Polytech Nice Sophia
SI3 / 2021-2022
Erick Gallesio

Programmation Procédurale

Corrigé TD 5

Fonctions à arité variable

1 Moyenne

La fonction que l'on doit écrire ici a comme prototype:

```
float moyenne(int count, ...);
```

Dans cet exercice, nous savons donc exactement le nombre de flottants qui on été passés à notre fonction. Une première version de notre fonction pourrait donc être:

```
float moyenne(int count, ...) {
  va_list ap;
  float sum = 0;

  va_start(ap, count);                // initiialiser "ap" après "count"
  for (int i = 0; i < count; i++) {
    sum += va_arg(ap, float);         // ajouter le ieme flottant dans "sum"
  }
  va_end(ap);

  return sum / count;                 // renvoyer la moyene
}
```

En fait **cette version ne marche pas** (elle provoque même l'arrêt brutal du programme). Toutefois, le compilateur nous aide ici avec un *warning* indiquant que les nombres flottants sont convertis en double lorsqu'ils sont passés en paramètre à une fonction (on dit qu'ils sont promus en doubles). Par conséquent, il faudra indiquer que l'on va chercher un `double` dans la pile (putôt qu'un `float`) puisque c'est ce qu'a mis le compilateur dans la pile lors de l'appel.

Par ailleurs, il faut que nous fassions attention quand `count` est égal à 0, sous peine de provoquer une division par 0 lors du calcul de la valeur qui suit le `return` ! Le code correct de la fonction est donc:

```
float moyenne(int count, ...) {
  va_list ap;
  float sum = 0;

  if (!count) return 0;           // Cas particulier où il n'y a pas de valeur

  va_start(ap, count);
  for (int i = 0; i < count; i++) {
    sum += (float) va_arg(ap, double);
  }
  va_end(ap);
  return sum / count;
}
```

2 La fonction cat_strings

Cet exercice est très proche du code vu en cours pour calculer le maximum d'une série d'entiers positifs.

Pour la fonction `cat_strings`, les objets que l'on doit aller chercher dans la pile sont des chaînes de caractères.

Pour aller chercher une chaîne, il faut faire `va_arg(ap, char*)`. Malheureusement, la notation `va_arg(ap, char[])` ne marche pas ici (on verra pourquoi quand on aura vu les pointeurs).

A part cela, pas de difficulté et la fonction peut s'écrire de la façon suivante:

```
void cat_strings (char str[], ...) {
  va_list ap;
  va_start(ap, str);

  for (; str != NULL; str = va_arg(ap, char *))
    printf("%s", str);

  va_end(ap);
}
```

3 Calculatrice

Pour travailler on initialise ici le résultat (`res`) à la valeur contenue dans le premier opérande. On place ensuite l'opérande courant dans la variable `courant` et on effectue le calcul.

```
    res = res θ courant   // où θ ∈ { '+', '-', '*', '/' }
```

Bien sûr, ce calcul s'effectue tant que courant est positif. Notre version de la fonction `evaluer` peut donc s'écrire de la façon suivante:

```
int evaluer(char op, int operande, ...) {
  va_list ap;
  int res = operande;      // initialisation du résultat avec le premier nombre

  va_start(ap, operande);  // initialisation de ap
  if (res < 0) return 0;   // si aucun opérande on renvoie 0 par convention

  for(int courant = va_arg(ap, int); courant >= 0; courant = va_arg(ap, int)) {
    switch (op) {
      case '+': res += courant; break;
      case '-': res -= courant; break;
      case '*': res *= courant; break;
      case '/': res /= courant; break;  // incorrect si courant == 0
    }
  }

  va_end(ap);
  return res;
}
```

Remarque:

Cette version de la fonction n'est pas correcte dans le cas où on fait une division par 0 (le programme se termine en erreur). Il faut donc tester ce cas explicitement.

On obtient donc la version corrigée suivante:

```c
int evaluer(char op, int operande, ...) {
  va_list ap;
  int res = operande;      // initialisation du résultat avec le premier nombre

  va_start(ap, operande);  // initialisation de ap
  if (res < 0) return 0;   // si aucun opérande on renvoie 0 par convention

  for(int courant = va_arg(ap, int); courant >= 0; courant = va_arg(ap, int)) {
    switch (op) {
      case '+': res += courant; break;
      case '-': res -= courant; break;
      case '*': res *= courant; break;
      case '/': if (courant != 0) {
                  res /= courant; break;
                } else {
                  printf("*** ERREUR: division par 0\n");
                  va_end(ap);
                  return 0;
                }
    }
  }
  va_end(ap);
  return res;
}
```

Code complet de la calculatrice

4 Printf

Le code de la fonction `Printf` consiste à afficher le format caractère par caractère. Deux cas sont possibles:

- le caractère courant n'est pas '%' ⟹ on l'affiche avec un `putchar`
- le caractère courant est le caractère '%' ⟹ on sait qu'il va falloir aller chercher un paramètre (c'est-à-dire appeler la macro `va_arg`). Il reste par contre à déterminer le type du paramètre à aller chercher. Ce type est déterminé par le caractère qui suit le '%' (chaîne si ce caractère est 's', entier si le caractère est 'd' ou 'x, …)

Le code de la fonction est donc:

```c
void Printf(char format[], ...) {
  va_list ap;

  va_start(ap, format);

  for (int i = 0; format[i]; i++) {
    if (format[i] != '%')
      putchar(format[i]);
    else {
      switch (format[++i]) {
        case 'd':                    // ---- Entier en décimal
          print_base(va_arg(ap, int), 10);
          break;
        case 'x':                    // ---- Entier en héxadécimal
          print_base(va_arg(ap, int), 16);
          break;
        case 'f': {                  // ---- Flottant
          char buffer[BUFF_SIZE];    // DOUBLE et pas FLOAT (lire le warning!)
          snprintf(buffer, BUFF_SIZE, "%f", va_arg(ap, double));
          print_string(buffer);
          break;
        }
        case 's':                    // ---- Chaîne de caractères
          print_string(va_arg(ap, char *));
          break;
        case 'c':                    // ---- Caractère
          putchar(va_arg(ap, int)); //  INT, pas CHAR (car promotion entière)
          break;
        case '%':                    // ---- Un bête '%' à afficher...
          putchar('%');
          break;
        default:                     // ---- On a %X où X ∉ {%, s, c, x, d, f}.
          putchar('%');              //         afficher le '%'
          if (format[i]) {           //         et le car. qui le suit (si il existe)
            putchar(format[i]);
            break;
          } else {
            va_end(ap);              //         un '%' seul à la fin de la chaîne
            return;
          }
      }
    }
  }
  va_end(ap);
}
```

Remarques:

1. la fonction `print_base` est celle que l'on avait écrite précédemment pour imprimer un nombre dans une base quelconque.
2. la fonction `print_string` se résume à

   ```c
   void print_string(char str[]) {         // équivalent à fputs(current, stdout)
      for (int i =0; str[i]; i++)
        putchar(str[i]);
    }
   ```

3. Pour `%c` et `%f`, les type utilisés dans `va_arg` sont respectivement `int` et `double`. Cela est du au fait que le compilateur C étend les caractères et les flottants en des entiers et des doubles quand il les utilise dans une expression, ou les passe en paramètres. Comme les objets dans la pile d'exécution sont des objets étendus. Il faut que notre fonction aille les récupérer avec le type utilisé par le

compilateur. Si on ne le fait pas, les valeurs récupérées sont fausses (voire provoquent une erreur à l'exécution). Heureusement, le compilateur C signale ce problème assez clairement (si touefois vous lisez les *warnings* … 😉).

4. Noter que l'on peut avoir:

   - un '%' non suivi d'une des lettres prévues (`d` , `x` , `c` , `s` , `%`). Dans ce cas, on affichera ces deux caractères;
   - un `%` tout à fait à la fin de la chaîne. Dans ce cas, on affichera simplement `%`

   Ces deux cas sont traités dans le `default` de notre `switch` .

Code complet de Printf