TD8: Nombre variable d'arguments et tables de hachage

Programmation en C (LC4)

Semaine du 17 Mars 2008

1 Fonctions à nombre variable d'arguments

Exercice 1 Écrire une fonction `moyenne` à nombre variable d'arguments prenant un premier argument fixe de type entier indiquant le nombre d'arguments qui suivent, supposés de type `double`, et renvoyant leur moyenne arithmétique. On rappelle que la moyenne arithmétique d'un ensemble de nombres $(x_i)_{1 \leqslant i \leqslant n}$ est donnée par :

$$\frac{1}{n} \sum_{i=1}^{n} x_i.$$

► **Correction**

```
double moyenne_arithmetique(int n, ...){
  va_list ap;
  double somme = 0;
  int i;

  va_start(ap, n);
  for(i=0; i<n; i++){
    somme += va_arg(ap, double);
  }
  va_end(ap);

  return somme / n;
}
```

Exercice 2 Écrire une fonction `concatenation` qui prend en argument fixe un caractère `sep` suivi de chaînes de caractères puis d'un pointeur `NULL` et affiche les chaînes dans l'ordre, séparées par le caractère `sep`.

► **Correction**

```
void concatenation(char sep, ...){
  va_list ap;
  int i;
  char *s;

  va_start(ap, sep);
  while((s=va_arg(ap, char *))){
    printf("%s%c", s, sep);
  }
  printf("\n");
  va_end(ap);
}
```

Exercice 3 Écrire une fonction `monprintf` prenant en argument une chaine de caractère pouvant contenir un certain nombre de fois le motif `%i` et affichant sur la sortie standard la chaine où les différentes occurrences de `%i` sont remplacé par les arguments suivants de l'appel à `monprintf`. Vous pouvez utiliser la fonction `printf` pour cet exercice.

▶ **Correction**

```
void monprintf(char *format, ...) {
  va_list ap;

  va_start(ap, format);

  while(*format != '\0') {
    if (*format == '%')
      if (*(++format) == 'i') {
        printf("%i", va_arg(ap, int));
      } else
        putchar(*format);
    else
      putchar(*format);
    format++;
  }

  va_end(ap);
}
```

2 Tables de hachage

Exercice 4 Écrire une fonction table_de_hachage_t cree_table_de_hachage(**int** taille) qui crée une table de hachage vide avec un tableau de la taille indiquée (mais qui ne contient que des listes vides puisqu'il n'y a pas encore d'élément dans la table de hachage).

▶ **Correction**

```
table_de_hachage_t cree_table_de_hachage(int taille) {
    int i;
    table_de_hachage_t table = malloc(sizeof(struct table_de_hachage_s));
    table->taille = taille;
    table->tableau = malloc(table->taille * sizeof(liste_t));
    for (i = 0; i < table->taille; i++)
        table->tableau[i] = NULL;
    return table;
}
```

Exercice 5 Écrire une fonction **void** detruit_table_de_hachage(table_de_hachage_t table) qui libère la mémoire occupée par une table de hachage donnée en argument.

▶ **Correction**

```
void detruit_table_de_hachage(table_de_hachage_t table) {
    int i;
    liste_t liste;
    for (i = 0; i < table->taille; i++)
        while (table->tableau[i] != NULL) {
            liste = table->tableau[i]->suivant;
            free(table->tableau[i]);
            table->tableau[i] = liste;
```

```
        }
    free(table->tableau);
    free(table);
}
```

Exercice 6 Écrire une fonction **int** hachage(table_de_hachage_t table, **char** *cle) qui calcule le haché d'une clé, c'est-à-dire l'indice du tableau de listes de la table de hachage que l'on doit utiliser pour placer les valeurs associées à la clé donnée en argument : pour cet exercice, le haché sera simplement la somme des valeurs associées aux caractères de la clé modulo la taille du tableau.

▶ **Correction**

```
int hachage(table_de_hachage_t table, char *cle) {
    int hache = 0;
    for (; *cle; cle++)
        hache += *cle;
    return (hache % table->taille);
}
```

Exercice 7 Écrire une fonction **void** insere(table_de_hachage_t table, **char** *cle, **int** valeur) qui insère dans la table de hachage la valeur associée à la clé donnée en argument.

▶ **Correction**

```
void insere(table_de_hachage_t table, char *cle, int valeur) {
    int hache;
    liste_t liste = malloc(sizeof(struct liste_s));
    hache = hachage(table, cle);
    liste->suivant = table->tableau[hache];
    liste->cle = cle;
    liste->valeur = valeur;
    table->tableau[hache] = liste;
}
```

Exercice 8 Écrire une fonction **int** recherche(table_de_hachage_t table, **char** *cle, **int** *valeur) qui cherche si une entrée de la table a pour clé celle indiquée en argument, et le cas échéant renvoie la valeur correspondante via le pointeur donné en argument. Cette fonction renvoie un entier qui indique si la recherche a abouti.

▶ **Correction**

```
int recherche(table_de_hachage_t table, char *cle, int *valeur) {
    liste_t liste = table->tableau[hachage(table, cle)];
    for (; liste; liste = liste->suivant)
        if (strcmp(cle, liste->cle) == 0) {
            *valeur = liste->valeur;
            return 1;
        }
    return 0;
}
```

Exercice 9 Écrire une fonction **void** supprime(table_de_hachage_t table, **char** *cle) qui efface l'éventuelle entrée de la table ayant pour clé celle donnée en argument (et on suppose qu'il y a au plus une seul telle entrée dans la table).

▶ **Correction**

```
void supprime(table_de_hachage_t table, char *cle) {
    int hache;
    liste_t liste, *precedent;
    hache = hachage(table, cle);
    liste = table->tableau[hache];
    precedent = &table->tableau[hache];
    for (; liste; precedent = &liste->suivant, liste = liste->suivant)
        if (strcmp(cle, liste->cle) == 0) {
            *precedent = liste->suivant;
            free(liste);
            return;
        }
}
```

Exercice 10 — *Bonus* Écrire une fonction **void** affiche_numero(**int** numero) qui affiche un numéro de téléphone donné sous la forme d'un entier : ainsi si numero = 564302120, la fonction devra afficher `05.64.30.21.20` (Pourquoi doit on omettre le 0 du préfixe téléphonique ?). Écrire dans la fonction main() une séquence d'opérations qui crée une table de hachage représentant un répertoire téléphonique, insère plusieurs entrées dans la table (noms et numéros), effectue une recherche puis supprime l'élément trouvé, fait à nouveau la même recherche et enfin détruit la table.

▶ Correction

```
void affiche_numero(int numero) {
    int n = 100000000, m = 0;
    printf("0");
    while (n) {
        printf("%d", (numero / n) % 10);
        m++;
        if (m % 2 && m != 9)
            printf(".");
        n /= 10;
    }
    printf("\n");
}
```