

INF3135

Construction et maintenance de logiciels

Cours 6: Entrées et sorties

Alexandre Blondin Massé

Université du Québec à Montréal
Département d'informatique

Été 2020

Table des matières

1 Résumé du chapitre 3

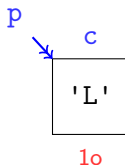
2 Travail pratique 1

3 Résumé du chapitre 4

Résumé du chapitre 3

Adresse et pointeur

- **Adresse**: indique emplacement d'une donnée
- **Opérateur &**: retourne l'adresse d'une *left-value*
- **Pointeur**: *left-value* qui contient une adresse
- Pointeurs **typés**: `int*`, `double**`, `struct point*`, etc.
- Plusieurs **types**: nul, constant (`const`), générique (`void*`), de fonction



Deux « sortes » de pointeurs constants

- `const <type> *p`: pointeur en lecture seule
- `<type> *const q`: pointeur constant

Opérations sur les pointeurs

Déférencement

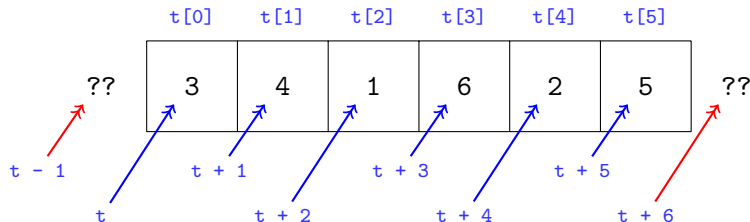
- *: accès à la donnée « pointée »
- ->: déréférencement du champ d'une structure

Autres opérations

- =: affectation
- (type*): conversion
- ==/!=: égalité et différences d'adresses
- <=, >=, <, >: comparaison d'adresses
- +/-: retourne un pointeur décalé
- ++/--: incrémentation et décrémentation

Tableaux et arithmétique des pointeurs

```
int t[] = {3, 4, 1, 6, 2, 5};
```



- On a `t + i == &t[i]`
- De façon équivalente, `*(t + i) == t[i]`

Chaînes de caractères

```
char s1[10];      // Tableau de caractères de taille fixe
char *s2;         // Pointeur vers début d'une chaîne
const char *s2;   // Chaîne en lecture seule
```

Bibliothèque `string.h`:

- `strcat`: concatène une chaîne à la suite d'une autre
- `strncat`: concatène une chaîne à une autre en tronquant
- `strcpy`: copie une chaîne dans une autre
- `strncpy`: copie une chaîne dans une autre en tronquant
- `strlen`: longueur d'une chaîne
- `strcmp`: compare deux chaînes
- `strncmp`: compare deux chaînes en tronquant
- `strchr`: cherche un caractère dans une chaîne de gauche à droite
- `strrchr`: cherche un caractère dans une chaîne de droite à gauche
- `strstr`: cherche une chaîne dans une autre de gauche à droite
- `strrstr`: cherche une chaîne dans une autre de droite à gauche
- `strtok`: segmente une chaîne en morceaux (*tokens*)

Pointeurs de fonctions

```
// Pointeur de fonction de type int -> int
int (*f)(int x);
// Pointeur de fonction de type (int, int) -> int
int (*g)(int x, int y);
```

Bibliothèque stdlib.h:

```
// Trie les valeurs d'un tableau
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));

// Effectue une fouille binaire dans un tableau
void *bsearch(const void *key,
              const void *base,
              size_t nmemb,
              size_t size,
              int (*compar)(const void *, const void *));
```

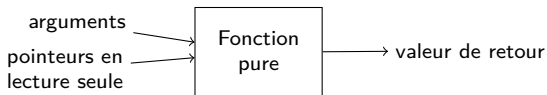

Travail pratique 1

Travail pratique 1

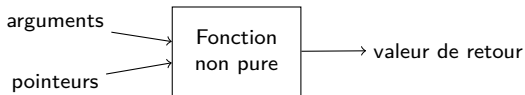
- Date de remise: **19 juin**, à **23h59**
- **20%** de la note totale
- Doit être fait **seul**
- **Dépôt GitLab**: doit être *forké*, sa visibilité mise à *privé*, puis l'accès en mode *Developer* doit être donné à blondin_al
- **Description du travail**: dans le fichier `sujet.md`
- À **compléter**: `canvascii.c`, `Makefile`, `.gitignore`, `README.md`
- À **modifier**: dans `check.bats`, supprimer `skip` pour activer le test

Résumé du chapitre 4

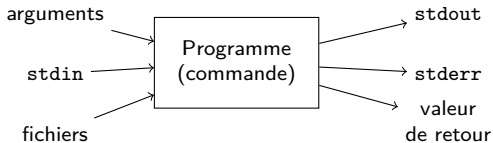
Entrées et sorties



Entrées



Sorties



La bibliothèque `stdio.h`

- *stdio* = *standard input output*
- Inclusion avec `#include <stdio.h>`

Macros:

- EOF: caractère de fin de fichier
- `stdin`, `stdout`, `stderr`: canaux standards
- `NULL`: pointeur nul, ...

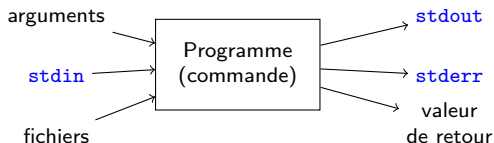
Types:

- `FILE`: flux (*stream*)
- `size_t`: taille en octets
- `fpos_t`: position dans un flux, ...

Variables externes: `optarg`, `opterr`, `optind`, `optopt` (gestion des arguments)

Plusieurs dizaines de **fonctions** (`man stdio`)

Canaux



3 canaux standards

- `stdin`: entrée standard (canal 0)
- `stdout`: sortie standard (canal 1)
- `stderr`: sortie d'erreur standard (canal 2)

Comportement par défaut (peut être redéfini)

- `stdin`: saisie clavier (*line buffered*)
- `stdout`: affichage sur le terminal (*line buffered*)
- `stderr`: affichage sur le terminal (*unbuffered*)

Redirections (1/2)

- Par défaut, `stdin` lit la saisie clavier
- Et `stdout`/`stderr` écrivent sur le terminal
- Ces comportements peuvent être modifiés avec des **redirections**
- Les redirections sont gérées par le **shell**
- Elles ne sont donc **pas gérées** par `argc` et `argv`

Syntaxe

- `commande < fichier`: redirige fichier sur `stdin`
- `commande > fichier`: redirige `stdout` dans fichier
- `commande 2> fichier`: redirige `stderr` dans fichier

Redirections (2/2)

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

```
int main(void) {  
    char c = getchar();  
    if (c == 'y')  
        printf("Yes!\n");  
    else  
        fprintf(stderr, "No!\n");  
    return 0;  
}
```

```
$ cat redirections.y
```

```
yoyo
```

```
yaourt
```

```
$ cat redirections.z
```

```
zèbre
```

```
zoo
```

```
$ gcc redirections.c -o redirections
```

```
$ ./redirections < redirections.y
```

```
Yes!
```

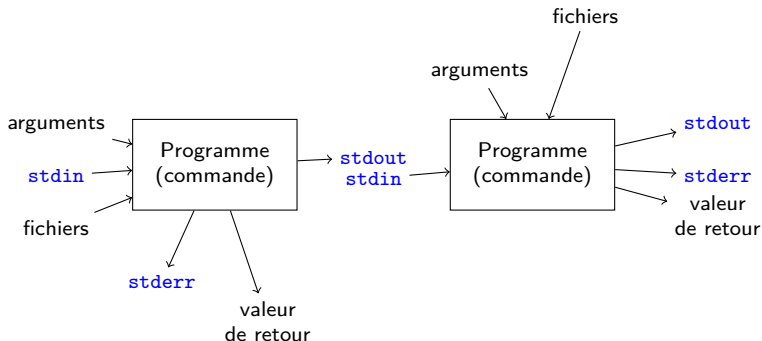
```
$ ./redirections < redirections.z
```

```
No!
```

```
$ ./redirections < redirections.z 2> /dev/null
```


Tubes

- Permet d'**enchaîner** des programmes
- Le contenu écrit sur stdout par la première commande
- Est lu sur stdin par la deuxième commande
- **Syntaxe:** commande1 | commande2 | ... | commandeN



Filtres utiles

Filtre: programme souvent utilisé dans un tube

- `sort`: trie les lignes d'un flux
- `uniq`: supprime les doublons consécutifs
- `grep`: filtre selon une expression régulière
- `fmt`: formate des données
- `pr`: formate du texte pour impression
- `head`: affiche les premières lignes d'un flux
- `tail`: affiche les dernières lignes d'un flux
- `tr`: traduit caractère par caractère
- `sed`: transforme du texte
- `awk`: transforme du texte

Plus de détails dans **INF1070**

Consulter le manuel (`man`)

Exemple de filtres

Fichier maj.c:

```
#include <stdio.h>
#include <ctype.h>

int main(void) {
    char c;
    while ((c = getchar()) != EOF) {
        putchar(toupper(c));
    }
    return 0;
}
```

```
$ gcc maj.c -o maj
$ head -n 2 maj.c | ./maj
#include <STDIO.H>
#include <CTYPE.H>
$ head -n 2 maj.c | ./maj | tail -n 1
#include <CTYPE.H>
$ grep 'char' maj.c | ./maj
CHAR C;
WHILE ((C = GETCHAR()) != EOF) {
    PUTCHAR(TOUPPER(C));
```

Valeur de retour

- En Unix, tout programme retourne une **valeur entière**
- Lorsque son exécution est **terminée**

Sémantique

- 0: le programme s'est terminé « **normalement** »
- $\neq 0$: le programme s'est terminé « **anormalement** »

Récupérer la valeur de retour

- Contenue dans la **variable spéciale** \$?
- Valeur de retour de la **dernière commande**

Valeur de retour en C

Fonction main

- Valeur retournée à l'aide de `return`
- Doit être **entière**
- Peut être **négative**
- Par défaut, retourne 0
- **Bonne pratique**: toujours spécifier la valeur de retour

La fonction exit

```
void exit(int status);
```

- Permet de **terminer** l'exécution du programme proprement
- Vide et ferme les **flux** encore ouverts
- Supprime les **fichiers temporaires**

Combinaisons de commandes

- On peut **combinaisonner** des commandes avec ; && et ||
- Le comportement dépend de la **valeur de retour**
- ;: deux commandes consécutives indépendantes
- &&: 2e commande exécutée seulement si la 1re réussit
- ||: 2e commande exécutée seulement si la 1re échoue

```
$ echo "commande" && echo $?
```

```
commande
```

```
0
```

```
$ echo "commande" || echo $?
```

```
commande
```

```
$ cat fichier.inexistant && echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
$ cat fichier.inexistant ; echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
1
```

```
$ cat fichier.inexistant || echo $?
```

```
cat: fichier.inexistant: No such file or directory
```

```
1
```