Fonction à ellipse

1. Fonction à ellipse

1.1. Description

- C'est une fonction ayant un nombre variable d'arguments dont les types ne sont pas connus par la fonction appelée. Cette fonction est disponible dans le langage C et par extension dans le langage C++.

- Ce nombre variable est symbolisé par 3 points de suspension consécutifs (…)

```
void test(int i, double j, ...);
```

- Lors de l'appel de la fonction « test », nous devons fournir au moins deux arguments : le premier du type « int » et le second du type « double ».

- Par exemple, les appels suivants vont générer des erreurs à la compilation :

```
test(); // Le 1er et le 2d argument sont manquants
test(5); // Le 2d argument est manquant
```

- Par contre les appels suivants, sont syntaxiquement corrects :

```
test(5,10.5); // OK, car au moins 2 arguments
test(2,7.3, "toto"); // OK les 2 arguments attendus + 1 sup.
```

- La fonction doit avoir obligatoirement au moins un argument « classique » avant les « … ».

```
test(int d, …); // OK, car au moins 1 argument classique
test(…); // Erreur aucun argument classique avant les « … »
```

1.2. Extraire la liste des arguments

- Des macros et un type ont été définis pour permettre de récupérer les arguments « anonymes » de la liste de paramètres variables.

- Ces macros sont : « va_start », « va_arg » et « va_end » ; le type est « va_list ».

- Pour pouvoir utiliser ces macros, il faudra inclure dans votre programme le fichier de définition « cstdarg » (l'équivalent de « stdarg.h » dans le langage C »).

- Le processus d'extraction suit les étapes ci-dessous :

  - On commence par déclarer une variable « nom_var » du type « va_list ». Cette variable sert à identifier le prochain paramètre à extraire de la liste.

  - On initialise par la suite une variable donnée par exemple « nom_var » en utilisant la macro « va_start » comme suit :

    - « va_start(nom_var,dernier_arg) » ou « nom_var » est la variable précédemment définie et « dernier_arg » est le dernier argument classique de la fonction.

  - Après l'initialisation de « nom_var », on peut récupérer maintenant un à un les arguments anonymes de la liste en utilisant pour cela la macro « va_arg » comme suit :

    - « va_arg(nom_var,type) » ; cette macro va renvoyer le premier argument variable et fait pointer « nom_var » sur l'argument suivant. « type » est le type de l'argument qui va être lu et « va_arg » va générer une expression de ce même type.

    - Ce processus suppose connaître préalablement le type des arguments à extraire.

    - On peut faire encore appel à « va_arg » autant de fois que l'on veut.

- Quand l'extraction prend fin, il faudra faire appel à la macro « va_end » pour remettre les choses dans l'état normal avant le retour à la fonction appelante. Cette macro va détruire la variable « nom_var » en procédant comme suit : « va_end(nom_var) ».

- Parmi les critères d'arrêt de la lecture des arguments anonymes, nous citons :

  - Un paramètre formel peut être utilisé pour indiquer le nombre d'arguments anonymes. C'est la solution la plus recommandée.
  -

```
double moyenne(int nbPara, ...){
  double total = 0;
  int j;
  va_list ap;

  // Il faudra ajouter un test sur la valeur nbPara.
  // Il ne faut que cette valeur soit égale à 0

  va_start(ap, nbPara);

  /* boucler sur tous les paramètres (sauf le 1er) */
  for (j = 1; j <= nbPara; j++){
    /* ajoute le prochain paramètre */
    total += va_arg(ap, double);
  }
  va_end(ap);
  return total / nbPara;
}
```

- Un argument anonyme a une valeur préfixée et le critère d'arrêt consiste à tester chacun des arguments anonymes lus contre cette valeur.
- Boucler jusqu'à trouver un pointeur nul.
- Le premier argument classique doit être structuré de telle façon que la fonction appelée doit pouvoir compter le nombre d'arguments anonymes. Généralement, cet argument est du type « char* ».

```
#include <cstdarg>
#include <cstdio>

/* minprintf: un printf minimal avec
   une liste variable d'arguments Kernighan&Ritchie*/

void minprintf(const char *fmt, ...){

    /* pointe à tour de rôle chacun des arguments anonymes */

    va_list ap;

    const char *p, *sval;
    int ival;
    double dval;

    /* « ap » pointe le 1er argument anonyme */

    va_start(ap, fmt);
```

```
    for (p = fmt; *p; p++) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
        switch (*++p) {
            case 'd':
                ival = va_arg(ap, int);
                printf("%d", ival);
                break;
            case 'f':
                dval = va_arg(ap, double);
                printf("%f", dval);
                break;
            case 's':
                for (sval = va_arg(ap, char *); *sval; sval++)
                    putchar(*sval);
                break;
            default:
                putchar(*p);
                break;
        }
    }

    va_end(ap); /* ménage quand c'est fini */
}
```

```
int main() {
    minprintf("int %d\n",5);

    minprintf("String %s et int %d\n","Bonjour",5);

    minprintf("Une serie d'entiers: %d, %d, \
                        %d, %d, %d\n",1,2,3,4,5);

    return 0;
}
```

Affichage en sortie

int 5
String Bonjour et int 5
Une serie d'entiers : 1, 2, 3, 4, 5