Le langage C

Naïm Qachri, Stéphane Fernandes Medeiros, Eythan Levy, Cédric Meuter

Université Libre de Bruxelles

2010/2011

Introduction

- Syntaxe du C **proche** de celle du **C++**
- Quelques **grosses différences**:
  - Pas de classes
  - Pas de *new* et *delete*
  - Pas de *cin»* et *cout«*
  - Pas de type *bool*
  - Librairies standards différentes
  - Autres différences...

Les différences de base

- Déclarations de variables
  - Seulement en début de fonction
  - Avant toute instruction
  - *for(int i=0;i<n;++i)* INTERDIT !!!
- **Pas de type booléens**: *0* pour faux, *1* pour vrai
- **Pas** de passage de paramètres **par référence**
- Utiliser des pointeurs.
- Exemple: `swap.c`:

```
void swap(int *i, int *j) {
    int t;

    t = *i;
   *i = *j;
   *j = t;
}
```

Les struct

- **Pas de classes !** Utiliser des `struct`
- `struct` est type composite très simple:
  - Pas de *constructeurs/destructeurs*.
  - **Pas** de **fonctions** membres.
  - Pas de mot-clés de visibilité (*private/public*).
- **Syntaxe:**
  - `struct` nom {données} liste-variables
- Souvent combiné avec `typedef`
- Accès aux variables: comme en C++ (“.” et “->”)

Exemple: *struct.c (1)*

```
struct {
  int a;
  char b;
  float c;
} x1;

struct {
  int a;
  char b;
  float c;
} y1[20], *z1;

/* Utilisation avec nom */

struct SIMPLE {
  int a;
  char b;
  float c;
};

struct SIMPLE x2; /* déclaration */
struct SIMPLE y2[20], *z2; /* ATTN: répétition de "struct" */
```

Exemple: *struct.c (1/1)*

```c
/* Utilisation avec typedef*/

typedef struct {
  int a;
  char b;
  float c;
} Simple;

Simple x3; /* déclaration */
Simple y3[20], *z3; /* Plus besoin de répéter "struct" */

/* Initialisation des struct */

struct INIT_EX {
  int a;
  short b[10];
  int c;
} x4 = { 10, {1,2,3,4,5}, 12 };
```

La fonction *main()*

- ***Type de retour:***
  - `int main(void)`
  - La librairie `stdlib.h` permet de renvoyer `EXIT_SUCCESS` ou `EXIT_FAILURE` à l'OS.

- ***Passage de paramètres:***
  - `int main(int argc, char**argv)`
  - `argc` est ***le nombre d'arguments entrés*** en ligne de commande
  - `argv` est un vecteur de strings (*1* par paramètre).
  - `argv[0]` est toujours le nom du programme.

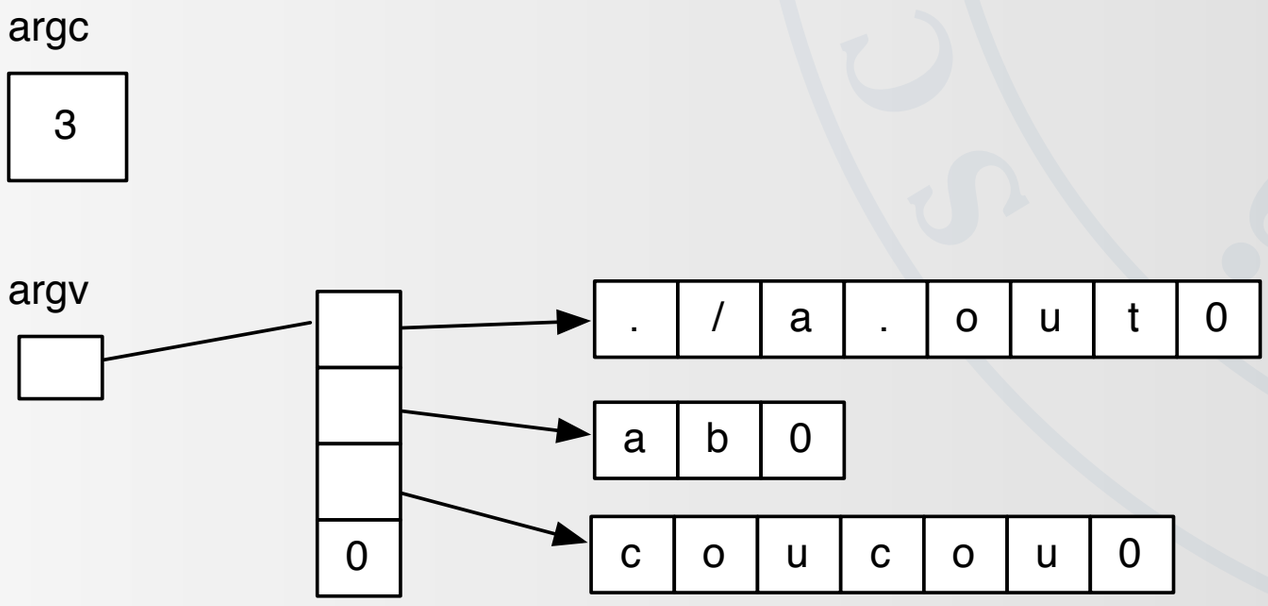
Exemple: *main.c (I)*

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

int main(int argc, char** argv) {
  int i;

  for(i = 0; i < argc;++i)
     printf("%s\n", argv[i]);

  return EXIT_SUCCESS;
}
```

Exemple: *main.c (II)*

- Utilisation:

```
[elevy@litpc49 tp1]$ gcc main.c
[elevy@litpc49 tp1]$ ./a.out  ab  coucou
./a.out
ab
coucou
```



L'allocation dynamique de mémoire

- Pas de `new` ni de `delete`.
- Utiliser (dans stdlib.h):
  - *`void* malloc(size_t size)`*
  - *`void free(void* pointer)`*
- **`malloc`**:
  - Reçoit le nombre de *bytes voulus*
  - Renvoie un bloc contigu de mémoire ou `NULL`.
- Renvoie un `void*`, quel que soit l'usage futur de la mémoire. Ce pointeur peut être converti en ce qu'on veut.
- **`free`** permet de rendre la mémoire à l'OS.

Exemple: *malloc.c*

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

int main() {
  int i, *pi;

  pi = (int*) malloc(25*sizeof(int));

  if(pi==NULL)
    printf("Pas assez de memoire\n");
  else {
    for(i = 0; i < 25; ++i) /* Accès classique à la mémoire */
      pi[i] = 0;
    free(pi);
  }
  return EXIT_SUCCESS;
}
```

Les entrées-sorties standards en C (I)

- Utilisation de `stdio.h`
- Concept de ***flux*** (stream) de *bytes entrant et sortant* d'un programme.
- Chaque programme a automatiquement les flux: `stdin`, `stdout`, et `stderr` ouverts à son lancement.
- Accès à un flux par une structure de type `FILE`.
- Vue générale des I/O de fichiers:
  - Déclaration d'un pointeur `FILE*` par ouverture.
  - Ouverture par `fopen`.
  - Lectures et/ou écritures.
  - Fermeture par `fclose`.

Les entrées-sorties standards en C (II)

- Ouverture de flux:
  - `FILE* fopen(const char* file, const char* mode)`
  - Modes principaux: r (lecture), w (écriture), a (append), r+ (lecture/écriture), w+ (lecture/écriture), a+ (lecture/append).
  - Renvoie NULL si erreur.
  - Taper `man fopen` pour plus de détails.
  - Exemple:

```
FILE* input;

input = fopen("data.txt", "r");

if (input == NULL)
     printf("erreur fichier");
```

Les entrées-sorties standards en C (III)

- ***Lecture/écriture:***
  - `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
  - `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `fread` lit *nmemb* éléments de données, ayant chacun une longueur *size*, dans le flux *stream*, et les place en mémoire à l'adresse *ptr*.
- `fwrite` écrit *nmemb* éléments de données, ayant chacun une longueur *size*, dans le flux *stream*, éléments obtenus à l'adresse *ptr*.
- `fread` et `fwrite` renvoient le nombre d'éléments lus ou écrits.

Les entrées-sorties standards en C (IV)

- ***Fermeture d'un flux::***
  - `int fclose(FILE *stream);`
- ***Gestion des erreurs:***
  - `int feof(FILE *stream);`
  - `int ferror(FILE *stream);`
  - `void clearerr(FILE *stream);`
  - `feof` teste si l'indicateur de **fin de fichier** est activé, sur le flux *stream* et renvoie □□□ □ si c'est le cas.
  - `ferror` teste l'indicateur d'erreur du flux (permet de tester si une erreur de lecture/écriture a eu lieu).
  - `clearerr` réinitialise les indicateurs.

Les entrées-sorties standards en C (V)

- ***Vidage d'un buffer:***
  - `int fflush(FILE *stream);`
  - Les écritures sur les flux sont *bufferisées*.
  - Utile pour le ***déboggage***.

- ***Positionnement sur un flux:***
  - `int fseek(FILE *stream, long offset, int whence);`
  - Permet de **mettre à jour l'indicateur de position courante** sur le fichier.
  - la nouvelle position est la position donnée par le paramètre `whence` + `offset` bytes.
  - Valeurs symboliques pour `whence`: `SEEK_SET` (début de fichier), `SEEK_CUR` (position courante), `SEEK_END` (fin de fichier)

Les entrées-sorties standards en C (VI)

- ***Lecture de caractères:***
  - ```
    int fgetc(FILE *stream);
    int getc(FILE *stream);
    int getchar(void);
    ```
  - `fgetc()` lit le prochain caractère du flux, et le renvoie en tant que `unsigned char` converti en `int`, ou `EOF` en cas d'erreur ou de fin de fichier
  - `getc()` est une variante de `fgetc()`
  - `getchar()` est équivalent à `getc(stdin)`
    ```

Les entrées-sorties standards en C (VII)

- ***Ecriture de caractères:***
  - ```
    int fputc(int c, FILE *stream);
    int fputs(const char *s, FILE *stream);
    int putc(int c, FILE *stream);
    int putchar(int c);
    int puts(const char *s);
    ```
- `fputc()` et `putc()` écrivent un caractère sur le flux donné en paramètre.
- `fputs()` **écrit un string** sur le flux donné en paramètre.
- `putchar()` et `puts()` écrivent un caractère ou un string sur l'output standard.
    ```

Les entrées-sorties standards en C (VIII)

- ***Ecriture de caractères:***
  - `fputc()`, `putc()` and `putchar()` renvoient le caractère en tant que `unsigned char` converti en `int`, ou `EOF` en cas d'erreur.
  - `puts()` et `fputs()` renvoient un entier non-négatif en cas de succès et `EOF` en cas d'erreur.

Les entrées-sorties standards en C (IX)

- ***Lecture/écriture de strings:***
  - `char *fgets(char *s, int size, FILE *stream);`
  - `char *gets(char *s);`
  - `fgets()` lit au plus `size -1` caractères du flux et les sauve dans la zone-mémoire pointée par `s` (supposée déjà allouée et assez grande). La lecture s'arrête après un `EOF` ou une fin de ligne (auquel cas, la fin de ligne est incluse dans la zone-mémoire). Un `'\0'` est placé à la fin dans le buffer.
  - `gets()` lit une ligne de `stdin` dans la zone mémoire pointée par `s` jusqu'à une fin de ligne ou un `EOF`, qui sont remplacés par `'\0'`.

Les entrées-sorties standards en C (X)

- ***Les sorties formatées:***
  - `int printf(const char *format, ...);`
  - `int fprintf(FILE *stream, const char *format, ...);`
  - `int sprintf(char *str, const char *format, ...);`
  - `printf` écrit sur `stdout`, `fprintf` sur le flux `stream`, et `sprintf` sur le string `str`.
  - Fonctions à nombre variable de paramètres.
  - Renvoient le nombre de caractères écrits.
  - Produisent un output en fonction d'un "format".
  - Le "format" est un string composé de caractères ordinaires (copiés sur le stream d'output) et des spécifications de conversion spécifiant le mode d'affichage de chaque argument additionnel.

Les entrées-sorties standards en C (XI)

- **Exemples:**

```
printf("blabla\n");
printf("Quelques nombres: %d, %d, et %d\n", 1, 2, 3);
```

- Le second exemple produit en output:

```
Quelques nombres: 1, 2, et 3
```

- Autres spécifications de conversion:

%d,%i	entiers décimaux
%o, %x	octal/ hexadécimal
%c	caractère
%s	string
%f	flottant *(ou float)*
%g	double

Les entrées-sorties standards en C (XII)

- ***Les entrées formatées:***
  - ```
    int scanf(const char *format, ...);
    int fscanf(FILE *stream, const char *format, ...);
    int sscanf(const char *str, const char *format, ...);
    ```

- Lisent les éléments d'un flux selon un format, et mettent les valeurs dans des variables aux adresses transmises en paramètres sous forme de pointeurs.

- ***Exemple 1:***
  - ```
    int x, y, z;
    scanf("%d, %d, %d", &x,&y,&z);
    ```
  - Lit 3 entiers séparés par des virgules.

Exemple: *copy_std.c*

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

int main() {
  int c;
  FILE *in, *out;

  in = fopen("file.in", "r");
  out = fopen("file.out", "w");

  while( (c=fgetc(in)) != EOF  )
    fputc(c, out);

  fclose(in);
  fclose(out);

  return EXIT_SUCCESS;
}
```

Les entrées-sorties standards en C (XIII)

- ***Exemple 2:***
  - ```
    char s[256];
    int n;
    float f;
    char c;

    scanf(“Bonjour %d, %f, %c, %s", &n, &f, &c, s);
    ```
  - Force le string à commencer par “Bonjour”.
- Renvoie le nombre d’éléments correctement lus, ou `EOF`.
    ```

Les entrées-sorties standards Unix(I)

- Fonctions plus ***"bas-niveau"***, standard sur toutes les versions de UNIX, mais ne faisant pas partie de la bibliothèque standard C.

- La notion de `FILE*` est remplacée par celle de *file descriptor*, un entier obtenu à l'ouverture d'un fichier et utilisé pour lire et écrire dessus.

- Les descripteurs 0, 1, et 2, sont utilisés pour, respectivement, les entrées, sorties, et erreurs standards.

Les entrées-sorties standards Unix(II)

- ***Ouverture:***
  - ```
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    int open(const char *pathname, int flags);
    int open(const char *pathname, int flags, mode_t mode);
    ```

- Reçoit un nom de fichier et renvoie un file descriptor (ou `-1` en cas d'erreur).
- Le champ `flags` contient `O_RDONLY`, `O_WRONLY` ou `O_RDWR`, éventuellement combiné à d'autres flags par un "`|`".
- L'argument `mode` est utilisé pour initialiser les permissions en cas de création d'un nouveau fichier.
    ```

Les entrées-sorties standards Unix(III)

- ***Lecture:***
- `#include <unistd.h>`
  `ssize_t read(int fd, void *buf, size_t count);`
- Essaie de lire jusque `count` bytes du file descriptor `fd` dans la zone-mémoire commençant à l'adresse `buf`.
- Avance le pointeur de lecture/écriture.
- Renvoie le nombre de *bytes* lus ou -1 si erreur.

Les entrées-sorties standards Unix(IV)

- ***Ecriture:***
  - ```
    #include <unistd.h>
    ssize_t write(int fd, const void *buf, size_t count);
    ```
  - Ecrit jusque `count` bytes sur le fichier référencé par le file descriptor `fd`, à par tir de la zone-mémoire commençant à l'adresse `buf`.
  - Avance le pointeur de lecture/écriture.
  - Renvoie le nombre de bytes écrits ou -1 si erreur.
- ***Fermeture:***
  - ```
    #include <unistd.h>
    int close(int fd);
    ```
  - Libère le file descriptor.
  - Renvoie 0, ou -1 si erreur.

Les entrées-sorties standards Unix(V)

- ***Positionnement:***

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

- Fonctionnement comparable à `fseek`

Exemple: *copy_uni.c*

```c
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>

int main() {
  char c;
  int in, out;
  in = open("file.in", O_RDONLY);
  out = open("file.out", O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);

  while(read(in,&c,1) == 1)
    write(out, &c, 1);

  close(in);
  close(out);

  return EXIT_SUCCESS;
}
```