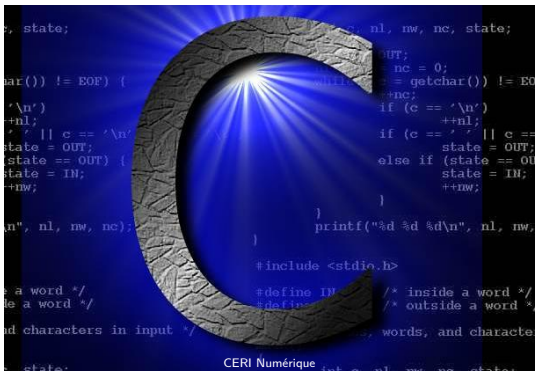
Algorithmique et langage C

— pointeurs et allocation dynamique —



Luc Fabresse
luc.fabresse@imt-nord-europe.fr

version 1.2

Plan

Gestion dynamique de mémoire ?

Pourquoi ?

Utile lorsqu'on ne connaît pas à l'avance le nombre de valeurs à créer

Exemple

Nombre d'éléments d'un tableau fourni par l'utilisateur d'un logiciel

Plan

Les tableaux de taille variable (C99)

```
void foo (int n)
{
  int tab[n];
  // ...
}
```

Portée

Variables automatiques, détruites à la fin du bloc dans lequel elles sont déclarées

Interdit dans les struct

Il n'est pas possible de déclarer un tableau de taille variable dans une structure

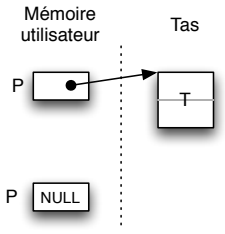
Plan

Les procédures `reserve` et `libere`

`reserve` et `libere`

Si P est un pointeur sur une variable de type T

- reserve(P) acquiert dans le Tas une zone pointée par P, utilisable par une variable de type T. La place réservé est accessible via *P, qui signifie objet pointé par le pointeur P.
- libere(P) restitue au Tas la place pointée par P, en détruisant le lien entre P et cette place P ne pointe sur rien prend pour valeur NULL (*adresse invalide*)



Allocation dynamique de mémoire (1)

Fonction d'allocation `malloc`

Signature `void *malloc(unsigned int t)`

Effet demande au système de réserver un espace mémoire pouvant contenir t octets, retourne un pointeur vers l'espace alloué ou NULL en cas d'erreur

Remarques
- Nécessité de convertir le type du pointeur résultat !
- Pour utiliser `malloc` il faut inclure `stdlib.h`
- `calloc(n,t)` est équivalent à `malloc(p)` où `p==n*t`
- toutefois, `malloc` n'initialise pas la zone avec des 0

Remarque

Préférer `calloc` car l'intention du développeur de créer un tableau ou pas est rendue explicite à travers le premier paramètre n.

Allocation dynamique de mémoire (2)

Fonction d'allocation `calloc`

Signature `void *calloc(unsigned int n, unsigned int t)`

Effet demande au système de réserver un espace mémoire pouvant contenir n éléments de t octets chacun et retourne un pointeur vers l'espace alloué ou `NULL` en cas d'erreur

Remarques
- initialise la zone allouée avec des 0
- Le nombre d'octets (`unsigned int n`) peut être obtenu avec `sizeof`
- Nécessité de convertir le type du pointeur résultat !
- Pour utiliser `calloc` il faut inclure `stdlib.h`

Allocation dynamique de mémoire (3)

Fonction de ré-allocation `realloc`

Signature `void *realloc(void *p, unsigned int t)`

Effet demande au système d'agrandir l'espace mémoire démarrant à l'adresse p pour contenir t octets. Retourne un pointeur vers l'espace alloué ou NULL en cas d'erreur. Peut être différent de p s'il y a allocation d'une autre zone mémoire et transfert du contenu de p

Remarques
- Nécessité de convertir le type du pointeur résultat !
- Pour utiliser `realloc` il faut inclure `stdlib.h`

Libération de la mémoire

Règle d'or :

Tout espace mémoire dynamiquement réservé doit être libéré !

Fonction de libération de la mémoire `free`

Signature `void free(void *p)`

Effet Le paramètre p doit être un pointeur sur un bloc mémoire précédemment alloué (résultat de calloc par exemple)

Remarques
- Pour utiliser `free` il faut inclure `stdlib.h`

Exemple

```c
int *pi;
char *pc;

pi = (int *) malloc(sizeof(int)); // allouer de la mémoire pour 1 int

pc = (char *) calloc(10, sizeof(char)); // allouer de la mémoire pour 10 char
*pi = 10;
pc[0] = 'H';
pc[1] = 'I';
pc[2] = '!';
pc[3] = '\0';

// libération de la mémoire
free(pi);
free(pc);
```


Plan

Structures et `malloc`

Pointeur dans une strutcture

```c
#include <stdio.h>
#include <stdlib.h>

struct incomplet {
  int n;
  int *tab; };

int main (void) {
  struct incomplet *a = NULL;

  a = malloc (sizeof (*a));
  if(a) {
    a->n = 3;
    a->tab = malloc(3*sizeof(int));
    if(a->tab) {
       for(int i = 0; i < a->n; i++)
         a->tab[i] = i;

       printf ("a->tab = { ");
       for(int i = 0; i < a->n; i++)
         printf ("%d, ", a->tab[i]);
       printf(" };\n");

       free(a->tab); a->tab = NULL;
    }
    free(a); a = NULL;
  }
  return 0;
}
```


Structures et tableaux de taille variable (C99)

Taille facultative si dernier champ de la structure

```c
#include <stdio.h>
#include <stdlib.h>

struct incomplet {
  int n;
  int tab[];
};

int main (void) {
  struct incomplet *a = NULL;

  a = malloc (sizeof (*a) + 3 * sizeof (int));
  if (a) {
    a->n = 3;
    for (int i = 0; i < a->n; i++)
      a->tab[i] = i;

    printf ("a->tab = { ");
    for (int i = 0; i < a->n; i++)
      printf ("%d, ", a->tab[i]);
    printf (" };\n");

    free (a), a = NULL;
  }
  return 0;
}
```


Plan

Paramètres sur la ligne de commande

main(int argc, char *argv[])

- argc : nombre de paramètres
- argv : pointeur sur des chaînes de caractères chacune étant un paramètre

```
$ ./surface_triangle 3 4 5
surface du triangle = 6.00
```

Exemple : surface_triangle.c

```c
/** compute the surface of a triangle given the
 * length of its sides, passed on the command line
 * ./surface_triangle 3 4 5 **/
#include <stdio.h>
#include <stdlib.h> // atof
#include <math.h> // sqrt

int main(int argc, char *argv[]) {
   float a,b,c,s,surface;
   int i;

   if ( argc != 4 ) {
      printf("there must be exactly 3 parameters\n");
      return -1; // retourne une erreur
   }

   a = atof(argv[1]); b = atof(argv[2]); c = atof(argv[3]);

   if( a+b<=c || a+c<=b || b+c<=a ) {
      printf("this is not a triangle\n");
      return -1;
   }

   s = (a+b+c)/2;
   surface = sqrt(s*(s-a)*(s-b)*(s-c));
   printf("triangle surface = %.2f\n",surface);
   return 0;
}
```


Plan

La fonction scanf

Syntaxe

```
int scanf(<format>, <arg1>, <arg2>, ... , <argn>)
```

Traitement réalisé

scanf doit stocker les valeurs saisies dans arg1, arg2, ...,argn

Passage de paramètres par copie

Il est impossible de modifier la valeur d'un paramètre d'une fonction !

Les paramètres de scanf

Solution : Passer l'adresse de la zone mémoire à remplir

```c
...
int i,j;

// we pass the addresses of variables to scanf
scanf("%d%d",&i,&j);

...
```

Passage de paramètres par copie

Ce sont les adresses des variables i et j qui sont recopiées

Le résultat de scanf

Valeur de retour de scanf

```c
...
int i,j,nbAssigned;
nbAssigned=i=j=0;

// we pass the addresses of variables to scanf
nbAssigned = scanf("%d%d",&i,&j);

// check of the return value of scanf
if(nbAssigned!=2)
   printf("input error\n");

printf("%d %d\n",i,j);
...
```

Bonne pratique

Toujours tester le résultat de scanf pour être sûr que les saisies ont bien été effectuées

Saisir une chaîne de caractères %s

Exemple de code

```c
...
char s[11];
scanf("%s",s); /* pas de & car s est un pointeur */
printf("[%lu] >%s<\n",strlen(s),s);
...
```

Exemples de saisies

```
$ ./a.out
a
[1] >a<

$ ./a.out
    lkweqdmql
[9] >lkweqdmql<

$ ./a.out
   sd ed ef
[2] >sd<
```

Fonctionnement de %

- supprime les espaces avant le motif à reconnaître
- s'arrête au premier caractère *blanc* (espace,\n) ou lorsque la longueur de champ a été atteinte (si spécifiée) avant un blanc
- ajoute un terminateur de chaîne (\0) automatiquement

Buffer overflow lors d'une saisie de chaîne avec %s

Exemple de code

```c
...
char s[2]; /* ATTENTION 2 caractères au plus dont le \0 terminal */
scanf("%s",s);
printf("[%lu] >%s<\n",strlen(s),s);
...
```

Exemples de saisies

```
$ ./a.out
asd
[3] >asd<

$ ./a.out
    lkweqdmql
[9] >lkweqdmql<

$ ./a.out
   sd ed ef
[2] >sd<

$ ./a.out
qwertyuiopasdfghjkl
[19] >qwertyuiopasdfghjkl<
[1] 79754 segmentation fault ./a.out
```

Règles d'or

1. Toujours allouer un espace mémoire de taille suffisante pour la chaîne à saisir ET le caractère \0 terminal
2. Contrôler la taille des saisies

Exemple protégé

```c
...
char s[11];
scanf("%10s",s);
printf("[%lu] >%s<\n",strlen(s),s);
...
```


Saisir tableau de caractères avec %c

Exemple

```c
...
char tab[11] = {0}; /* la derniere case est pour le \0 terminal */
scanf("%10c", tab); /* recuperation de 10 char */
printf("lu>%s<\n", tab);
...
```

Exemples de saisies

```
$./prog
azertyuiop
lu>azertyuiop<

$./prog
     azertyuiop
lu> azer<

$./prog
  we er er rt
lu> we er er<
```

Fonctionnement de %c

- lit par défaut **un** seul caractère
- ne supprime pas les espaces en tête de saisie : ils sont intégrés dans le motif reconnu
- n'ajoute pas de caractère final : **il faut ajouter manuellement** \0 si l'on souhaite travailler sur une chaîne valide

Eliminer les blancs lors de la saisie d'un tableau de caractères

Exemple

```
...
char tab[11] = {0};
scanf(" %10c", tab); /* ESPACE avant le % */
printf("lu>%s<\n", tab);
...
```

Exemples de saisies

```
$./a.out
      azertyuiop
lu>azertyuiop<

$./a.out
  we er er rt
lu>we er er r<
```

Attention

Un espace dans la chaîne de format de `scanf` avant un % permet d'éliminer les *blancs* dans le motif à reconnaître qui suit

Ne saisir que des lettres

Exemple

```
char tab[81] = {0};
scanf(" %80[a-z]", tab);
```

Explication

- l'espace devant le % indique qu'il faut éliminer tous les caractères *blancs* avant de commencer la reconnaissance du motif
- le motif saisi fera au plus 80 caractères
- le motif doit obligatoirement être constitué de caractères compris entre a et z

Expressions rationnelles/régulières

- entre []
- caractères autorisés dans le motif
- par défaut, ne supprime pas les espaces devant le motif (comme pour %c)

Ne saisir que des lettres

Exemple

```
char tab[81] = {0};
scanf(" %80[atgc]", tab);
scanf(" %80[a-zA-Z]", tab);
```

Explication

- l'espace devant le % indique qu'il faut éliminer tous les caractères *blancs* avant de commencer la reconnaissance du motif
- le motif saisi fera au plus 80 caractères
- le motif doit obligatoirement être constitué de caractères compris entre a et z

Expressions rationnelles/régulières

- entre []
- caractères autorisés dans le motif
- par défaut, ne supprime pas les espaces devant le motif (comme pour %c)

Saisir une ligne

Exemple

```
char tab[81] = {0};
char c;

if (scanf("%80[^\n]", tab) == 1) {
  /* saisie reussie */
  if ( (c = getchar()) != '\n') {
    /* on n'a pas tout le motif (> 80) */

  }
}
else{
  /* echec de la saisie */

}
```

Explications

- `[^\n]` n'importe quel caractère sauf \n (fin de ligne)
- `getchar`, fonction permettant de saisir un et un seul caractère
- Dans le cas d'une erreur de saisie, le buffer `stdin` doit absolument être vidé avant d'effectuer une nouvelle saisie

Le buffer `stdin`

Exemple

```
char tab[11] = {0};
scanf("%10c", tab);
printf("lu>%s<\n", tab);
scanf("%10c", tab);
printf("lu>%s<\n", tab);
```

Exemples de saisies

```
$./a.out
12345678901234567890123456789O
lu>1234567890<
lu>1234567890<
// il reste 1234567890 dans stdin
```

Buffer `stdin`

- N'est pas vidé automatiquement
- Contient tous les caractères entrés par l'utilisateur
- `scanf`, `fget`, ... vont lire dans ce buffer

Vider `stdin`

Solution 1

```
int c;
while ( ((c = getchar()) != '\n') && c != EOF){};
```

Solution 2

```
scanf("%*[^\n]");
getchar();
```


Exemple de boucle de saisie complète

```
#include <stdio.h>

int main(void){
  int nombre = 0, ok = 0, nbValeurAffectee;

  printf("saisissez un nombre :\n");
  while (!ok){

    nbValeurAffectee = scanf("%d%*[^\n]", &nombre);
    printf("retour : %d\n", nbValeurAffectee);
    if ( !nbValeurAffectee ){ // nbValeurAffectee == 0 ?
      /* echec de la saisie */
      int c;
      while ( ((c = getchar()) != '\n') && c != EOF);

      printf("on vous a demande de saisir un nombre\n");
      printf("veuillez recommencer :\n");
    }
    else {
      /* reussite de la saisie */
      getchar(); /* on enleve le '\n' restant */

      printf("saisie acceptee\n");
      ok = 1; /* sort de la boucle */
    }
  }
  // stdin est vide et nombre contient une valeur
  return 0;
}
```


Plan

Place aux exercices !