Chaînes de caractères

Les prototypes des fonctions de manipulation de chaînes les plus courantes sont donnés en annexe, leur connaissance peut être utile pour vérifier le sens de ce que l'on écrit.

1 Préambule : compléments sur les caractères

Pour coder les caractères dans les octets de la machine on utilise une correspondance caractère - nombre entier nommée code ASCII. Sur les PC utilisés les caractères sont écrits sur un octet, on peut donc coder au maximum 256 caractères différents. Par exemple `A` (majuscule) correspond à l'entier 65 et est donc codé sous la forme de 65 écrit en binaire, `j` (minuscule) correspond à 106, { à 123, etc. Le C permet d'interpréter l'octet d'un caractère comme un caractère ou comme un entier simplement en utilisant un format caractère (`%c`) ou entier (`%d`) :

```
#include<stdio.h>
int main()
{
  char x;
  x='A';
  printf("A comme caractère : %c\n",x);
  printf("A comme entier : %d\n",x);
  return 0;
}
```

Donne :

A comme caractere : A
A comme entier : 65

Tous les caractères ne sont pas imprimables. La fonction `isprint(i)` a la valeur *vrai* si le caractère dont le code ASCII est `i` est imprimable, *fausse* sinon. Le programme suivant permet donc d'imprimer tous les caractères imprimables avec le nombre entier qui leur correspond :

```
#include<stdio.h>
#include<ctype.h>
int main()
{
  int i;
  for(i=0;i<256;i++)
    {
      if(isprint(i)) printf("caractère=%c code ascii=%d\n",i,i);
    }
  return 0;
}
```

On rappelle qu'une constante caractère s'écrit entre simples quotes, par exemple : `'A'`.

2 Chaînes

Une chaîne est une séquence de caractères. Il n'existe pas de variables de type chaîne. On traite une chaîne à l'aide d'un tableau de caractères ou d'un tableau-pointeur de caractères[1].

2.1 Constante explicite chaîne de caractères

C'est une chaîne placée entre double quotes, telle que : `"inca"`.
Donc `'A'` est une constante caractère et `"A"` est une constante chaîne de un caractère.
La notation `"inca"` représente une adresse, à partir de laquelle sont écrits successivement :

les quatre caractères `i`, `n`, `c` et `a`, chacun dans un octet
un octet nul (dont tous les bits sont à 0 et qui est noté `\0`) qui, par convention, indique la fin de la chaîne et est appelé caractère de terminaison

[1] dans la suite tous les exemples donnés avec un tableau pourraient être traités avec un tableau-pointeur

Comme un tableau une constante chaîne représente une adresse mais ne s'identifie pas à une adresse. Ceci est illustré par le programme suivant :

```
#include<stdio.h>
int main()
{
  int i;
  i=0;
  while(1)
    {
      printf("%c",*("inca"+i));
      if(*("inca"+i)=='\0') break;
      i++;
    }
  printf("\n");
  printf("valeur finale de i=%d\n",i);
  printf("sizeof(\"A\")=%d\n",sizeof("A"));
  printf("sizeof(\"inca\")=%d\n",sizeof("inca"));
  return 0;
}
```

dont le résultat est :

inca
valeur finale de i=4
sizeof("A")=2
sizeof("inca")=5

On constate que, comme pour un tableau, la chaîne `"inca"` représente une adresse mais que sa taille n'est pas celle d'une adresse mais celle des 4+1 octets qu'elle occupe.

2.2 Utilisation de tableaux de caractères pour manipuler les chaînes

On déclare un tableau dans lequel seront placés les caractères d'une chaîne :

```
#include<stdio.h>
#define N 20
int main()
{
  char x[N+1];
  ...
  return 0;
}
```

Le tableau doit avoir au moins un élément de plus que le nombre de caractères de la chaîne puisqu'il faut la place d'écrire le caractère de terminaison `'\0'` dans l'octet suivant immédiatement celui qui contient le dernier caractère.
Il existe des moyens pour ne pas avoir à manipuler les éléments du tableau un par un, c'est à dire les chaînes caractère par caractère, ce qui serait très lourd.
Dans la suite on appelle chaîne un tableau utilisé pour contenir des caractères.

2.3 Initialisation d'une chaîne dans sa déclaration

Une chaîne pourrait âtre initialisée dans sa déclaration comme tout tableau par :

```
  ...
  char x[N+1]={'i','n','c','a','\0'};
  ...
```

mais cette façon de faire est laborieuse et peu lisible.
Il existe une autre écriture plus pratique :

```
  ...
  char x[N+1]="inca";
  ...
```

2.4 Lecture de la valeur d'une chaîne au clavier et écriture à l'écran

Pour la lecture il existe deux fonctions :

`scanf` avec le format `%s`
`fgets`[2]

et pour l'écriture :

`printf` avec le format `%s`
`puts` ou `fputs`

2.4.1 `scanf` et `printf`

Supposons qu'on veuille lire au clavier et écrire à l'écran des noms de `N` caractères maximum. On peut, par exemple, utiliser :

```
#include<stdio.h>
#define N 20
int main()
{
  char x[N+1];
  printf("Entrer une chaine de %d caracteres au maximum",N);
  scanf("%s",x);
  printf("%s\n",x);
  return 0;
}
```

On écrit au clavier :

inca

et on obtient à l'écran :

inca

Le `scanf` lit au clavier tous les caractères jusqu'au premier caractère délimiteur exclu, qui peut être un blanc ou une fin de ligne[3] (comme dans le cas des `char`, `int` et `double`), les place dans les premiers éléments du tableau et ajoute le caractère de terminaison `\0`, occupant donc dans cet exemple les cinq premiers éléments du tableau.
Comme pour ce qui se passe avec les formats `%d`, `%lg` et `%c` le caractère délimiteur reste dans le tampon. Mais avec `%s`, comme avec `%d` et `%lg`, et contrairement au cas de `%c`, ce caractère délimiteur est sauté par un éventuel `%s` suivant. Il n'est donc pas nécessaire de faire précéder un `%s` suivant d'un blanc.
Le `printf` écrit tous les éléments jusqu'au caractère de terminaison exclu.

2.4.2 `fgets`

On voit tout de suite que la façon de faire précédente ne permet pas de lire des chaînes contenant des blancs.
Il existe donc également une fonction nommée `fgets` permettant de lire des chaînes contenant des blancs comme un seul bloc. Par exemple :

```
#include<stdio.h>
#define N 20
int main()
{
  char x[N+1];
  fgets(x,N+1,stdin);
  printf("%s\n",x);
  return 0;
}
```

On écrit au clavier :

inca et maya

et on obtient à l'écran :

inca et maya

[2]On n'utilisera pas `gets` jugé peu sûr par le compilateur lui-même.
[3]indiquée par un *Entrée* ou *Return*

`fgets(x,N+1,stdin)` lit la chaîne frappée au clavier et la place dans `x`, en limitant cependant le nombre de caractères transférés à `N`. Contrairement au cas de `scanf` :

un blanc dans la chaîne frappée au clavier n'est pas considéré comme un délimiteur, seule la fin de ligne en tient lieu
le délimiteur, c'est à dire le caractère de fin de ligne `\n`, ne reste pas dans le tampon
le délimiteur, c'est à dire le caractère de fin de ligne `\n`, est écrit dans la chaîne `x`, juste avant le caractère de fin de chaîne `\0`, s'il y a assez de place compte-tenu de la longueur de la chaîne à écrire et de la valeur de `N+1` : si on frappe au plus `N-1` caractères au clavier le `\n` est écrit, sinon non.

On verra dans la suite comment éviter que le caractère de fin de ligne `\n` ne soit écrit dans la chaîne quand cela est gênant.
La valeur retournée par `fgets` est l'adresse de la chaîne lue ou la valeur `NULL` en cas d'erreur.
Avec `fgets` on ne peut traiter qu'une chaîne à la fois contrairement à `scanf` pour lesquel on peut écrire par exemple :

```
scanf("%s%s%s",x,y,z);
```

Les deux possibilités ont donc leur utilité.

2.4.3 `puts` et `fputs`

`puts(x)` et `fputs(x,stdout)` affichent tous les deux le contenu de la chaîne `x` à l'écran en entier, que ce contenu comprenne des blancs ou non[4].
`puts(x)` passe ensuite à la ligne, `fputs(x,stdout)` non. Dans l'exemple précédent on pourrait donc remplacer :

```
printf("%s\n",x);
```

par :

```
puts(x);
```

ou :

```
fputs(x,stdout); fputs("\n",stdout);
```

2.4.4 La fonction `sscanf`

De la même façon que `scanf` permet de lire au clavier et `fscanf` dans un fichier, `sscanf` permet de lire dans un tableau de caractères[5]. En supposant par exemple que `i` est une variable entière et `x` une chaîne, on peut écrire :

`scanf("%d%s",&i,x);` si on veut les lire au clavier
`fscanf(fich,"%d%s",&i,x);` si on veut les lire dans un fichier
`sscanf(t,"%d%s",&i,x);` si on veut les lire dans une seconde chaine nommée `t`

`sscanf` présente un avantage par rapport à `scanf` et `fscanf` : s'il y a trop de caractères à lire par rapport au format les caractères excédentaires ne restent pas dans le tampon et ne perturbent pas les `sscanf` suivants. Combiné à `fgets`, `sscanf` permet de lire des chaînes, mais aussi des entiers, des réels et des caractères sans qu'une éventuelle inadéquation des données au format ne provoque une totale et définitive désorganisation de la lecture. C'est ce qui est présenté au paragraphe suivant.

2.4.5 Combinaison de `fgets` et `sscanf`

Considérons le programme suivant qui lit trois fois au clavier un réel et une chaîne et les imprime :

```
#include<stdio.h>
#define N 10
#define K 3
int main()
{
int k; double x; char ch[N+1];
for(k=1;k<=K;k++)
{
scanf("%lg%s",&x,ch);
printf("k=%d x=%lg ch=%s\n",k,x,ch);
}
return 0;
}
```

[4] comme d'ailleurs `printf("%s\n",x)`
[5] Il existe la fonction correspondante pour écrire : `sprintf`.

Si on entre les données conformément au format :

Le programme affiche : L'utilisateur écrit :

1.8 DUPONT
k=1 x=1.8 ch=DUPONT
3.4 DURAND
k=2 x=3.4 ch=DURAND
9.7 MARTIN
k=3 x=9.7 ch=MARTIN

tout se passe normalement.
Mais si, par exemple, il y a un blanc de trop dans les données :

Le programme affiche : L'utilisateur écrit :

1.8 DU PONT
k=1 x=1.8 ch=DU
k=2 x=1.8 ch=DU
k=3 x=1.8 ch=DU

les données sont faussées non seulement sur la première ligne, ce qui est normal, mais aussi sur toutes les suivantes.
Si maintenant on remplace le `scanf` par une lecture en deux temps avec `fgets` et `sscanf` :

```
#include<stdio.h>
#define N 10
#define NS 40
#define K 3
int main()
{
int k; double x; char ch[N+1];
char s[NS+1];
for(k=1;k<=K;k++)
{
fgets(s,NS,stdin);                    // ces deux lignes remplacent
sscanf(s,"%lg%s",&x,ch);              // le scanf
printf("k=%d x=%lg ch=%s\n",k,x,ch);
}
return 0;
}
```

on obtient maintenant :

Le programme affiche : L'utilisateur écrit :

1.8 DU PONT
k=1 x=1.8 ch=DU
3.4 DURAND
k=2 x=3.4 ch=DURAND
9.7 MARTIN
k=3 x=9.7 ch=MARTIN

La fraction de chaîne `PONT` est perdue mais ensuite la lecture se poursuit normalement.

2.4.6 Conclusion

Si on doit lire des chaînes ne comportant pas de blancs et dont on est certain qu'elles sont conformes au format de lecture[6] on peut se contenter d'utiliser `scanf` ou `fscanf`. Sinon il vaut mieux lire en deux temps avec `fgets` et `sscanf`.

2.5 Lecture et écriture dans un fichier

Tout ce qui vient d'être dit se transpose directement à la lecture et à l'écriture dans un fichier, il suffit de remplacer :

[6] Ce peut être le cas par exemple si elles ont été écrites dans un fichier avec ce format

`scanf("%s",x)` par `fscanf(fich,"%s",x)`
`printf("%s\n",x)` par `fprintf(fich,"%s\n",x)`
`fgets(x,N+1,stdin)` par `fgets(x,N+1,fich)`
`fputs(x,stdout)` par `fputs(x,fich)`

`fich` étant un pointeur sur un `FILE`.
Comme dans le cas où `fgets` lit au clavier (appelée avec `stdin`), la valeur retournée est l'adresse de la chaîne lue ou la valeur `NULL` en cas d'erreur. Mais dans le cas où `fgets` lit dans un fichier la valeur `NULL` peut aussi signifier qu'une fin de fichier, sans rien d'autre avant, a été rencontrée, ce qui permet de savoir quand il faut arrêter la lecture dans un fichier dont on ne connaît pas le nombre d'enregistrements.

3 Opérations sur les chaînes

Les chaînes étant des tableaux, il n'est pas possible d'écrire une affectation avec le signe `=` comme dans l'exemple suivant :

```
#define N 20
...
char x[N+1]="inca",y[N+1];
y=x;
...
```

(les noms de tableaux sans les crochets `[]` représentent des adresses et ne peuvent donc se trouver à gauche d'un signe `=`).
Pour l'affectation et pour toutes les opérations sur les chaînes, il faut recourir à des fonctions dont on examine maintenant les plus courantes. On décrit ces fonctions uniquement dans leur version la plus sûre, c'est à dire celle dans laquelle on peut imposer un nombre maximum de caractères à traiter. Leur emploi nécessite la bibliothèque de chaînes accessible par `#include<string.h>`.

3.1 Copie de chaîne : `strncpy(destination,origine,longueur_max)`

C'est elle qui permet l'attribution de la valeur d'une chaîne à une autre chaîne :

```
#include<stdio.h>
#include<string.h>
#define N 20
int main()
{
  char x[N+1]="inca",y[N+1];
  strncpy(y,x,N);
  puts(y);
  return 0;
}
```

L'ordre des arguments est chaîne de destination $\longrightarrow$ chaîne d'origine.
`longueur_max` permet de spécifier le nombre maximum de caractères pouvant être recopiés. Il faut s'assurer que la chaîne de destination a une longueur suffisante.
La valeur de `strncpy` est l'adresse de la chaîne de destination.

3.2 Longueur d'une chaîne : `strlen(chaine)`

Fournit la longueur effective de la chaîne contenue dans la chaîne `x`, c'est à dire le nombre de caractères précédents le caractère de terminaison. Par exemple :

```
#include<stdio.h>
#include<string.h>
#define N 20
int main()
{
  char x[N+1]="inca";
  printf("%d\n",strlen(x));
  ...
}
```

donne :

4

3.3 Concaténation de chaînes : `strncat(chaine_1,chaine_2,longueur_max)`

Elle permet d'ajouter la chaîne `chaine_2` à la suite de la chaîne `chaine_1` de façon à n'en former qu'une seule[7]. `longueur_max` permet de spécifier le nombre maximum de caractères pouvant être ajoutés. Exemple :

```
#include<stdio.h>
#include<string.h>
#define N 20
int main()
{
  char x[2*N+1]="incas";
  char y[N+1]=" et mayas";
  puts(x); puts(y);
  strncat(x,y,N);
  puts(x);
  return 0;
}
```

donne le résultat :

incas
et mayas
incas et mayas

Il faut évidemment prévoir la place nécessaire dans `chaine_1`.
La valeur de `strncat` est l'adresse de la chaîne `chaine_1` ou la valeur `NULL` en cas d'erreur.

3.4 Comparaison de chaînes : `strncmp(chaine_1,chaine_2,longueur_max)`

Cette fonction a une valeur :
- négative si `chaine_1` précéde `chaine_2` dans l'ordre alphabétique
- nulle si `chaine_1` et `chaine_2` sont identiques
- positive si `chaine_1` suit `chaine_2` dans l'ordre alphabétique

`longueur_max` permet de limiter le nombre de caractères comparés.
La fonction `strnicmp(chaine_1,chaine_2,longueur_max)` fait la même chose que `strncmp`, mais sans tenir compte de la différence entre minuscules et majuscules.

3.5 Extraction de champs : `strtok(chaine_1,chaine_2)`

`strtok` permet d'extraire de la chaîne `chaine_1` des sous-chaînes délimitées par des caractères définis dans la chaine `chaine_2`. Exemple :

```
#include<stdio.h>
#include<string.h>
#define N 100
int main()
{
  char x[N+1]="nsthiurds(;q;jhs ?kjs^";
  char y[N+1]="dh";
  puts(strtok(x,y));
  return 0;
}
```

`strtok(x,y)` recherche dans `chaine_1` l'adresse du premier caractère qui n'est ni un `d` ni un `h`. Il cherche ensuite après ce caractère la première occurrence d'un `d` ou d'un `h` et la remplace par une fin de chaîne `\0`. La chaîne obtenue est la valeur de `strtok(x,y)`. Ici on obtiendra donc :

nst

Pour obtenir les chaînes suivantes comprises entre les mêmes délimiteurs il faut appeler `strtok` avec la valeur `NULL` à la place de la chaîne `x` :

[7] Donc en supprimant le caractère de terminaison de `chaine_1`

```
strtok(NULL,y);
```

jusqu'à ce que la valeur de `strtok(NULL,y)` devienne `NULL`.
On peut, si on veut, changer les délimiteurs contenus dans `chaine_2` à chaque appel.
En remplaçant, dans le programme précédent, la ligne :

```
puts(strtok(x,y));
```

par :

```
char *z;
puts(strtok(x,y));
while((z=strtok(NULL,y))!=NULL) puts(z);
```

on obtiendra donc :

nst
iur
s(;q ;j
*s ?kjs^*

Une des principales applications de `strtok` est d'extraire des champs dans une chaîne. Un exemple sera vu dans la suite.
On peut aussi l'utiliser pour que `fgets` n'écrive pas le caractère de fin de ligne dans la chaîne qu'il remplit : voir la fonction `fgets_a` au paragraphe suivant.

3.6 Autres fonctions

Il existe d'autres fonctions, permettant d'extraire des sous-chaînes d'une chaîne suivant différents critères (voir *La référence du C norme ANSI/ISO*, Claude Delannoy page 482).

4 Exemples

4.1 Constitution d'un fichier de renseignements

Programme permettant de constituer un fichier avec quatre champs par enregistrement : nom, prénom, sujet de projet, groupe de TD. Chaque champ peut contenir des blancs.

```
#include<stdio.h>
#include<string.h>
#define N 20
#define NN 72
char *fgets_a(char *u,int n,FILE *flux)
{
  char *v;
  v=fgets(u,n,flux);
  u=strtok(u,"\n"); // ici on peut mettre le signe = parcequ'il s'agit
                   // de pointeurs et non de tableaux
  return(v);
}
int main()
{
  char p[N+1],n[N+1],suj[NN+1],gr[N+1],nom_fich[NN+1];
  int g,i;
  printf("Nom du fichier dans lequel il faut écrire ? (%d lettres maximum) : ",NN);
  fgets_a(nom_fich,NN+1,stdin);
  FILE *fich=fopen(nom_fich,"w");
  printf("Quand c'est terminé répondre \"fin\" à la place du nom\n");
  i=0;
  while(1)
    {
      printf("Nom ? (%d lettres max) : ",N); fgets_a(n,N+1,stdin);
      if(strcmp(n,"fin")==0) break;
      printf("Prénom ? (%d lettres max) : ",N); fgets_a(p,N+1,stdin);
```

```
      printf("Sujet de projet ? (%d lettres max) : ",NN); fgets_a(suj,NN+1,stdin);
      printf("Groupe de TD ? : "); fgets_a(gr,N+1,stdin); sscanf(gr,"%d",&g);
      i++;
      fprintf(fich,"%s | %s | %s | %d\n",n,p,suj,g);
    }
  printf("Le fichier %s contient %d enregistrements\n",nom_fich,i);
  fclose(fich);
  return 0;
}
```

Le programme affiche :	L'utilisateur écrit :
Nom du fichier dans lequel il faut écrire ? (72 lettres maximum) :	*etudiants_sujets_constitution.res*
Quand c'est terminé répondre "fin" à la place du nom	
Nom ? (20 lettres max) :	*MARTIN*
Prénom ? (20 lettres max) :	*Luc Henri*
Sujet de projet ? (72 lettres max) :	*Aberrations d'une lentille réelle*
Groupe de TD ? :	*5*
Nom ? (20 lettres max) :	*DE BROGLIE*
Prénom ? (20 lettres max) :	*Louis*
Sujet de projet ? (72 lettres max) :	*Propagation d'un paquet d'ondes*
Groupe de TD ? :	*2*
...	*...*
Nom ? (20 lettres max) :	*fin*
Le fichier etudiants_sujets_constitution.res contient ... enregistrements	

et dans le fichier `etudiants_sujets_constitution.res` on trouve :

```
MARTIN | Luc Henri | Aberrations d'une lentille réelle | 5
DE BROGLIE | Louis | Propagation d'un paquet d'ondes | 2
...
```

On a placé des séparateurs, ici une barre verticale |, nécessaires pour la relecture car il peut y avoir des blancs dans les chaînes.

4.2 Extraction à partir d'un fichier de renseignements

Le programme suivant lit le fichier créé par le programme précédent et écrit dans un second fichier uniquement les deux champs correspondant au nom et au sujet, pour un groupe de TD choisi par l'utilisateur.

```
#include<stdio.h>
#include<string.h>
#define N 20
#define NN 72
#define NNN 256
int main()
{
  char n[N+1],p[N+1],suj[NN+1],gr[N+1];
  char nom_fich_1[NN+1],nom_fich_2[NN+1],ligne[NNN+1];
  int g,gt;
  printf("Nom du fichier dans lequel il faut lire ? (%d lettres maximum) : ",NN);
  fgets_a(nom_fich_1,NN+1,stdin);
  printf("Nom du fichier dans lequel il faut écrire ? (%d lettres maximum) : ",NN);
  fgets_a(nom_fich_2,NN+1,stdin);
  FILE *fich_1=fopen(nom_fich_1,"r"),*fich_2=fopen(nom_fich_2,"w");
  printf("Quel groupe de TD est selectionné ? : "); scanf("%d",&gt);
  fprintf(fich_2,"GROUPE : %d\n",gt);
  while(fgets_a(ligne,NNN+1,fich_1)!=NULL)
    {
      strcpy(n,strtok(ligne,"|"));  // extraction du premier champ du premier fichier
      strcpy(p,strtok(NULL,"|"));   // extraction du second champ         "
      strcpy(suj,strtok(NULL,"|")); // extraction du troisième champ      "
      strcpy(gr,strtok(NULL,"\n")); // extraction du quatrième champ      "
```

```
      sscanf(gr,"%d",&g);
      if(g!=gt) continue;
      fputs(n,fich_2);     // écriture du premier champ du second fichier
      fputs(" | ",fich_2); // on place un séparateur entre le premier et le
                           // second champ du second fichier
      fputs(suj,fich_2);   // écriture du second champ du second fichier
      fputs("\n",fich_2);  // passage à la ligne dans le second fichier
                           // (fputs ne le fait pas tout seul)
    }
  fclose(fich_1); fclose(fich_2);
  return 0;
}
```

Le programme affiche :	L'utilisateur écrit :
Nom du fichier dans lequel il faut lire ? (72 lettres maximum) :	*etudiants_sujets_constitution.res*
Nom du fichier dans lequel il faut écrire ? (72 lettres maximum) :	*etudiants_sujets_extraction.res*
Quel groupe de TD est selectionné ? :	*2*

et, dans le fichier *etudiants_sujets_extraction.res*, on trouve :

```
GROUPE : 2
DE BROGLIE  |  Propagation d'un paquet d'ondes
...
```

4.3 Lecture d'une fonction au clavier

On reprend le programme vu au chapitre **Fonctions** qui calcule l'intégrale d'une fonction :

```
#include <stdio.h>
#include <math.h>
double f(double x)
{
return(1./(1.+x*x));
}
double integ(double a,double b,int n)
{
double h,s; int i;
h=(b-a)/n; s=0.;
for(i=1; i<=n; i++) s=s+f(a+(i-0.5)*h);
return (h*s);
}
int main()
{
double x1,x2; int np;
printf("Borne inferieure=? "); scanf("%lg",&x1);
printf("Borne superieure=? "); scanf("%lg",&x2);
printf("Nombre de rectangles=? "); scanf("%d",&np);
printf("Integrale=%lg\n",integ(x1,x2,np));
return 0;
}
```

Si on veut intégrer une fonction qui ne figure pas déjà dans le programme il faut modifier ce dernier pour y inclure la nouvelle fonction. On souhaite que cette modification soit faite automatiquement par un programme supervisant l'opération de telle façon que l'utilisateur puisse fournir la fonction mathématique à intégrer au clavier, comme s'il s'agissait d'une simple donnée fournie en cours d'exécution. L'utilisateur aura, en apparence, un seul programme à faire exécuter qui lui demandera la fonction à intégrer, les bornes d'intégration, le nombre de rectangles et qui lui fournira le résultat. En réalité il y a deux programmes, le superviseur et le programme qui intègre. Le superviseur est exécuté en premier, il demande la fonction à intégrer et l'inscrit dans un fichier. Ensuite il commande la compilation du programme d'intégration qui insère la fonction à intégrer dans ce programme, puis l'exécution.
Le programme superviseur, nommé *superviseur_integration.c*, peut s'écrire :

```
#include <stdio.h>
#include<stdlib.h> // indispensable pour pouvoir utiliser la fonction system
```

```
#define N 128
int main()
{
  char x[N+1];
  FILE *fich=fopen("fff","w");
  printf("Ecrire la fonction a integrer en C (%d caracteres max),\n",N);
  printf("la variable doit s'appeler x : ");
  fgets(x,N+1,stdin);
  fputs(x,fich);
  fclose(fich);
  system("g++ -o toto -lm integration.c && ./toto && rm toto");
  system("rm -f fff");
  return 0;
}
```

Le programme d'intégration, nommé *integration.c*, est modifié comme suit :

```
#include <stdio.h>
#include <math.h>
double f(double x)
{
return(
#include "fff"                          // <--- Unique modification
);
}
double integ(double a,double b,int n) // <--- Aucun changement
{
...
}
int main()                           // <--- Aucun changement
{
...
}
```

A la place de la fonction à intégrer on met un `include` qui, au début de la compilation, insère simplement les lignes contenues dans le fichier nommé `fff`.
Le programme superviseur demande la fonction à intégrer et l'écrit dans un fichier.
Par la fonction `system`[8] il commande :

1. la compilation du programme d'intégration[9], la fonction à intégrer est alors insérée au bon endroit dans le fichier *integration.c*
2. l'exécution du programme d'intégration

Exemple d'exécution :

Le programme affiche :	L'utilisateur écrit :
	ccc superviseur_integration.c
Ecrire la fonction a integrer en C (128 caracteres max),	
la variable doit s'appeler x :	*1/(1+x*x)*
Borne inferieure= ?	*0*
Borne superieure= ?	*1*
Nombre de rectangles= ?	*1000*
Integrale=0.785398	

4.4 Ouverture d'une séquence de fichiers

Le programme suivant crée et ouvre `NF` fichiers nommés *xxx_0.res*, *xxx_1.res*, etc., *xxx.c*, étant le nom du fichier contenant le programme, quelle que soit la chaîne *xxx* à condition que sa longueur soit strictement inférieure à `L-6`. Si ce n'est pas le cas un diagnostic est émis et le programme s'interrompt. Il suffit alors d'augmenter la valeur de `L`. Le nombre de fichiers à ouvrir est de 10 au maximum, il n'est pas possible d'augmenter ce maximum sans modifier le programme.

[8] qui permet de faire exécuter, depuis un programme C, n'importe quelle commande *Linux*
[9] on impose à l'exécutable de s'appeler *toto* pour qu'il n'y ait pas de confusion avec le *a.out* de *superviseur_integration.c*

```
#include<stdio.h>
#include<string.h>
#define L 73
#define NF 4
int main(int n, char *t[])
{
  int i;
  char x[L]; char *z;
  if(NF>10) {printf("Nombre de fichiers trop grand\n"); exit(0);} // il n'est pas possible d'augmenter c
  FILE *fich[NF];
  if((strlen(z=strtok(t[0],"./")))>L-1-6) {printf("Nom de fichier trop long\n"); exit(0);} // 6 est le r
  for(i=0;i<NF;i++)
    {
      sprintf(x,"%s_%d.res",z,i);
      fich[i]=fopen(x,"w");
      fprintf(fich[i],"En-tete du fichier %s\n",x);
    }
  // ...
  for(i=0;i<NF;i++)
    {
      fclose(fich[i]);
    }
  return 0;
}
```

Le nom du programme passé en argument de `main` par l'intermédiaire du pointeur `t[0]` est celui de l'exécutable et non celui du fichier *xxx.c*. Si cet exécutable s'appelle systématiquement *a.out* comme c'est le cas par défaut, le nom du fichier C est perdu. Il faut donc imposer au compilateur de conserver le nom du fichier C dans celui de l'exécutable, ce qui peut être obtenu en compilant avec l'option *-o* et en donnant par exemple le nom *xxx.exe* à l'exécutable :

```
g++ -lm -o xxx.exe xxx.c
```

ce qui est fait par la commande *ccc*.
Le nom transmis par `t[0]` est alors `./xxx.exe` et la fonction `strtok(t[0],"./")` en extrait la chaîne `xxx`.

Remarque

Pour travailler avec des chaînes de caractères il est souvent beaucoup plus pratique d'écrire directement dans un fichier avec un éditeur ou d'utiliser les nombreux outils *Linux* de traitement de fichiers. Mais il peut aussi être inévitable de le faire dans un programme C.

Annexe : prototypes de quelques fonctions

```
int printf(const char *,...)
int fprintf(FILE *,const char *,...)
int sprintf(char *,const char *,...)
int scanf(const char *,...)
int fscanf(FILE *,const char *,...)
int sscanf(char *,const char *,...)
char *fgets(char *,int,FILE *)
int puts(const char *)
int fputs(const char *,FILE *)
char *strncpy(char *,const char *,size_t)
size_t strlen(const char *)
char *strncat(char *,const char *,size_t)
int strncmp(const char *,const char *,size_t)
char *strtok(char *,const char *)
```