Arguments en nombre variable

Plan

1. Principe
2. Premier exemple
3. Autres exemples

1. Principe

Il est possible d'écrire des fonctions ayant un nombre variable d'arguments. Avec des fonctions `concatener` ou `max`, on aimerait pouvoir écrire

```
concatener("bonjour", "monde !");
concatener("<<", s, ",", t, ">>");
max(2,3,5);
max(8,4,1,6,9);
```

On peut le faire, sous réserve d'indiquer en plus le nombre d'arguments ou la fin de la liste d'arguments. Ainsi

```
concatener("bonjour", "monde !", NULL);
concatener("<<", s, ",", t, ">>", NULL);
max(3, 2,3,5);    // 3 args
max(5, 8,4,1,6,9);// 5 args
```

Déclaration

Les premiers arguments sont toujours fixes (au moins un), les autres sont déclarés par *ellipse* `...`.

```
char* concatener(char *,...);
int max(int nargs,...);
```

Dans la *définition*, on utilise quatre constructions contenues dans le fichier `<stdarg.h>` :

- `va_list` déclare un point pour parcourir les arguments;
- `va_start` initialise le parcours sur un premier paramètre;
- `va_arg` donne le paramètre suivant;
- `va_end` termine le parcours.

2. Premier exemple

```
#include <stdarg.h>

int somme(int nargs,...) {
  va_list args;                  //declaration
  int i, total = 0;

  va_start(args, nargs);         //initialisation

  for (i = 0; i < nargs; i++)
    total += va_arg(args, int);  //recuperation, progression

  va_end(args);                  //nettoyage
  return total;
}

main()
{
  printf("%d\n", somme(7, 1,2,3,4,5,6,7));  //=28
  printf("%d\n", somme(3, 1,2,3));          //=6
}
```

Commentaires

- `va_list args;` déclare une variable `args` de type pointeur générique `void*`.;
- `va_start(args, nargs);` initialise `args` avec l'adresse de l'argument suivant `nargs` : dans notre cas `args = &nargs + 1;`;
- `va_arg(args, int);` retourne la valeur de l'argument courant, et fait progresser le pointeur vers l'adresse de l'argument suivant;
- `va_end(args);` fait du "nettoyage".

3. Autres exemples

La fonction `max()`; ici aussi, le premier arguments est le nombre de nombres à suivre.

```
#include <stdio.h>
#include <limits.h>   /* pour INT_MIN */
#include <stdarg.h>

int max(int nargs,...) {
  va_list args;
  int i, maxloc = INT_MIN, x;

  va_start(args, nargs);
  for (i = 0; i < nargs; i++)
    if  ((x =va_arg(args, int)) > maxloc)
      maxloc = x;
  va_end(args);
  return maxloc;
}

main()
{
  printf("%d\n", max(7, 1,2,3,4,5,6,4));
  printf("%d\n", max(3, -1,-2,-3));
  printf("%d\n", max(0));
}
```

Résultats:

```
6
-1
-32768
```

La fonction `char* concat(char* mot,...)` concatène ses arguments, et retourne le résultat. Le dernier argument est `NULL`. Par exemple,

```
concat("Adieu","Monde", "Cruel.",NULL)
```

retourne

```
AdieuMondeCruel.
```

La fonction parcourt deux fois la liste des arguments:

- la première fois, elle totalise la longueur des chaînes données en argument;
- après avoir alloué la mémoire nécessaires, la fonctions recopie, dans un deuxième parcours, es arguments dans la chaîne résultat.

Première partie

```
char* concat(char* mot,...) {
  va_list chaines;

  char *x, *ou, *res;
  int longueur;

  longueur = strlen(mot);
  va_start(chaines, mot);

  while (x = va_arg(chaines, char*))  //dernier argument NULL
    longueur += strlen(x);

  va_end(chaines);
```

Deuxième partie

```
  // Allocation
  ou = res = (char *) malloc(1+longueur);

  strcpy(ou, mot); ou += strlen(mot);
  va_start(chaines, mot);

  while (x = va_arg(chaines, char*)) {
    strcpy(ou, x);
    ou += strlen(x);
  }
  va_end(chaines);

  return res;
}
```