Recherche d'Information Multimédia

Le Langage C

Georges Quénot

Les types numériques

```
/* ceci est un commentaire */
int i,j;    /* entier par défaut (32 bits) */
long k,l;   /* entier long, 32 (ou 64) bits */
short m,n;  /* entier court, 16 bits */
char c;     /* caractère, 8 bits */
unsigned int ui;   /* entier non signé */
unsigned long ul;  /* entier 32 bits non signé */
unsigned short us; /* entier 16 bits non signé */
unsigned char uc;  /* caractère non signé */
float x,y;  /* nombre flottant 32 bits */
double z;   /* nombre flottant 64 bits */
```

Pointeurs et tableaux

```
int *pi;   /* pointeur sur un entier */
           /* ou sur une zone mémoire */
long *k;   /* pointeur sur un entier long */
           /* etc... */
int ti[4]; /* tableau de 4 entiers ti[0] à ti[3]*/
int k,l;
ti[1] = k;
l = *pi;   /* déréférencement de pointeur */
l = pi[0]; /* accès “style” tableau, identique */
k = ti[2]; /* accès aux éléments d’un tableau */
k = *(ti+2); /* accès “style” pointeur */
```

Pointeurs et tableaux, suite

```
int i,*pi,j;
pi = &i; /* prise de référence */
         /* pi prend la valeur “adresse de i” */
j = *pi; /* équivalent à j = i maintenant */

int n,*ti;
n = 100;
ti = (int *) malloc(n*sizeof(int)); /* allocation
            d’une zone mémoire pour un tableau */
ti[45] = 12; /* ... */
free(ti); /* libération de l’espace mémoire */

void *p;  /* pointeur générique */
```

Pointeurs et tableaux, suite

```
int **pi;   /* pointeur de pointeur */
int *qi;    /* pointeur simple */
int r;
qi = *pi;
r = qi[3];  /* r = (*pi)[3] = pi[0][3]; */

int n,*ti;
n = 100;
ti = NULL;  /* pointeur nul : invalide */
ti = (int *) realloc(ti,n*sizeof(int));
/* réallocation d'une zone mémoire */
ti = (int *) realloc(ti,2*n*sizeof(int));
```

Chaines de caractères

Une chaine de caractères est un tableau de caractères terminée par un 0 (zéro).

```
char c,*s,name[256]; int l;     /* s non alloué */
name[0] = name[1] = c = 'A';   /* aff. multiple */
name[2] = 0;    /* name contient la chaine "AA" */
s = strdup(name);       /* copie name dans s avec
  allocation d'une zone mémoire de 3 caractères */
strcat(name,s);         /* ajoute la chaine s à la
       chaine name (name vaut maintenant "AAAA")*/
strcpy(name,"toto\ntiti");/* copie d'une chaine */
l = strlen(name); /longueur d'une chaine, l = 9 */
```

Formats d’impression

Écriture :

```
int i = 25; float r = 2.336; char s[256]; FILE *fp;
printf("toto\n");        /* sortie standard */
fprintf(fp,"toto\n");    /* dans un fichier */
sprintf(s,"toto\n");     /* dans une chaîne */
printf("%d",i);          /* "25"   */
printf("%4d",i);         /* "  25" */
printf("%04d",i);        /* "0025" */
printf("%f",r);          /* "2.336000" */
printf("%6.2f",r);       /* "  2.34"   */
printf("%s",s);          /* "toto\n"   */
printf("%d %f %s",i,r,s);   /* "25 2.336000 toto\n" */
```

Formats d’impression

Lecture :

```
int i; float r; char s[32],line[256]; FILE *fp;
scanf("%d",&i);              /* entrée standard */
fscanf(fp,"%d %f",&i,&r);  /* à partir d’un fichier */
sscanf(line,"%s",s);         /* à partir d’une chaîne */
```

`fscanf()` avance automatiquement dans le fichier.

Les délimiteurs sont l’espace, la tabulation ou le retour charriot.

Un pointeur doit être passé pour la “remontée” de l’élément lu : passer l’adresse de la variable sauf pour les chaînes de caractères (qui sont déjà des pointeurs).

Fonctions (C ansi)

```
int f(int n, int p)
{
    n = n*n;
    p = p*p;
    return(n+p);     /* SEUL élément retourné */
}
...

int n,p,q;
n = 3; p = 4;
q = f(n,p);    /* q = 25, n et p NON modifiés */
               /* passage par VALEUR */
```

Fonctions (C ansi)

```
int fp(int *n, int *p)
{
    *n = (*n)*(*n);
    p[0] = p[0]*p[0];
    return((*n)+p[0]);
}
...
int n,p,q;
n = 3; p = 4;
q = fp(&n,&p);    /* q = 25, n et p MODIFIES */
                  /* n = 9 et p = 16 */
                  /* passage par RÉFÉRENCE */
```

Structures

```
typedef struct {
    char *name;
    char *val;
} entry;
...
  entry entries[10000];              /* entries = */
  ...        /* tableau de 10000 structures entry*/
  entries[0].name = strdup("NOM");
  entries[0].val = strdup("Dupont");
  entries[1].name = strdup("INFO");
  entries[1].val = strdup("TELEPHONE");
...
```

Organisation modulaire

- Fichiers `".h"` : déclarations (headers),
- Fichiers `".c"` : programmes, sous-programmes,
- Ensembles de fonctions/procédures :
  1 paire de fichiers `".h"` et `".c"`.
  Exemple : `"cgiu.c"` et `"cgiu.h"`.
- Programmes finaux : 1 fichier `".c"` contenant :
  - Une fonction `" main()"`,
  - Les fonctions procédures utilisées localement,
  - Les appels aux fichiers de déclarations `".h"` utiles.
- Fichiers `".a"` ou `".so"` : bibliothèques.

Fichiers de déclarations

cgiu.h :

```
typedef struct {
    char *name;
    char *val;
} entry;                                  /* structure */

int countword(char *line, char stop);
char *getword(char *line, char stop);
entry *get_entries(int *);                /* procédures */
```

#define, #include, main()

```
#define  MAX_ENTRIES  10000        /*constante */

#include <stdio.h>                 /* E/S, système */
#include <stdlib.h>   /* Bibliothèque, système */

#include "cgiu.h"                  /* nos fonctions */

int main(int argc, char *argv[]) {
  ...
  exit(0);        /* exit() au lieu de return() */
}
```

Compilation :

```
cc –o post2 post2.c cgiu.c –lm
```

- `cc` : compilateur C,
- `-o post2` : nom du programme exécutable,
- `post2.c cgiu.c` : fichiers programmes,
- `-lm` : lier à la bibliothèque mathématique,
- `post2.c cgiu.c` sont compilés séparément en `post2.o cgiu.o` puis sont liés entre eux, avec les bibliothèques systèmes, et avec les bibliothèques spécifiées pour produire le fichier exécutable `post2`.

Structures de contrôle, `cgiu.c`

```
char *getword(char *line, char stop)
{
    int x,y;
    char *word;
    for(x = 0; (line[x] != 0) && (line[x] != stop); x++);
                                        /* get word length */
    word = (char *) malloc((x+1)*sizeof(char));
                                    /* allocate word buffer */
    for (y = 0; y < x; y++) word[y] = line[y];
                                     /* copy word in buffer */
    word[x] = 0;                        /* terminate string */
    if (line[x]) x++;                   /* skip stop character */
    for (y = 0; line[x+y] != 0; y++) line[y] = line[x+y];
                                         /* shift line left */
    line[y] = 0;                        /* terminate string */
    return(word);
}
```

Structures de contrôle, `cgiu.c`

Boucle :

```
for(x = 0;((line[x]) && (line[x] != stop));x++)
```

```
for (e1; e2; e3) {/* corps */}
```

e1 : condition(s) initiales (séparées par ","),

e2 : condition d'arrêt,

e3 : opération(s) à exécuter en fin de corps,

Équivalent à :

```
e1; while (e2) {/* corps */ ; e3}
```

Expressions booléennes

Booléen = entier, vaut “vrai” si non nul, “faux” si nul.

`((line[x]) && (line[x] != stop))` équivaut à :

```
((line[x] != 0) && (line[x] != stop))
```

`&&` : opérateur AND,

`||` : opérateur OR,

`!` : opérateur NOT,

`==` : test d’égalité entre deux valeurs numériques,

`!=` : test de non égalité entre deux valeurs numériques,

<, >, <=, >=, etc…

Affectations multiples, op. ++ et - -

La valeur d'une expression est celle de son terme gauche :

```
int i,j,k,n;
k = (j = i + n) + n;      /* k = i +2*n */
```

++x (--x): équivalent à x = x+1 (x = x-1);

x++ (x--): même chose mais l'opération est effectuée après en cas d'affectation mutliple :

```
x = 3; a = 2+(x++);      /* a = 5  x = 4 aff. après */
x = 3; a = 2+(++x);      /* a = 6  x = 4 aff. avant */

bool = (line[y++] = line[x++]);
/* copie de line[x] dans line[y] puis dans bool,
   puis incrémentation de x et y */
```

Fonction countword()

```
#include "cgiu.h"

int countword(char *line, char stop)
{
    int count;
    if (!line[0]) return(0);
    /*  (!line[0]) équivaut à (line[0] == 0)  */
    for (count = 1; *line; line++)
    /*  *line équivaut à (line[0] != 0)  */
        count += (*line == stop);
        /*  (*line == stop) equiv. à (line[0] == stop)  */
    return(count);
}
```

Fonction getword()

```
char *getword(char *line, char stop)
{
  /* extrait et retourne un mot situé en début de    */
  /* ligne, supprime le mot de la ligne en la        */
  /* décalant vers la gauche.                        */
  int x,y;
  char *word;
  for(x = 0; (line[x] != 0) && (line[x] != stop);x++);
                       /* calcul de la longueur du mot */
                  /* "stop" est le caractère séparateur */
  word = (char *) malloc((x+1)*sizeof(char));
  /* allocation d'une zone mémoire pour le mot.      */
  /* la valeur du pointeur "word" est retournée.     */
```

Fonction getword()

```
char *getword(char *line, char stop)
{
  ...
  for (y = 0; y < x; y++) word[y] = line[y];
        /* copie du mot dans la zone mémoire allouée */
  word[x] = 0;        /* terminaison de la chaîne mot */
  if (line[x]) x++;
  /* on saute le caractère de séparation si on n'est */
  /* pas en fin de ligne */
  for (y = 0; line[x+y] != 0; y++) line[y] = line[x+y];
              /* décalage de la ligne vers la gauche */
  line[y] = 0;        /* terminaison de la chaîne mot */
  return(word);
}
```

Opérations sur les fichiers

- Fichiers : décrits par des pointeurs de type “`FILE *`”.
- Fichiers définis par défaut (toujours ouverts) :
  - `stdin` : entrée standard (typiquement le clavier),
  - `stdout` : sortie standard (typiquement un terminal),
  - `stderr` : sortie des msg. d’erreur (typ. un terminal),
- Ouvrir et fermer des fichiers : `fopen` et `fclose` (voir `man`),
- Lire et écrire des caractères : `getc` et `putc` (voir `man`),
- Lectures et écritures formatées : `fscanf` et `fprintf` (voir `man`),
- Voir aussi : `fread, fwrite, fputs, fgets, printf, scanf, putchar, getchar,...`

Bon à savoir...

- Il y a très souvent une fonction de la bibliothèque standard C qui fait ce que l'on veut (en particulier pour la manipulation des chaînes de caractères) : ne pas réinventer `strdup()`, `strcmp()`, `atoi()`, `atof()`, `etc...`
- Les fonctions sont documentées dans le `man`.
- Il y a de nombreuses façons de faire une chose donnée en C : choisir la plus simple.
- Attention aux `malloc()`, au caractère terminal des chaînes et aux indices des tableaux (qui vont de `0` à `n-1` et non de `1` à `n`).