Le langage C

ASR2 - Système

Semestre 2, année 2012-2013

Mars 2013

Première partie

Introduction

Votre état d'esprit

```
 1    loadi 0
 2    store k
 3    loadi t
 4    store p
 5  loop
 6    load k
 7    sub n
 8    jzero end
 9    load k
10    storex p
11    loadi 1
12    add k
13    store k
14    loadi 1
15    add p
16    store p
17    jmp loop
18  end
19    halt 0
20  n word 9
21  k word 0
22  p word 0
23  t word 0
```

Le commencement

Apparu en 1972 dans les laboratoires de Bell, il est devenu rapidement un des langages les plus utilisés au monde [a].

a. http://www.tiobe.com

Il appartient aux langages dit **impératifs** comme Fortran ou encore Pascal qui l'ont précédé.

C'est un langage dit **bas-niveau** (i.e. proche du langage machine) contrairement à des langages comme Java ou encore PHP.

De C à C++

Apparu dans les années 1980 dans les laboratoires de Bell (toujours les mêmes), il se voulait à l'origine comme une extension du C.

Bonne nouvelle

Vous ne serez pas dépaysés par la syntaxe [a].

a. qui a aussi fortement imprégné celles de Java, C#, PHP, Javascript, Python, Perl etc.)

Mauvaise nouvelle

Il va falloir apprendre à se passer de mécanismes que vous connaissez en C++, mais qui n'existent pas en C.

Ce qui n'existe pas en C

- le passage de paramètres par références,
- la surcharge,
- les chaînes de caractères en tant qu'objet,
- les opérations << et >> pour lire et écrire sur des *streams*,
- *new* et *delete*,
- les classes et donc l'héritage ainsi que le polymorphisme,
- les *templates*,
- et plein d'autres choses...

Caractéristiques du C

Bas niveau

Utiliser au mieux les possibilités du matériel.

Simple

Limiter les fonctions de base du langage.

Portable

Écrire des programmes qui tournent sur des machines de types différents.

Structuré

Blocs, *boucles* et *alternatives* sont de la partie.

Concis

Limiter le nombre de mots-clés (32 en C, 50 en Java, 77 en C# et 357 en Cobol).

Deuxième partie

Let's code !

Hello World !

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

int main()
{
   printf("Hello world!\n");
   return EXIT_SUCCESS;
}
```

La ligne pour compiler le source (*myprog.c*) est la suivante :

```
$ gcc -std=c99 myprog.c -o myprog
```

On peut ajouter quelques options de compilation supplémentaires :

- **-Wall** notifie toute les *warnings*.
- **-Werror** transforme tous les *warnings* en erreurs.
- **-pedantic** lance des *warnings* pour des fautes de style...

Hello World !

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

int main()
{
   printf("Hello world!\n");
   return EXIT_SUCCESS;
}
```

```
    .file "base.c"
    .section .rodata
.LC0:
    .string "Hello World!"
    .text
    .globl main
    .type main, @function
```

```
main:
.LFB0:
    .cfi_startproc
    pushl %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl %esp, %ebp
    .cfi_def_cfa_register 5
    andl $-16, %esp
    subl $16, %esp
    movl $.LC0, (%esp)
    call puts
    movl $0, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size main, .-main
    .ident "GCC"
```

Hello World avec lecture et écriture à l'écran

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

int main()
{
   char name[100];
   int  year;

   printf("Ton nom ? ");
   scanf("%s", name);
   printf("Tu es ne quand ? ");
   scanf("%d", &year);
   printf("%s, tu as %d ans.\n", name, 2013-year);

   return EXIT_SUCCESS;
}
```

L'exécution de *myprog* donne :

```
$ ./myprog
Ton nom ? Jean-Claude
Tu es ne quand ? 1816
Jean-Claude, tu as 197 ans.
```

Hello World avec paramètres de la ligne de commandes

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

int main(int argc, char *argv[])
{
   printf("%s lance avec %d arguments :\n",
          argv[0], argc);

   for (int i=0 ; i<argc ; i++){
      printf("{%d: \"%s\"} ",
      i, argv[i]);
   }
   printf("\n");
   return EXIT_SUCCESS;
}
```

Voici un exemple d'exécution :

```
$ ./test je suis un chat
./test lance avec 5 arguments :
{0: "./test"} {1: "je"} {2: "suis"} {3: "un"} {4: "chat"}
```

La fonction *printf*

Cette fonction permet d'afficher d'écrire des messages *formatés* sur la sortie standard, i.e. l'écran. Elle est déclarée dans le *header* **stdio.h** de la bibliothèque standard de C.

Son prototype est le suivant :

```
#include <stdio.h>
printf(const char *string, ...);
```

Voici un exemple :

```
printf("Bonjour %s, tu as %d ans.\n", "E.T.", 2013-1982);
```

Et son exécution :

```
Bonjour E.T., tu as 31 ans.
```

La fonction *printf*

Séquences d'échappement

- $\backslash n$: retour à la ligne,
- $\backslash t$: tabulation,
- $\backslash r$: retour chariot,
- etc.

Indicateurs de format

- $\%d$: entier signé,
- $\%c$: caractère,
- $\%s$: un chaîne de caractères,
- $\%f$: pour un nombre à virgule flottante,
- $\%x$: pour un affichage hexadécimal d'un nombre,
- etc.

N'hésitez pas à consulter la *page man* pour plus d'informations !

La fonction *printf*

Exercice

Écrivez un programme qui affiche la table de multiplication sous forme d'un tableau. Les paramètres (nombre de lignes et nombre de colonnes) seront passés grâce à la ligne de commande.

```
$ ./mult 3 5
      1   2   3   4   5
    ___ ___ ___ ___ ___
1 :   1   2   3   4   5
2 :   2   4   6   8  10
3 :   3   6   9  12  15
```

Aide

Vous aurez besoin pour cela, de la fonction *atoi* qui convertit une chaîne de caractères en un entier :

```
int entier = atoi("1943");
```

Bravo !

Vous savez maintenant

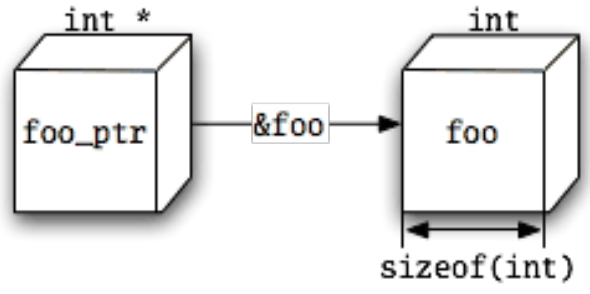
- écrire un programme minimal,
- utiliser les paramètres de la ligne de commande,
- afficher des messages à l'écran.

Les pointeurs

Définition

Un **pointeur** est une variable contenant une adresse. Il *pointe* généralement vers une case mémoire pouvant contenir une valeur.

```
1  int foo;
2  int *foo_ptr;
3  foo_ptr = &foo;
4  int foo_copy;
5  foo_copy = *foo_ptr;
```



Remarque

Un pointeur non initialisé peut pointer sur n'importe quelle adresse !

Pointeurs et paramètres

Le passage de paramètres à une fonction ne se fait que par **valeur**. Il est donc nécessaire d'utiliser des pointeurs si l'on souhaite modifier le contenu de certaines variables dans des sous-fonctions.

```c
void add(int a, int b, int c)
{
  a = b + c;
  printf("a = %d\n", a);
}

int main()
{
  int a = 0;
  add(a, 1, 2);
  printf("a = %d\n", a);
  return EXIT_SUCCESS;
}
```

```
a = 3
a = 0
```

```c
void add(int *a, int b, int c)
{
  *a = b + c;
  printf("a = %d\n", *a);
}

int main()
{
  int a = 0;
  add(&a, 1, 2);
  printf("a = %d\n", a);
  return EXIT_SUCCESS;
}
```

```
a = 3
a = 3
```

Les pointeurs

Exercice

Écrivez un programme qui :

- requiert en ligne de commande 3 nombres,
- appelle une fonction qui ordonne ces trois nombres,
- les affiche dans l'ordre croissant.

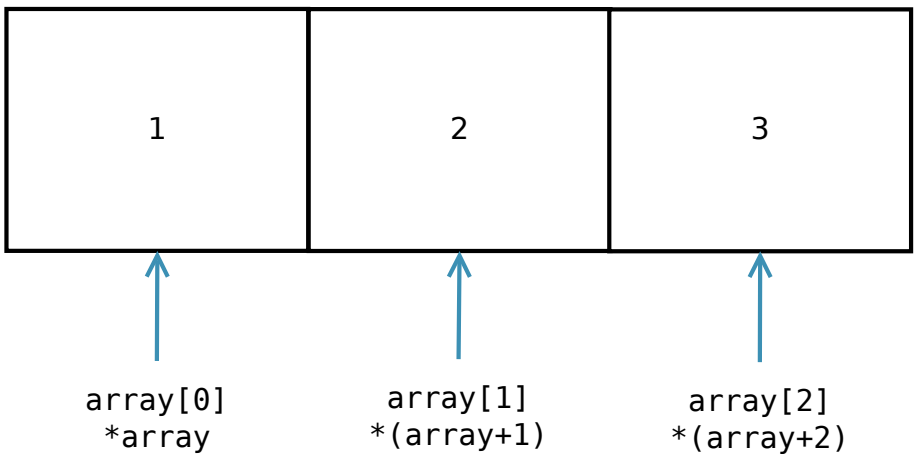
```
$ ./tri 7 1 5
Suite finale : 1 5 7
```

Pointeurs et tableaux

Définition

Un **tableau** définit par *type name[n]* est un pointeur nommé *name* pointant sur un espace mémoire de *n* éléments de type *type*.

```
int array[3] = {1,2,3};
```




```
int array[3];


array[2] = 42;

// est identique a

*(array + 2) = 42
```

Pointeurs et tableaux

Ces 3 versions sont-elles identiques ?

```
int main()
{
  int tab[10];
  for (int i = 0 ; i < 10 ; i++){
```

1)
```
tab[i] = i;
```

2)
```
*(tab+i)= i;
```

3)
```
i[tab] = i;
```

```
  }
  return EXIT_SUCCESS;
}
```

Les chaînes de caractères

Définition

Une **chaîne de caractères** est un tableau contenant des caractères et finissant par le caractère invisible '\0'.

On peut la parcourir par index :

```
1 char message [] = "Hello, world!";
2 for(int i = 0 ; message[i] != '\0' ; i++){
3   printf("%c", message[i]);
4 }
```

Ou encore par pointeur :

```
1 char message [] = "Hello, world!";
2 for(char *ptr = message ; *ptr != '\0' ; ptr++){
3   printf("%c", *ptr);
4 }
```

Les chaînes de caractères

Remarque

Une chaîne de caractères peut être déclarée de plusieurs façons.

```
char s[4];
s[0] = 'a';
s[1] = 'b';
s[2] = 'c';
s[3] = '\0';
```

```
char s[] = {'a','b','c','\0'};
s[1] = 'z';
```

```
char s[] = "abc";
s[1] = 'z';
```

```
char *s = "abc";
s[1] = 'z'; // --> SEG FAULT
```

Remarque

Quand un pointeur est déclaré avec une valeur initiale statique (chaîne de caractères entre guillemets), cette chaîne est une constante située dans une partie de la mémoire qui est accessible en lecture seulement.

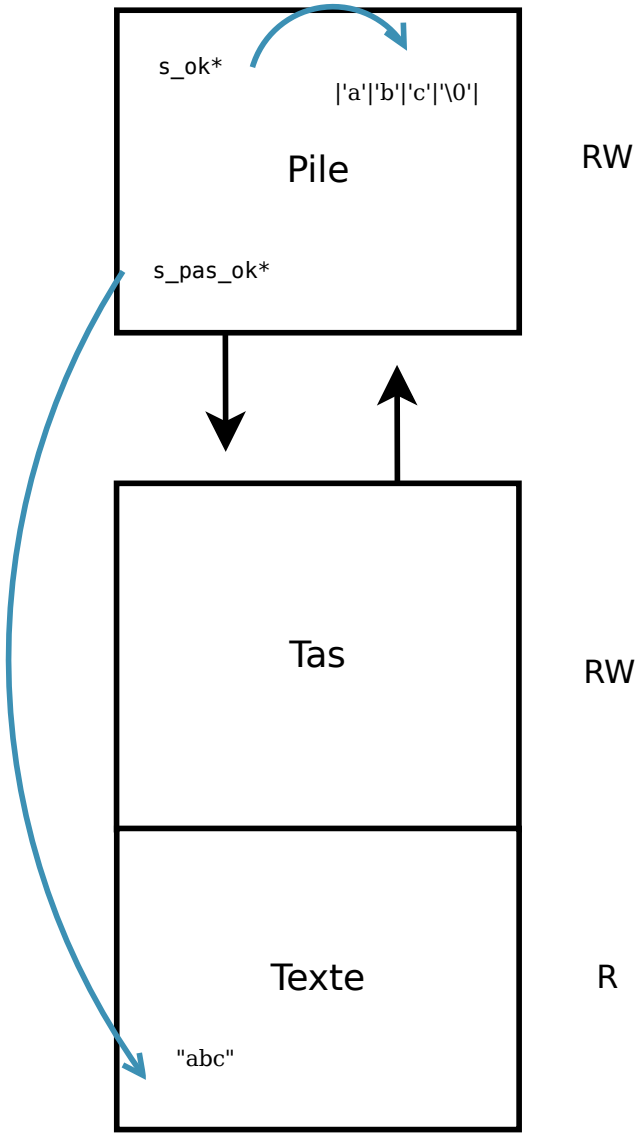
Les chaînes de caractères

```
char s_ok[4];
s[0] = 'a';
s[1] = 'b';
s[2] = 'c';
s[3] = '\0';
```

```
char s_ok[] = {'a','b','c','\0'};
```

```
char s_ok[] = "abc";
```

```
char *s_pas_ok = "abc";
```



Les chaînes de caractères

La plupart des fonctions relatives aux chaînes de caractères sont définies dans **string.h**.

```
1 #include <string.h>
2
3 size_t strlen(const char *s);
4 char  *strcpy(char *dest, const char *src);
5 char  *strcat(char *dest, const char *src);
6 int    strcmp(const char *s1, const char *s2);
7 ...
```

Il vaut néanmoins mieux utiliser les variantes suivantes qui évitent les débordements de mémoire en l'absence de '\0' en fin de chaîne.

```
1 char *strncpy(char *dest, const char *src, size_t n);
2 char *strncat(char *dest, const char *src, size_t n);
3 int   strncmp(const char *s1, const char *s2, size_t n);
```

Les chaînes de caractères

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

int main(int argc, char *argv[])
{
   if (argc != 2){
      printf("Usage: %s <lang>\n", argv[0]);
      return EXIT_FAILURE;
   }

   size_t size = 20;
   char msg[size];

   for(int i = 0 ; i < size ; i++)
      msg[i] = '\0';

   if (strcmp(argv[1], "fr") == 0){
      const char *mess = "Bonjour";
      strncpy(msg, mess, size-1);
   }
   else if (strcmp(argv[1], "en") == 0){
      const char *mess = "Hello";
      strncpy(msg, mess, size-1);
   }
   else{
      const char *mess = "Ich verstehe nicht alles";
      strncpy(msg, mess, size-1);
   }
   printf("%s\n", msg);
   return EXIT_SUCCESS;
}
```

Les chaînes de caractères

L'exécution du programme précédent donne :

```
$ ./lang
Usage: ./lang <lang>
$ ./lang fr
Bonjour
$ ./lang en
Hello
$ ./lang rr
Ich verstehe nicht
```

Les chaînes de caractères

Il existe également des opérations sur les caractères eux-mêmes définies dans **ctype.h**.

```
1  #include <ctype.h>
2
3  int isalpha(int c);
4  int isdigit(int c);
5  int isalnum(int c);
6  int isspace(int c);
7  int islower(int c);
8  int isupper(int c);
9  int tolower(int c);
10 int toupper(int c);
11 ...
```

En voici un exemple :

```
1  int c = 0;
2  char *str = "Je suis un chat";
3  for(char *ptr = str ; *ptr != '\0' ; ptr++){
4    if (isspace(*ptr))
5      c++;
6  printf("There are %d spaces in the string.", c);
```

La fonction *scanf*

La fonction **scanf** permet de lire des messages formatés sur l'entrée standard, i.e. le clavier.

Son prototype est le suivant :

```
scanf(const char *string, void *arg1, void *arg2, ...);
```

Voici un exemple :

```
1 char string[100];
2 int val = 0;
3 printf("-> ");
4 scanf("%s = %d", string, &val);
5 printf("str:%s -> val:%d\n", string, val);
```

Et son exécution :

```
-> plop = 12
str:plop -> val:12
```

Les chaînes de caractères

Exercice

Écrivez un programme qui lit au clavier des chaînes de caractères une par une et qui opère *minuscules* $\leftrightarrows$ *majuscules* en fonction du premier argument passé au programme. Le programme s'arrête quand l'utilisateur entre le mot clé **exit**.

```
$ ./to lower
-> toto
toto
-> tAtA
tata
-> exit
$ ./to upper
-> toto
TOTO
-> exit
```

Bravo !

Vous savez maintenant

- utiliser les pointeurs,
- utiliser les tableaux,
- utiliser les chaînes de caractères,
- lire des messages au clavier.