Mémento C

1. Mots-clés, identificateurs et noms des fichiers C

Les mots-clés et mots réservés du langage C (ne pouvant pas être utilisés comme identificateurs) sont :

```
auto    break   case     char   const     continue  default   do      double  else    enum    extern
float   for     goto     if     int       long      register  return  short   signed  sizeof  static
struct  switch  typedef  union  unsigned  void      volatile  while
```

Un identificateur (nom de variable, de type, de fonction, de procédure, de paquetage) est une chaîne de caractères composée de lettres, éventuellement de caractères *souligné* '_', de chiffres. Le premier caractère ne peut pas être un chiffre.

Attention : le langage C distingue les majuscules et les minuscules.

L'extension `.c` est utilisée pour le corps d'un paquetage ou d'un programme. Par convention, pour un paquetage `<nom>`, le corps du paquetage est placé dans le fichier `<nom>.c`, la spécification dans le fichier `<nom>.h`.

2. Opérations et priorités

Les opérations suivantes sont triées par ordre décroissant de priorité :

1. `f` *appel de fonction* | `t[...]` *accès tableau* | `a.x` *accès structure*
2. `-` *signe moins (unaire)* | `!` *négation booléenne* | `*` *indirection (pointeurs)*
3. `*` *multiplication* | `/` *division* | `%` *modulo*
4. `+` *addition* | `-` *soustraction*
5. `<=` *inférieur ou égal* | `>=` *supérieur ou égal* | `<` *inférieur strict* | `>` *supérieur strict*
6. `==` *égalité* | `!=` *différence*
7. `&&` *et*
8. `||` *ou*
9. `... ? ... : ...` *expression conditionnelle*

Dans une expression, les opérateurs de même priorité sont évalués de gauche à droite. Utiliser les parenthèses `( )` pour effectuer une opération en priorité ou en cas de doute sur une priorité.

Opérateurs booléens conditionnels

Les opérateurs booléens `&&` et `||` sont des *opérateurs conditionnels* : pour chacun le second opérande est évalué si et seulement si le premier a une certaine valeur.

exp1 `&&` *exp2*	*exp1* `||` *exp2*
exp2 est évaluée si et seulement si *exp1* est *vrai*	*exp2* est évaluée si et seulement si *exp1* est *faux*
Exemple : `(i<=10) && T[i]!=0`	Exemple : `(b==0) || (a/b > 1)`

3. Types de données

Types de base

int	entiers
char	caractères
float ou **double**	flottants

Types énumérés

Déclaration d'un type énuméré `Couleur`, comportant les valeurs `Rouge`, `Jaune`, `Vert` :

```
typedef enum {Rouge, Jaune, Vert} Couleur;
```

Tableaux

Déclaration d'un tableau `tab` de taille `N` d'éléments de type `Telem` :

```
Telem tab[N];
```

Les indices des cases du tableau `tab` sont des entiers sur l'intervalle $[0, \texttt{N} - 1]$. `tab[i]` est l'élément d'indice `i` dans le tableau.

Pointeurs

Déclaration d'un pointeur `p` sur une valeur de type `T` :

```
T * p;
```

Modification de la valeur du pointeur (fait pointer `p` sur la valeur de `x`) :

```
T x;
p = &x;
```

Accès ou modification de la valeur pointée par `p` :

```
*p = *p + 1; // x est incrémenté de 1
```

Allocation dynamique d'une valeur de type `T`, pointée par `p` :

```
p = (T *)malloc(sizeof(T));
```

Une variable de type tableau peut être considéré et utilisé comme un pointeur sur la première case du tableau.

Une chaîne de caractère (type **char** `*`) est un pointeur sur le premier caractère, et peut être considéré comme un tableau de caractères. Par convention, le dernier caractère de la chaîne est suivi du caractère spécial `'\0'` (de code ASCII 0).

Structures

Déclaration d'un type structure `couple` contenant les champs `x` (de type `T1` et `y` (de type T2) :

```
typedef struct {
   T1 x;
   T2 y;
} couple;
```

Déclaration d'une variable de type `couple` :

```
couple c;
```

Accès aux valeurs des champs : `c.x`, `c.y`.

On peut déclarer une structure récursive, en nommant la structure :

```c
// Type cellule pour liste chaînée d'entiers
typedef struct s_cellule {
   int element; // élément courant
   struct s_cellule * suivant; // pointeur vers la cellule suivante
} Cellule;
typedef Cellule * Liste; // Une liste est un pointeur vers la cellule de tête
```

4. Lecture et écriture de «fichiers texte» depuis un programme C

Dans la terminologie C un *fichier texte* est un fichier de caractères auquel on accède de manière séquentielle (dans l'ordre d'apparition dans le fichier). Des procédures et fonctions d'accès sont fournies dans le paquetage nommé `stdio`. Notons que les périphériques d'entrée-sortie «standard» (comme le clavier et l'écran) sont considérés comme des fichiers texte particuliers. Nous décrivons ci-dessous certaines procédures et fonctions d'accès à des fichiers texte.

Toute opération d'entrée/sortie sur un fichier consiste en :

1. l'ouverture du fichier,
2. des opérations de lecture/écriture dans le fichier,
3. la fermeture du fichier.

Différentes erreurs peuvent se produire lors de l'exécution des procédures et fonctions suivantes (par exemple un fichier externe inexistant ou impossible à créer, une valeur impossible à lire, ...).

Lorsque de telles erreurs se produisent, les fonctions retournent des valeurs particulières.

Ouverture de fichiers

Comme beaucoup de langages, C distingue :

— le *fichier externe*, stocké par exemple sur le disque dur de la machine, et que l'on connait par son **nom** (une chaîne de caractères) ;

— le *descripteur interne*, qui est une structure de données manipulée par le programme C et dont le type est `File *`.

Ouvrir un fichier depuis un programme C revient donc à associer un descripteur interne à un fichier externe. Cela se fait avec la fonction `fopen` :

```c
File *fopen(const char * name, const char * mode);
```

Cette procédure **ouvre** un fichier externe de nom `name` et renvoie le descripteur interne associé à ce fichier. La chaîne de caractère `mode` peut être :

— `"r"` pour un accès en lecture : il est alors possible d'y effectuer des opérations de lecture ;

— `"w"` pour un accès en écriture : il est alors possible d'y effectuer des opérations d'écriture. Si le fichier n'existe pas, il est alors créé.

Opérations de lecture / écriture

Opérations d'écritures

Les principales primitives d'écritures en C sont `printf` et `fprintf` :

```c
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

L'argument `stream` de la fonction `fprintf` désigne le descripteur du fichier (ouvert en écriture) dans lequel l'opération d'écriture est effectuée. La fonction `printf` écrit sur la sortie standard.

La chaîne de caractères `format` permet de donner le *format* de ce qui va être écrit : il s'agit d'une chaîne de caractère, comportant des *spécifications de conversions* (un `'%'`, indiquant comment interpréter les arguments fournis après le format, dans l'ordre d'apparition dans ce format.

L'instruction `printf("%s = %d", "un", 1);` écrit la chaîne `"un = 1"`. La spécification `%s` (chaîne de caractère) correspond à l'argument `"un"` ; `%d` (entier) correspond à `1`.

Les principaux types de spécifications sont : `%d` (écriture d'un entier), `%f` (flottant), `%c` (caractère), `%s` (chaîne de caractère).

Opérations de lecture

Les principales primitives d'écritures en C sont `scanf` et `fscanf` :

```
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

L'argument `stream` de la fonction `fscanf` désigne le descripteur du fichier (ouvert en lecture) dans lequel l'opération de lecture est effectuée. La fonction `scanf` écrit sur la sortie standard.

La chaîne de caractère `format` indique le format des données qui vont être lues (cf `printf`/`fprintf`). Les spécifications de conversions permettent d'affecter les valeurs lues à des variables, dont on donne la *référence* (ou l'adresse) :

```
int x;
// Lecture d'un entier, affectation de sa valeur àx
scanf("%d", &x);
```

La primitive `fgets` permet de lire une ligne (se terminant par le caractère spécial de retour à la ligne `'\n'`) dans un fichier, et placer cette ligne dans une chaîne de caractère donnée en argument :

```
char *fgets(char *s, int size, FILE *stream);
```

`stream` est le descripteur du fichier dans lequel la chaîne est lue; `s` est la chaîne de caractères à laquelle la ligne lue sera affectée; `size` est le nombre maximum de caractères placés dans `s` (caractère de fin de chaîne `'\0'` inclus : le nombre maximum de caractères réellement lus dans le fichier est donc `size` -1).

Fermeture de fichiers

Une fois les différentes opérations faites sur un descripteur interne, celui-ci doit être fermé :

```
int fclose(FILE *stream);
```

Cette procédure **ferme** le fichier externe associé au descripteur interne `stream`.