0)
  {
    c=va_arg(ap,int);
    som+=c;
    i++;
  }
  va_end(ap);
  return som;
}

int main()
{
  printf("la somme est : %d \n",somme(1,2,3,4,6,7,8,9,0));
  system("pause");
}
```

Résultat du programme

```
la somme est : 40
Press ENTER to continue
```

Le moment de l'exercice est venu.

EXERCICE

Faire une simulation de la fonction printf(); Pour ne pas compliquer les choses,on ne traite que les entiers, les caractères, et les chaînes de caractères; Donc notre fonction doit traiter le cas : *ma_printf("je suis %s,j'ai %d",nom,age);* Aide
On peut utiliser la fonction "fputc()" ou autre fonction pour afficher les caractères et "itoa()" (utilisable seulement sous Windows) pour la conversion d'un entier en chaîne de caractères. Si vous êtes sous Linux ou autre OS vous pouvez utiliser la fonction suivante au lieu de "itoa()"cette fonction permet de convertir un entier en chaîne de caractères,on peut aussi choisir la base de conversion(base :2,8,10,16):

```
1
2  char* convertion(const int x,const unsigned short base, char* resultat)
3  {
4    char HEX[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
5
6    /* espace de travail */
7    int i, j, cpt, reste; /* reste est compris entre 0 et 16*/
8    char chaine[34]; /*en base 2 le plus long int se représente sur 32 octets + 1 octet final
9
10   int quotient = x;
11
12   /* vérification de la base. il faut que ça cadre avec "HEX" */
13   if ((base<2)||(16<base))
14   {
15     printf("base non valide \n");
16     /* ici il faudrait presque mettre un exit... ce qui est un peut violent*/
17     return NULL;
18   }
19
20   /* parce qu'on ne travaille qu'avec des entiers positifs */
21   if (quotient<0)
22   {
23      quotient = -quotient;
24   }
25
26   /*initialisations*/
27   cpt = 0;
28
29   /* calculs */
30   while (quotient!=0)
31   {
```

Vous avez maintenant tous les moyens pour simuler la fonction "printf()"

CORRECTION

Je vais vous fournir plusieurs solutions. La 1ère en utilisant "itoa()"

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>

int ma_printf(char *p,...)
{
int i=0,j=0;
va_list ap;//création de la liste
va_start(ap,p);//intialisation de la liste ap
char v[30];
int n;

while(*(p+i)!='\0')//
{
switch(*(p+i))//
{
case '%':
{ i++;
if(*(p+i)=='c')
{
fputc(va_arg(ap,int),stdout);
}

if(*(p+i)=='d')
{ n=va_arg(ap,int);
itoa(n,v,10);

for(j=0;j<strlen(v);j++){
fputc(v[j],stdout);
}
```

La 2ème en utilisant la fonction que j'ai donné ("conversion()")

```
#include <stdio.h>

#include <stdarg.h>

#include <string.h>

#include <stdlib.h>

char* convertion(const int x,const unsigned short base, char* resultat)

{

char HEX[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F



/* espace de travail */

int i, j, cpt, reste; /* reste est compris entre 0 et 16*/

char chaine[34]; /*en base 2 le plus long int se représente sur 32 octets + 1 octet final +

int quotient = x;



/* vérification de la base. il faut que ça cadre avec "HEX" */

if ((base<2)||(16<base)){

printf("base non valide \n");
```

Résultat des deux programmes

```
--->Je m'appelle Leo, j'ai 21 ans
Appuyez sur une touche pour continuer...
```

J'espère que ce tutoriel vous aura plu, j'ai essayé de faire aussi clair que possible. J'espère qu'il vous a aussi donné des idées pour de futures applications.