centralelille

AUDACE • EXIGENCE • RESPECT

Algorithmique Avancée & Programmation

Séance 1 : pointeurs, fonctions, chaînes de caractères

Bonjour !

- Présentation de l'intervenant, du responsable du groupe TP
- Consignes générales :
  - Utilisation des postes de la salle de TP, idéalement linux en natif
    - Editeur de texte sur les machines du B7 : featherpad
    - ou d'un PC personnel sous Linux
  - Pas de repl.it sauf besoin impérieux

Cadrage séance 1 : Pointeurs, fonctions, chaînes de caractères, fichiers

- Test introductif : codage d'un exercice et rendu sur moodle (source+makefile)
  - Surtout pratique
- Pointeurs
  - Organisation d'un programme : Tas / Pile
  - Tableaux & Arithmétique des pointeurs
  - Allocation de mémoire
    - Fonctions de stdlib.h
- Fonctions & Pointeurs
  - Passages de paramètres par valeur, par référence
  - Mot-clé const
- Chaînes de caractères
  - Fonctions de string.h
  - Paramètres en ligne de commande
- Fichiers

Test 1

- Sujet : partagé par l'intervenant
  - Développer un code source + le fichier makefile correspondant
- Durée : 15 min
  - Chaque minute de dépassement : -1 point
- Déposer fichier source + makefile sur moodle

Trame de séance

- Cf. Trame séance 1 - 2022-2023
- Cours : codes ascii, flux d'entrée-sortie
  - Exercice 1 : Comptage des mots et des caractères dans un texte
- Cours : pointeurs, chaînes de caractères
  - Exercice 2 : Chiffrage par substitution mono alphabétique
- TEA

Après cette séance et le TEA, vous devez savoir :

- Faire des dessins pour représenter l'organisation mémoire d'un programme utilisant de l'allocation de mémoire
- Comprendre et expliquer les arguments des fonctions des fichiers d'entête string.h et stdlib.h
- Ecrire un programme allouant de la mémoire
- Comprendre l'erreur "segmentation fault" et ses possibles causes
- Manipuler rigoureusement des variables de type pointeur : initialisation, test à NULL, test des valeurs de retour des appels système

Pédagogie

temps restant
dans la séance

temps max consacré
au sujet courant

- ***Exemple*** : code commenté immédiatement
  - Le code est déjà fourni intégralement sur moodle
- ***Exercice corrigé*** : code corrigé immédiatement en live-coding par l'intervenant avec l'aide des étudiants
  - Un squelette est fourni sur moodle pour gagner du temps
- ***Exercice*** : squelette fourni, à compléter par les étudiants pendant que l'enseignant passe dans les rangs
  - Sera corrigé par l'enseignant ou un étudiant en fonction de la vélocité du groupe et des questions posées

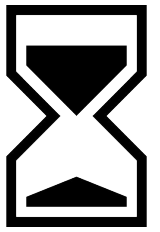
Raccourcis clavier LUbuntu

- Changer de bureau :
  - Ctrl+Alt+ flèches G/D
  - Chiffres 1/2/3/4 en bas à gauche de l'écran
- Ouvrir un terminal depuis l'explorateur de fichiers :
  - Outils -> ouvrir un terminal
  - Touche F4

centralelille

AUDACE • EXIGENCE • RESPECT

3h40



0 min

Rappels
Cf. Capsule 5

À ne pas présenter pendant la séance

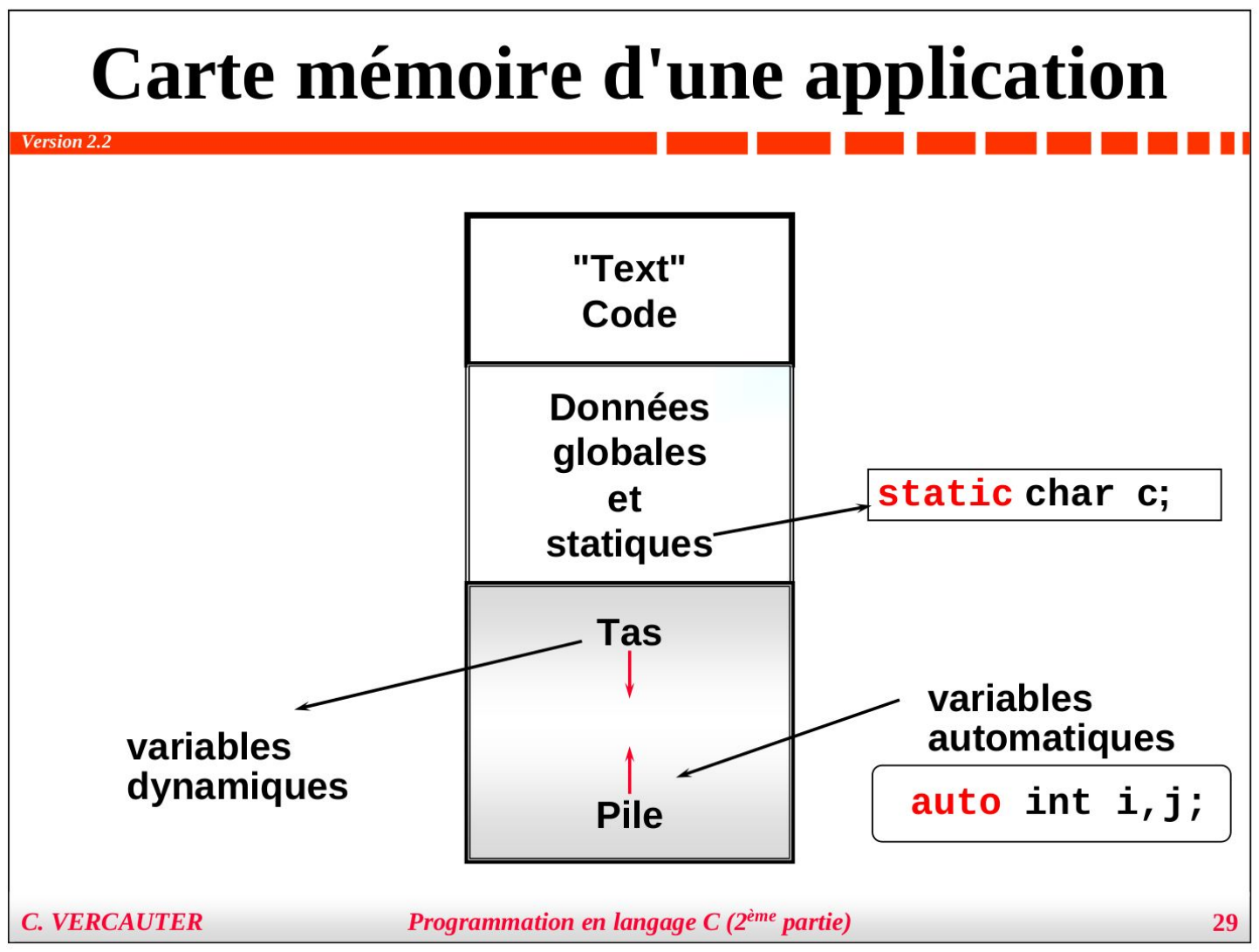
Référence pour les questions éventuelles

Tableaux

- Permet de regrouper des données de même type
- Possibilité de les initialiser entièrement au moment de leur déclaration
  - Sinon, il faut définir leur taille et les initialiser case par case
- Numérotation des cases à partir de 0

```
// déclarations et initialisations :
int tab1[] = {1,2,3};
int tab2[3] = {0};
// déclaration seule :
int tab3[3];
// affectation de la première case :
tab3[0] = 1;
tab3 = tab2; // interdit !
tab3 = {1,2,3}; // interdit !
```

Variables… Où sont elles ?



Pile & Tas

- Pile et variables globales/locales en C
  - Présentation de la pile (Stack)
  - https://youtu.be/geIng_jGQ3w
- Allocation dynamique et type opaque en C
  - Présentation du tas (Heap)
  - https://youtu.be/I2VU0RUWWXA

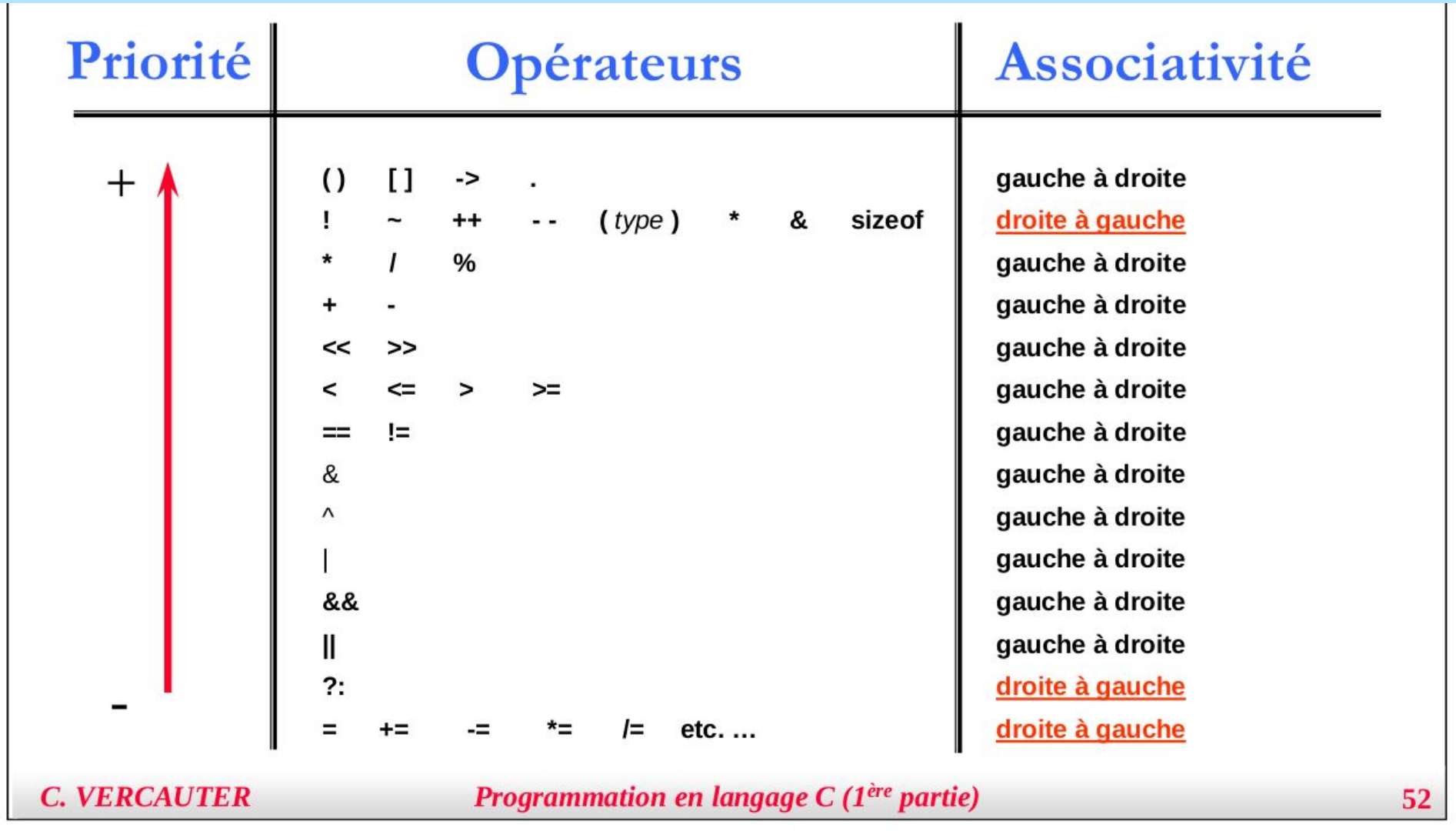
Passage de paramètres aux fonctions

- Les paramètres formels sont des variables locales
- L'appel de fonction utilise les paramètres effectifs
- Le passage de paramètre s'effectue toujours par valeur
  - Si on change la valeur du paramètre formel à l'intérieur de la fonction, cela n'a pas d'impact sur le paramètre effectif ayant servi à l'appel
- MAIS… parfois la valeur du paramètre correspond à l'adresse d'une autre variable !
  - Dans ce cas, on pourra modifier cette variable depuis la fonction !
  - On parle de passage par référence

Passage de paramètres de type structures ou tableaux

- Les tableaux sont passés aux fonctions par référence !
  - Ce que l'on passe, c'est l'adresse du tableau
  - On pourra donc modifier le tableau depuis la fonction !
- Les structures sont passées par valeur
  - On manipule une copie de la structure au sein de la fonction
  - On ne pourra pas modifier la structure ayant servi lors de l'appel depuis la fonction

Priorité et Associativité



Priorité	Opérateurs	Associativité
+	( ) [ ] -> .	gauche à droite
	! ~ ++ - - (*type*) * & sizeof	droite à gauche
	* / %	gauche à droite
	+ -	gauche à droite
	<< >>	gauche à droite
	< <= > >=	gauche à droite
	== !=	gauche à droite
	&	gauche à droite
	^	gauche à droite
	|	gauche à droite
	&&	gauche à droite
	||	gauche à droite
	?:	droite à gauche
-	= += -= *= /= etc. …	droite à gauche

Un opérateur a une associativité “de droite à gauche” quand a op b op c signifie a op (b op c)

Appels système : errno & perror()

- Les appels systèmes positionnent en cas d'erreur un code d'erreur dans la variable errno
  - Cf. fichier d'entête errno.h
- Ce code est associé à un message qu'il est possible d'afficher
- char * strerror(int code)
  - Affiche le message associé au code fourni
- void perror(const char *chaine)
  - Affiche le message associé à la valeur actuelle de errno, précédé de la chaîne fournie

Appels système : macros CHECK
Cf. include/check.h

- Tester les valeurs de retour des appels systèmes
  - Ils peuvent échouer !

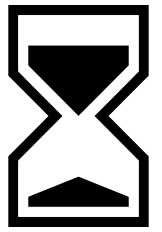
```
#define CHECK(sts, msg)                 \
    if (-1 == (sts) && errno != 0)      \
    {                                   \
        perror(msg);                    \
        exit(EXIT_FAILURE);             \
    } else
        \
    printf("%s : %d\n", msg, sts)
```

```
/* Lorsque stat vaut val alors l'opération dont le
résultat est stat, a échoué. Pour la plupart des
appels système Posix, la valeur signalant l'échec
est -1. Elle correspond à NULL lorsque l'appel
système retourne un pointeur.  */
#include <stdlib.h>
#include <stdio.h>
#ifndef DEBUG
#define CHECK_IF(stat, val, msg)      \
    if ( (stat) == (val) )            \
    {                                 \
        perror(msg);                  \
        exit(EXIT_FAILURE);           \
    }
#else
#define CHECK_IF(stat, val, msg)      \
    if ( (stat) == (val) )            \
    {                                 \
        perror(msg);                  \
        exit(EXIT_FAILURE);           \
    }                                 \
    else printf("%s ... OK\n", msg)
#endif
```

centralelille

AUDACE • EXIGENCE • RESPECT

3h40



10 min

Organisation des programmes

Arborescence : à placer dans un répertoire sans accents ni espaces !!

ex1/ : code de l'exemple/exercice 1

ex2/ : code de l'exemple/exercice 2

.../

include/ : fichiers d'entête utiles

makefile : fichier makefile principal

README : à lire

setup_env.sh : à exécuter au début de la séance

Production des exécutables : à exécuter depuis la racine du répertoire de la séance

- source setup_env.sh (à exécuter une seule fois)
  - Préparation de l'environnement
  - Pas obligatoire si cela pose problème sur votre machine
- make ex1
  - Production de l'exécutable ex1/ex1.exe
- ./ex1/ex1.exe
  - Exécution de l'exécutable ex1.exe

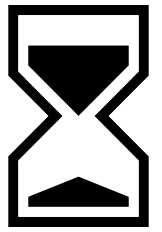
Traces d'exécution
Cf. include/traces.h

- Macro-fonctions utiles pour afficher des messages à l'utilisateur / mettre en pause le programme
- CLRSCR(); // Efface l'écran
- HR(); // Ligne horizontale
- NL(); // Saut de ligne
- ENTER2CONTINUE(); // Pause
- WHOAMI(); // Nom programme / date d'exécution
- TOUCH_HERE("msg"); // contexte + msg
  - **C'est ici que vous devez intervenir dans le code !!**
- ...

centralelille

AUDACE • EXIGENCE • RESPECT

3h30



10 min

Codes ASCII des caractères

ex1 : saisir ‘A’ de toutes les manières possibles

Représentation des caractères
Cf. exercice semaine 1

- Code ASCII *("American Standard Code for Information Interchange")* : norme de codage des caractères
- Type langage C : char
- Représentation : un octet en mémoire
- Interprétations :
  - Caractères affichables + caractères de contrôle
  - Entier signé (valeurs possibles entre -128 et 127)
- Symbole de format pour printf : %c

Exemple : **ex1** : Saisir le caractère 'A' de toutes les manières possibles

- https://fr.wikipedia.org/wiki/S%C3%A9quence_d%27%C3%A9chappement

```
#include <stdio.h>

int main(void) {
   printf("A = %c %d\n",'A','A');
   printf("A = %c %d\n",'\x41','\x41'); //hexadécimal
   printf("A = %c %d\n",'\101','\101'); //octal
   printf("A = %c %d\n",65,65);         //decimal
   printf("A = %c %d\n",0x41,0x41);     //hexadécimal
   printf("A = %c %d\n",0101,0101);     //octal
}
```

man ascii

ASCII(7) Manuel du programmeur Linux ASCII(7)

NOM

ascii - Jeu de caractères ASCII en octal, décimal, et hexadécimal

DESCRIPTION

ASCII est l'acronyme de « American Standard Code for Information Interchange ». Il s'agit d'un code sur 7 bits. De nombreux codes sur 8 bits (tels que l'ISO 8859-1) contiennent l'ASCII dans leur première moitié. L'équivalent international de l'ASCII est connu sous le nom de ISO 646.

La table suivante contient les 128 caractères ASCII.

Les séquences d'échappement '\X' pour les programmes C sont mentionnées.

Oct	Déc	Hex	Car.	Oct	Déc	Hex	Car.
000	0	00	NUL '\0'	100	64	40	@
001	1	01	DET (début d'en-tête)	101	65	41	A
002	2	02	DTX (début de texte)	102	66	42	B
003	3	03	FTX (fin de texte)	103	67	43	C
004	4	04	FTR (fin de transmission)	104	68	44	D
005	5	05	DEM (demande)	105	69	45	E
006	6	06	ACC (accusé de réception)	106	70	46	F
007	7	07	SON '\a' (sonnerie)	107	71	47	G
010	8	08	EFF '\b' (espace arrière)	110	72	48	H
011	9	09	TAB '\t' (tab. horizontale)	111	73	49	I
012	10	0A	PAL '\n' (changement ligne)	112	74	4A	J
013	11	0B	TAV '\v' (tab. verticale)	113	75	4B	K
014	12	0C	SDP '\f' (saut de page)	114	76	4C	L
015	13	0D	RC '\r' (retour chariot)	115	77	4D	M
016	14	0E	HC (hors code)	116	78	4E	N
017	15	0F	ES (en code)	117	79	4F	O
020	16	10	ÉCT (échap. transmission)	120	80	50	P
021	17	11	CD1 (commande dispositif 1)	121	81	51	Q
022	18	12	CD2 (commande dispositif 2)	122	82	52	R
023	19	13	CD3 (commande dispositif 3)	123	83	53	S
024	20	14	CD4 (commande dispositif 4)	124	84	54	T
025	21	15	ACN (accusé réception nég.)	125	85	55	U
026	22	16	SYN (synchronisation)	126	86	56	V
027	23	17	FBT (fin bloc transmission)	127	87	57	W
030	24	18	ANN (annulation)	130	88	58	X
031	25	19	FS (fin de support)	131	89	59	Y
032	26	1A	SUB (substitution)	132	90	5A	Z
033	27	1B	ÉCH (échappement)	133	91	5B	[
034	28	1C	SF (séparateur fichiers)	134	92	5C	\ '\\'
035	29	1D	SG (séparateur de groupes)	135	93	5D	]
036	30	1E	SA (sép. enregistrements)	136	94	5E	^
037	31	1F	SSA (sép. de sous-articles)	137	95	5F	_
040	32	20	ESP (espace)	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b

Tableaux

Pour plus de commodité, voici des tables plus compactes en hexadécimal et en décimal.

```
   2 3 4 5 6 7       30 40 50 60 70 80 90 100 110 120
  ------------      ----------------------------------
0:   0 @ P ` p     0:    (  2  <  F  P  Z  d   n   x
1: ! 1 A Q a q     1:    )  3  =  G  Q  [  e   o   y
2: " 2 B R b r     2:    *  4  >  H  R  \  f   p   z
3: # 3 C S c s     3: !  +  5  ?  I  S  ]  g   q   {
4: $ 4 D T d t     4: "  ,  6  @  J  T  ^  h   r   |
5: % 5 E U e u     5: #  -  7  A  K  U  _  i   s   }
6: & 6 F V f v     6: $  .  8  B  L  V  `  j   t   ~
7: ' 7 G W g w     7: %  /  9  C  M  W  a  k   u  SUP
8: ( 8 H X h x     8: &  0  :  D  N  X  b  l   v
9: ) 9 I Y i y     9: '  1  ;  E  O  Y  c  m   w
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o SUP
```

NOTES

Historique

Une page de manuel **ascii** est apparue dans AT&T UNIX version 7.

Sur les terminaux anciens, le code de soulignement (« underscore ») est affiché sous forme de flèche vers la gauche (« backarrow »), l'accent circonflexe (« caret ») est affiché sous forme de flèche vers le haut, et la barre verticale est interrompue en son centre.

Les caractères majuscules et minuscules ne diffèrent que d'un bit, et le caractère ASCII « 2 » ne diffère du guillemet que d'un bit (ils partagent la même touche sur un clavier QWERTY). Cela facilite l'encodage mécanique ou par un clavier sans micro-contrôleur, comme sur les anciens télétypes.

La norme ASCII a été publiée en 1968 par l'USASI (United States of America Standards Institute).

VOIR AUSSI

iso_8859-1(7), **iso_8859-10**(7), **iso_8859-13**(7), **iso_8859-14**(7), **iso_8859-15**(7), **iso_8859-16**(7), **iso_8859-2**(7), **iso_8859-3**(7), **iso_8859-4**(7), **iso_8859-5**(7), **iso_8859-6**(7), **iso_8859-7**(7), **iso_8859-8**(7), **iso_8859-9**(7), **utf-8**(7)

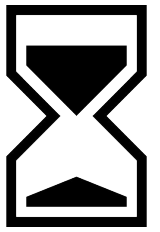
COLOPHON

Cette page fait partie de la publication 3.65 du projet man-pages Linux. Une description du projet et des instructions pour signaler des anomalies peuvent être trouvées à l'adresse http://www.kernel.org/doc/man-pages/.

centralelille

AUDACE • EXIGENCE • RESPECT

3h20



15 min

Entrées-Sorties (1)

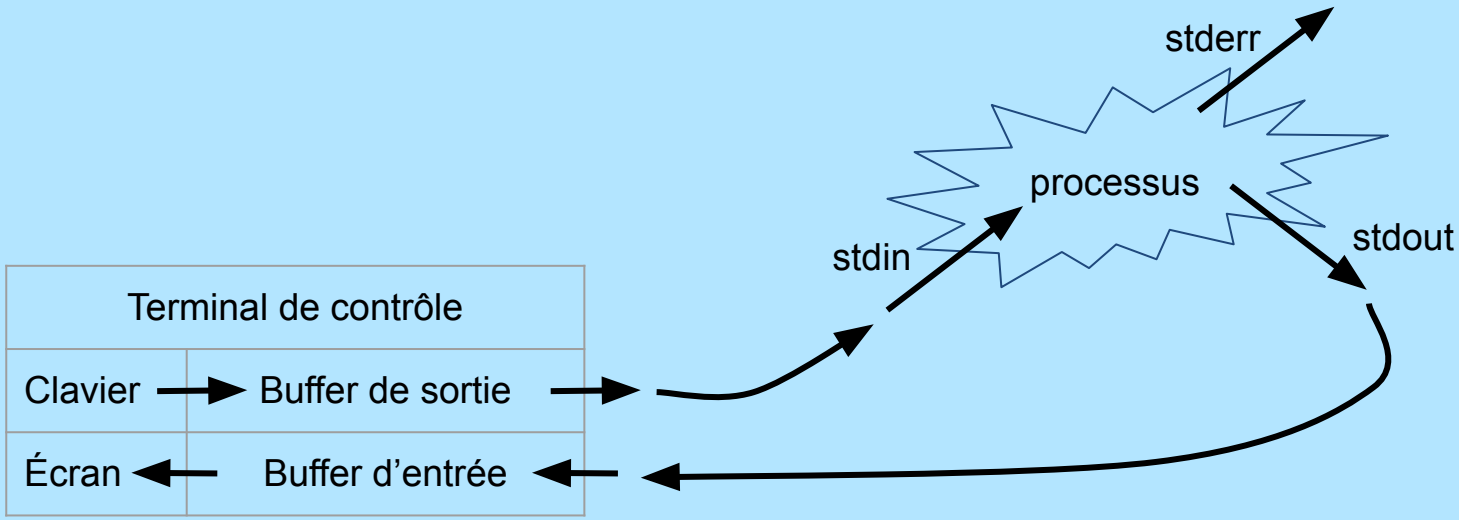
ex2 : lecture d'un texte au clavier

Flux d'entrées-sorties

- https://fr.wikibooks.org/wiki/Programmation_C/Entr%C3%A9es/sorties
- Flux : canal destiné à envoyer ou recevoir des données, associé à un tampon
- structure FILE : type opaque (Cf. /usr/include/x86_64-linux-gnu/bits/libio.h pour les curieux)
  - "coquille entourant les descripteurs de fichiers en ajoutant une mémoire tampon"
  - donne accès aux données, à la gestion du tampon, à la position dans le flux…
- 3 flux ouverts par défaut (Cf. redirections du shell !)
  - FILE * stdin : entrée standard
  - FILE * stdout : sortie standard
  - FILE * stderr : sortie d'erreur
- Ouvrir d'autres flux : fp = fopen(<chemin>,<mode>), fclose(fp)

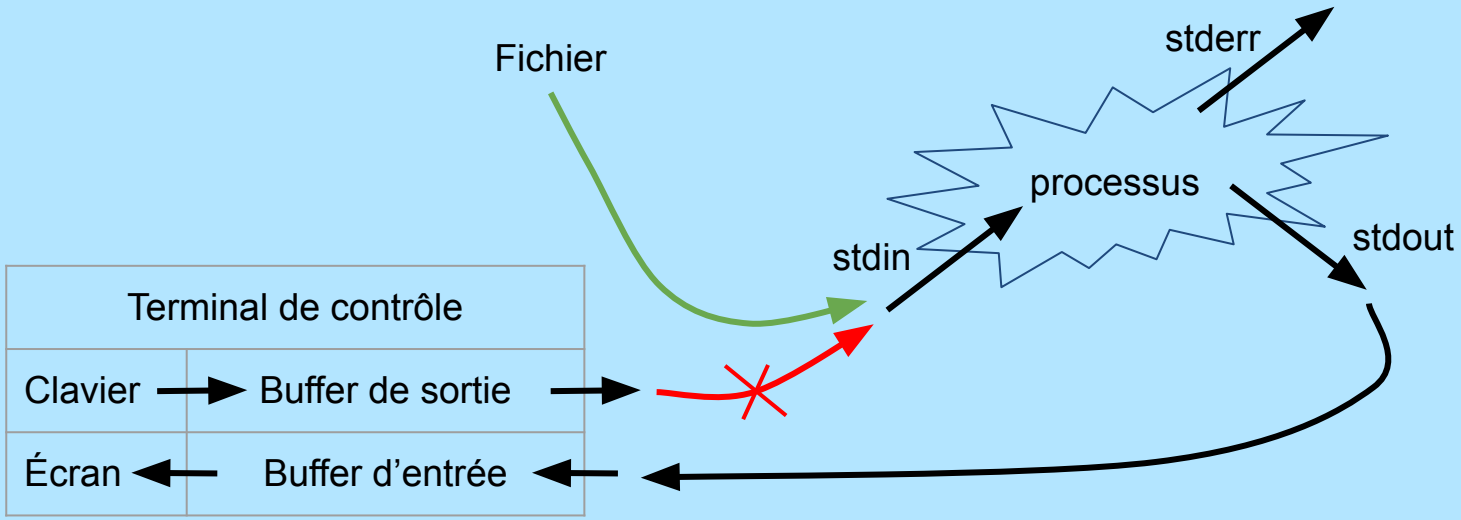
Relations Terminal-Processus

- Le buffer de sortie est vidé vers le processus lors de l'appui sur <ENTREE>
- Le caractère \n est inséré dans le buffer



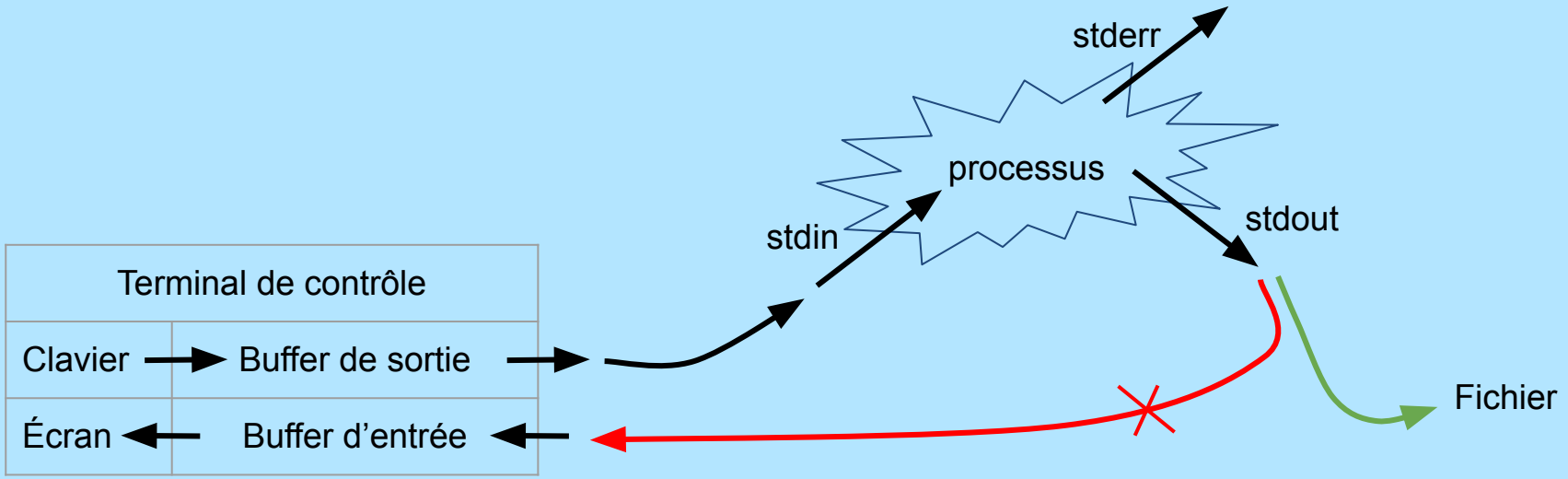
Redirection d'entrée / de données
Symbole <

cmd < fichier



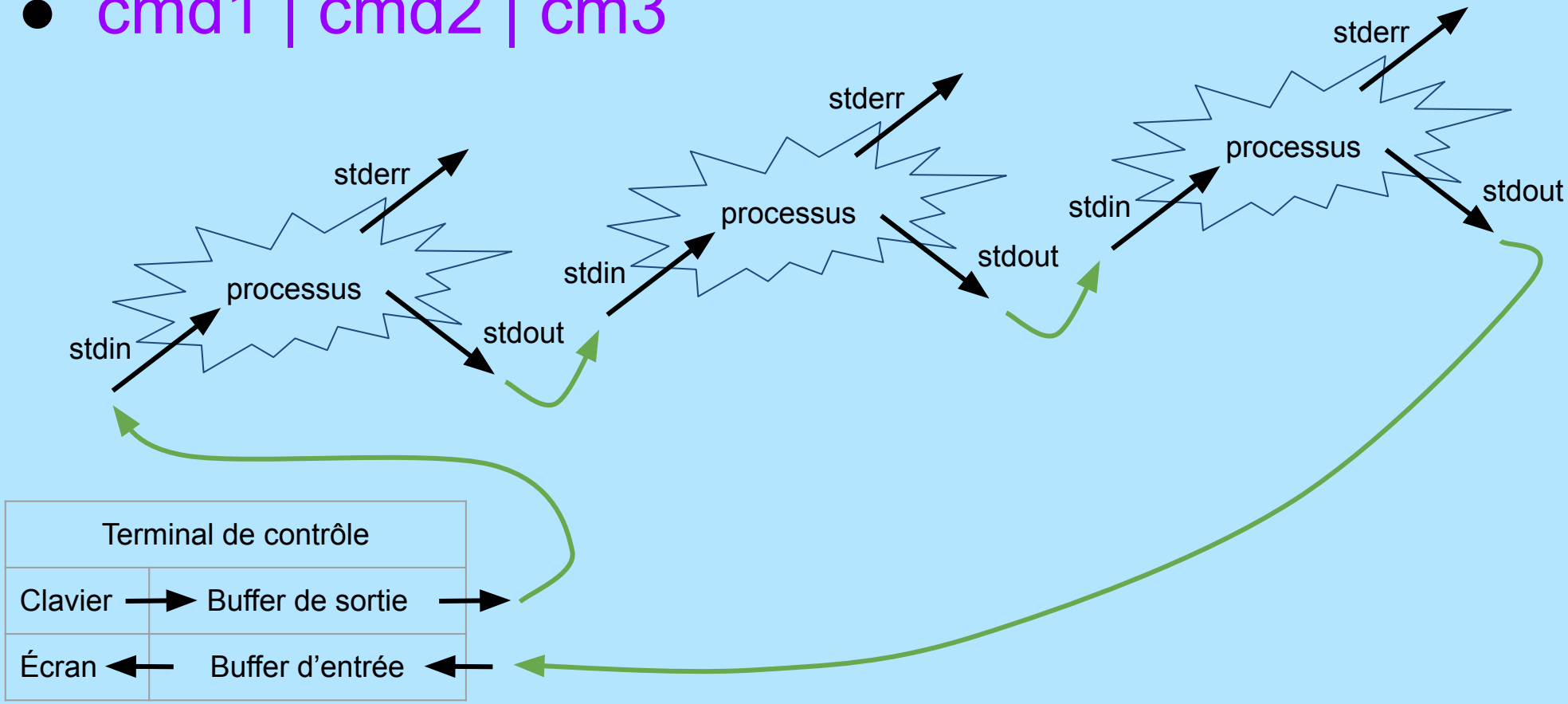
Redirection de sortie / de résultats
Symboles > ou >>

- cmd > fichier (écrase fichier)
- cmd >> fichier (concaténation)



Redirections de commandes
Symbole |

- cmd1 | cmd2 | cm3



Lecture d'un caractère depuis un flux : getchar, fgetc

- Lecture d'un caractère depuis le flux d'entrée standard stdin :
  - getchar()
  - fgetc(stdin)
- Valeur renvoyée :
  - Code ascii du caractère lu
  - EOF ("*END OF FILE*")
    - Fin de fichier atteinte dans le cadre d'une redirection
    - Ctrl+D saisi par l'utilisateur dans le cadre d'un mode interactif

Exemple : Lecture d'un texte au clavier

```
#include <stdio.h>

int main(void) {
   char c ;

   printf("Saisir des caractères au clavier (Ctrl+D pour
terminer)\n");

   while ( ( c = getchar() ) != EOF) {
      printf("Caractère lu : '%c'\n",c);
   }

}
```

Exercice Corrigé : ex2 : Lecture d'un texte au clavier

- Réaliser la lecture de caractères depuis l'entrée standard
  - Les caractères saisis dans le terminal seront envoyés au processus lors de la saisie d'un saut de ligne
  - Le caractère de contrôle EOF peut être envoyé à l'aide du raccourci clavier Ctrl+D
  - [BONUS] Permettre de sélectionner la fonction de lecture par une constante symbolique, modifiée lors de la compilation depuis le fichier makefile
- [BONUS] Réaliser la lecture de caractères depuis une redirection d'entrée / de données

centralelille

AUDACE • EXIGENCE • RESPECT

3h05

45min

Exercice : ex_comptage

Comptage de caractères dans un texte

Cf. Trame séance 1 - 2022-2023

centralelille

AUDACE • EXIGENCE • RESPECT

2h20

20 min

Pointeurs

ex3 : utilisation de pointeurs sur char

Pointeur : Kezako ?

- Un type de variable : “adresse de ...”
  - Représentation mémoire : 8 octets (sur PC 64 bits)
  - Interprétation : représente l’adresse d’une autre variable dont le type est connu (ce qui permettra d’évaluer les expressions d'arithmétique de pointeurs)
- Syntaxe :

```
char c; // caractère (1 octet)

char * p_sur_c; // adresse d’un caractère (8 octets)
```

Pointeur : initialisation

- char c = 'A'; // A vaut 65
- char * p = NULL;
  - Mot-clé NULL : représente (void *) 0
- p = &c;
  - & ⇔ “Adresse de”
  - Le pointeur p vaut l’adresse mémoire de c

Pointeur : utilisation

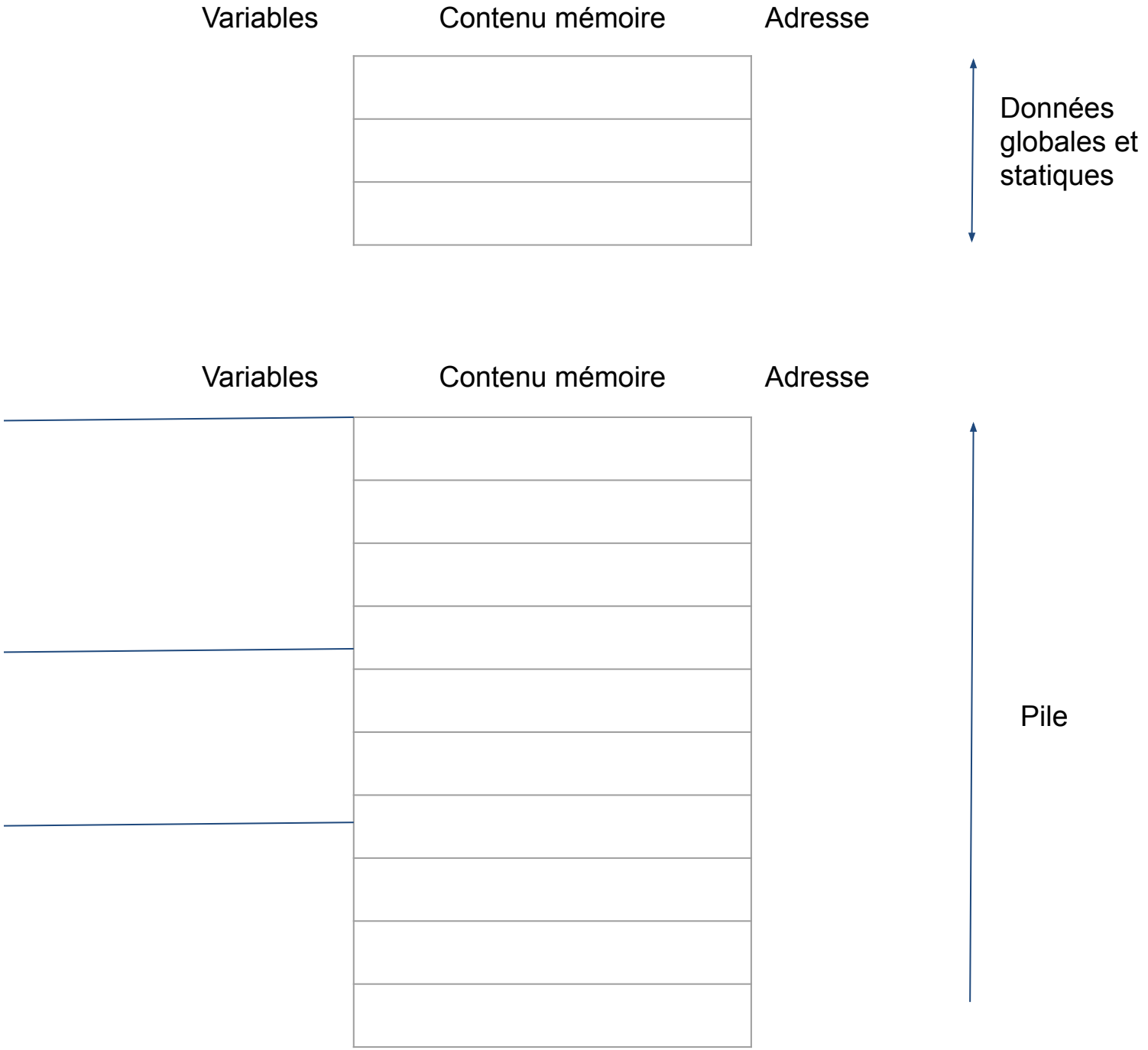
- char c = ‘A’; // A vaut 65
- char * p = &c; // Le pointeur p vaut l’adresse de c
- *p = 'B';
  - (* pointeur) employé dans une affectation ⇔ “ce qui est pointé par le pointeur”
- Afficher la valeur (*qui est une adresse*) d’un pointeur : symbole de format %p

Traces d'exécution pour les pointeurs
Cf. include/traces.h

- Macro-fonctions utiles pour afficher des informations sur la mémoire du programme
- CODE(); // Affiche CODE: <...>
- DUMPHEAD(); // Légende des tableaux
- DUMPCHAR(varname); // Affichage d'une variable de type char
- DUMPINT(varname); // Affichage d'une variable de type int
- DUMPAD(varname); // Affichage d'un pointeur

Exemple : ex3

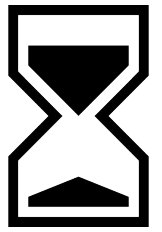
- Création et utilisation d'un pointeur sur caractère
- Affichage des adresses mémoire concernées
  - Inscrire les adresses des variables manipulées sur la pile dans la représentation mémoire située sur le slide suivant
- Comprendre l'arithmétique des pointeurs sur char
  - Peut-on généraliser avec des pointeurs sur int ?


Variables
Contenu mémoire
Adresse
Données globales et statiques
Variables
Contenu mémoire
Adresse
Pile

centralelille

AUDACE • EXIGENCE • RESPECT

2h



15 min

Arithmétique des pointeurs

ex4 : tableaux et pointeurs

Arithmétique des pointeurs
Notation "tableau" : p[n] == *(p+n)

- Soit p un pointeur sur un <type>
- (*p)+1 renvoie simplement la valeur pointée par p augmentée de 1...
- *(p+1) par contre :
  - Calcule l'adresse "p+1", ce qui représente (en octets), la valeur p + sizeof(<type pointé par p>) !!
  - Renvoie le contenu de la zone mémoire située à cette adresse

Ce n'est pas une simple somme !

- Si p dénote l'adresse d'une case dans un tableau, p+1 est l'adresse de la case suivante, et *(p+1) le contenu de cette case
  - Quelles que soient les tailles des cases du tableau !
- p[n] (avec n un entier) représente précisément l'expression *(p+n) !!

p[n] représente *(p+n) Pourquoi ??

- La déclaration d'un tableau (de valeurs d'un <type> donné) alloue de la mémoire pour stocker ce tableau de manière contigüe
  - sur la pile
- Le nom du tableau est un identificateur qui représente l'adresse mémoire de sa première case !
  - nom[0] représente le contenu de la première case
    - C'est bien *nom puisque nom est l'adresse de cette case
  - nom[n] représente le contenu de la nième case
    - Cette case est située à l'adresse nom + n * sizeof(<type>)
    - C'est donc bien égal à *(nom+n)

Attention !

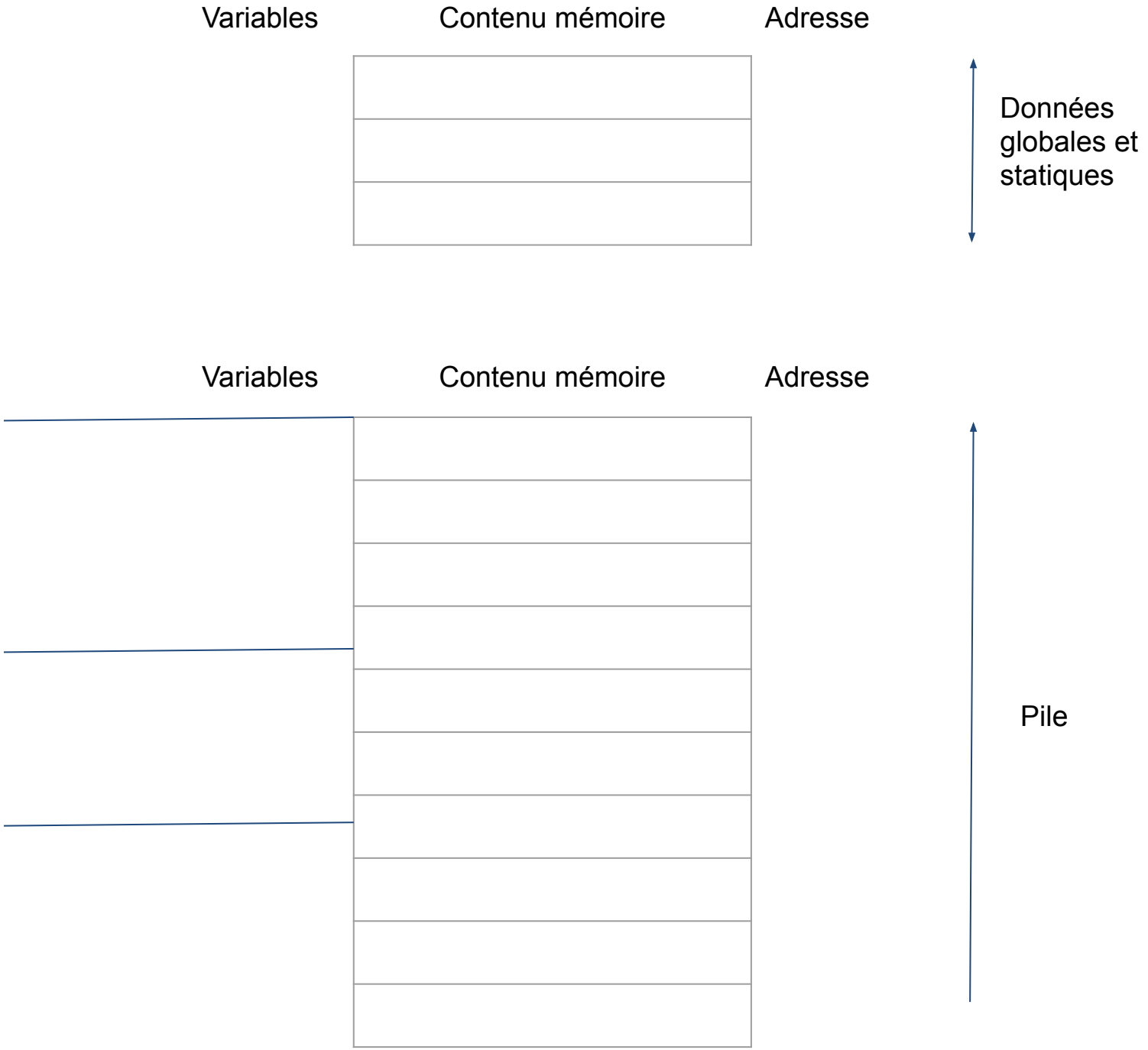
- On ne peut pas écrire :

```
int tab1[5];
int tab2[5];
tab1 = tab2;
```

- L'expression tab1 représente l'adresse du tableau tab1
  - Les tableaux ne se déplacent pas !

Exercice corrigé : ex4

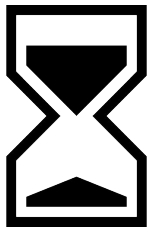
- Quelle erreur s'affiche lors de l'affectation de tableaux entre eux ?
- Vérification de l'arithmétique des pointeurs en utilisant les notations *(p+i) et p[i]
- Manipulation d'un tableau d'entiers :
  - Créer un tableau d'entiers de MAXLEN cases
  - Initialiser le contenu des cases du tableau à l'aide d'une boucle
  - Créer un pointeur pointant sur le début du tableau
  - Utiliser le pointeur pour afficher le contenu du tableau et les adresses de ses cases (format %p dans printf)
  - Utiliser les deux notations


Variables
Contenu mémoire
Adresse
Données globales et statiques
Variables
Contenu mémoire
Adresse
Pile

centralelille

AUDACE • EXIGENCE • RESPECT

1h45



15 min

Allocation dynamique de mémoire

ex5 : résoudre un problème de segfault avec une allocation dynamique

Allocation dynamique de mémoire
Cf. man malloc <stdlib.h>

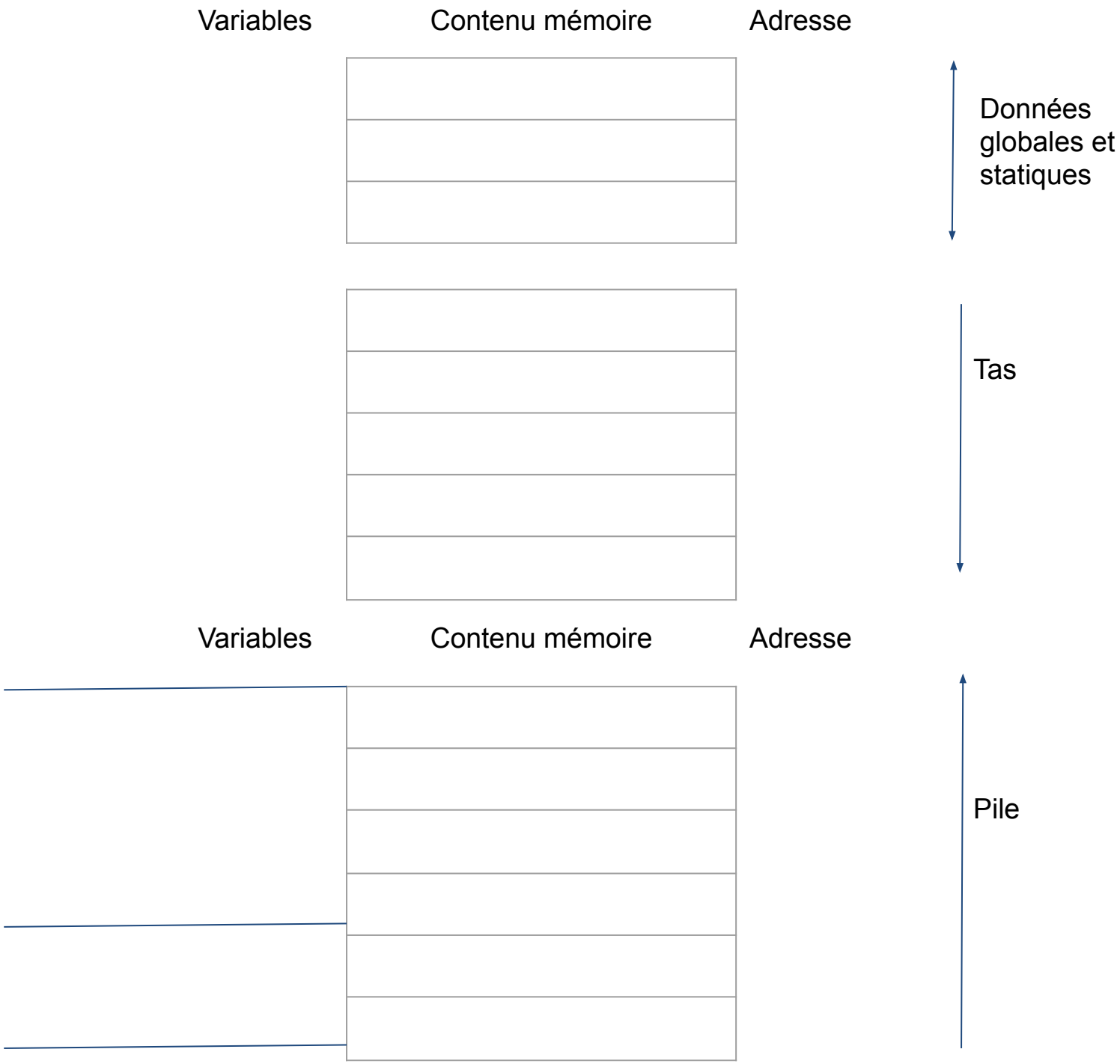
- malloc(), calloc(), realloc() : Pour initialiser les pointeurs
  - Renvoie l'adresse de début de la zone mémoire (void *)
- void *malloc(size_t size);
  - zone non initialisée
- void *calloc(size_t nmemb, size_t size);
  - zone initialisée à 0
- void *realloc(void *ptr, size_t size);
  - agrandissement d'une zone préalablement allouée
- void free(void *ptr);
  - libère la zone de mémoire pointée

Bonnes pratiques

- Il est d'usage de "*caster*" l'adresse de retour de la fonction malloc
  - int * p = (int *) malloc(MAXTAB * sizeof(int));
  - int * p = (int *) calloc(MAXTAB, sizeof(int));
- Si ces fonctions échouent, elles renvoient NULL
  - Elles peuvent donc échouer !
  - Cf. Macro CHECK_IF, capsule 6 (Cf. include/check.h)
- Penser à libérer explicitement la mémoire avec lorsqu'elle n'est plus utilisée :
  - free(p);
  - p = NULL;

Exemple : ex5

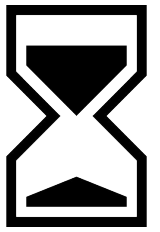
- Provoquer une segmentation fault à cause de l'utilisation d'un pointeur sur int mal/non initialisé
- Résoudre le problème en utilisant une allocation dynamique de mémoire
  - Tester les valeurs de retour des appels système
- Inscrire les adresses des variables manipulées sur le tas dans la représentation mémoire située sur le slide suivant


Variables
Contenu mémoire
Adresse
Données globales et statiques
Tas
Variables
Contenu mémoire
Adresse
Pile

centralelille

AUDACE • EXIGENCE • RESPECT

1h30



15 min

Fonctions et pointeurs

Passage de paramètres par référence

Modificateur const

ex6 : fonctions incr/swap

Passage de paramètre par référence

- Pour effectuer un passage de paramètre par référence, il suffit de passer, par valeur, l'adresse d'une variable !

```
void modifier(int * p) {
    // p contient l'adresse d'un entier que l'on va pouvoir modifier
}
```

Exercice corrigé : ex6

- Produire une fonction permettant d'incrémenter une valeur passée en paramètre
  - void incr(int * p);
- Produire une fonction permettant d'échanger les valeurs de deux variables
  - void swap(int * p1, int * p2);

Modificateur const

- En utilisant le mot-clé const, on s'engage à ne pas modifier les valeurs situées à l'adresse passée en paramètre : void lecture_seule(const int * p) { …
- Le compilateur refusera* de compiler si le code à l'intérieur de la fonction modifie les données situées à l'adresse pointée
  - ou si vous envoyez cette adresse à une fonction qui ne s'engage pas elle-même à ne pas y toucher...

Passage de paramètre de type tableau

int tab[] <=?=> int * pTab

- **Dans les arguments d'une fonction**, "tableau de" et "pointeur sur" représentent la même chose : l'adresse de la première case du tableau !
- Les prototypes suivants sont donc équivalents :
  - void traiterTableau(int t[])
  - void traiterTableau(int * t)
- Mais **partout ailleurs**, ce sont deux choses différentes !

int tab[] <=/=> int * pTab
Segmentation Fault !

- int tab[MAXTAB]; demande au compilateur de réserver MAXTAB * sizeof(int) octets
  - tab représente l'adresse de début de la zone allouée
- int * pTab; demande au compilateur de réserver un espace de la taille d'une adresse (8 octets)
  - Dans cet espace il n'y a certainement pas l'adresse de quoi que ce soit d'utile
  - Vouloir accéder à l'adresse indiquée conduira à des erreurs de segmentation

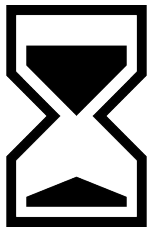
int tab[] <=/=> int * pTab
Segmentation Fault ! => Solution

- int * pTab; // pTab pointe n'importe où
- Il faut affecter à pTab l'adresse d'une zone mémoire que l'on pourra manipuler
  - On utilise de l'allocation dynamique de mémoire
- pTab = (int *) malloc (MAXLEN * sizeof(int));

centralelille

AUDACE • EXIGENCE • RESPECT

1h15



15 min

Chaînes de caractères

Chaînes de caractères

string.h

Arguments en ligne de commande

Chaînes de caractères

- Un tableau de caractères, qui contient un caractère de fin de chaîne, de code ASCII 0 (== '\0')
  - Cf. man ascii
  - NB : 'A' == 65
- Initialisation par : char * chaine = “coucou”;
  - 7 caractères, donc !
  - L’identificateur chaine dénote l’adresse du premier caractère de la chaîne
- Symbole de format pour printf : %s
- Une expression entre guillemets dénote l’adresse où le compilateur aura le premier caractère de la chaîne correspondante
  - Attention : les expressions entre guillemets sont stockées dans des segments mémoire en lecture seule !

Traitement de chaînes de caractères
Fichier d'entête <string.h>
https://fr.wikipedia.org/wiki/String.h

- char *strcat(char *dest, const char *src)
  - Concatène la chaîne src à la suite de dest
- char *strncat(char * dest, const char * src, size_t n)
  - Concatène au plus n caractères de la chaîne src à la suite de dest
- char *strchr(const char *, int)
  - Cherche un caractère dans une chaîne et renvoie un pointeur sur le caractère, en cherchant depuis le début
- int strcmp(const char *, const char *)
  - Compare deux chaînes lexicalement
- char *strcpy(char *toHere, const char *fromHere)
  - Copie une chaîne de caractères d'une zone à une autre
- char *strdup(const char *)
  - Alloue et duplique une chaîne en mémoire
- size_t strlen(const char *)
  - Retourne la longueur d'une chaîne caractères
- + de nombreuses autres...

Tests et conversions de chaînes de caractères
Fichier d'entête <ctype.h>

- int isupper(int C) : retourne VRAI si C est une majuscule
- int islower(int C) : retourne VRAI si C est une minuscule
- int isdigit(int C) : retourne VRAI si C est un chiffre décimal
- int isalpha(int C) : retourne VRAI si islower(C) ou isupper(C)
- int isalnum(int C) : retourne VRAI si isalpha(C) ou isdigit(C)
- int isxdigit(int C) : retourne VRAI si C est un chiffre hexadécimal
- int isspace(int C) : retourne VRAI si C est un signe d'espacement
- int tolower(int C) : retourne C converti en minuscule si possible, sinon C
- int toupper(int C) : retourne C converti en majuscule si possible, sinon C

atoi()
Fichier d'entête <stdio.h>

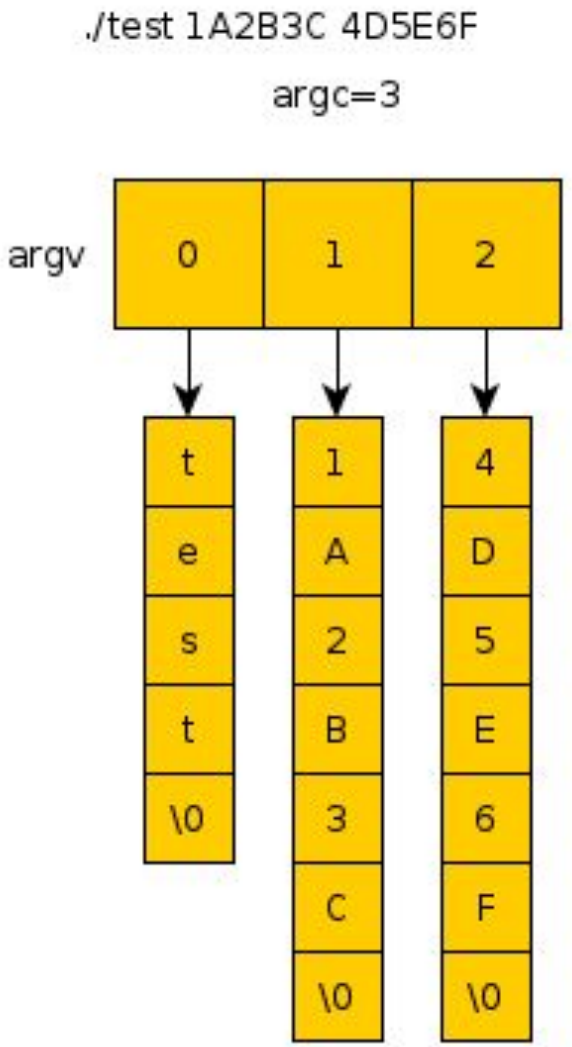
- int atoi(const char *nptr) : convertit le début de la chaîne pointée par nptr en entier de type int

sprintf()
Fichier d'entête <stdio.h>

- Écriture d'une chaîne dans un buffer
  - Qu'il faut avoir préalablement alloué !
- int sprintf(char *str, const char *format, ...);
- int snprintf(char *str, size_t size, const char *format, ...);

Arguments en ligne de commande

- int main(int argc, char * argv[])
- int main(int argc, char ** argv)
- Récupération du nombre et des chaînes passées en paramètres lors de l'exécution du programme



centralelille

AUDACE • EXIGENCE • RESPECT

Entrées-Sorties (2)

Flux d'entrées-sorties

- https://fr.wikibooks.org/wiki/Programmation_C/Entr%C3%A9es/sorties
- Flux : canal destiné à envoyer ou recevoir des données, associé à un tampon
- structure FILE : type opaque (Cf. /usr/include/x86_64-linux-gnu/bits/libio.h pour les curieux)
  - "coquille entourant les descripteurs de fichiers en ajoutant une mémoire tampon"
  - donne accès aux données, à la gestion du tampon, à la position dans le flux…
- 3 flux ouverts par défaut (Cf. redirections du shell !)
  - FILE * stdin : entrée standard
  - FILE * stdout : sortie standard
  - FILE * stderr : sortie d'erreur
- Ouvrir d'autres flux : fp = fopen(<chemin>,<mode>), fclose(fp)

Lectures : scanf, getchar, gets, fgetc, fgets
Fichier d'entête <stdio.h>

- getchar() ⇔ fgetc(stdin) : lecture d'un caractère ou EOF
- scanf("%s", buffer) : lecture depuis stdin
  - Laisse dans le tampon du flux les caractères d'espacement tel que l'espace, la tabulation ou le retour chariot ⇒ Ne permet que la saisie d'un mot continu et non d'une phrase
- gets(buffer) : lecture depuis stdin
  - Lit une ligne entière au clavier et retire le retour chariot final du tampon du flux
  - Pas de contrôle de la longueur des informations entrées ⇒ risque de débordement, déconseillé (cf. man gets !!)
- fgets(buffer, sizeof(buffer), stream) : lecture depuis stream
  - Permet de lire des lignes entières de taille contrainte + le retour chariot final
  - Place un caractère de fin de chaîne dans le buffer

Terminal

Fichier Édition Affichage Rechercher Terminal Aide

GETS(3) Manuel du programmeur Linux GETS(3)

NOM

gets - Récupérer une chaîne sur l'entrée standard (OBSOLÈTE)

SYNOPSIS

```
#include <stdio.h>

char *gets(char *s);
```

DESCRIPTION

Ne jamais utiliser cette fonction.

gets() lit une ligne depuis stdin et la place dans le tampon pointé par s jusqu'à atteindre un retour chariot, ou **EOF**, qu'il remplace par un octet nul (« \0 »). Il n'y a pas de vérification de débordement de tampon (voir la section des **BOGUES** plus bas).

VALEUR RENVOYÉE

gets() renvoie le pointeur s en cas de succès et NULL en cas d'erreur, ou si la fin de fichier est atteinte avant d'avoir pu lire au moins un caractère. Cependant, le débordement de tampon n'étant pas surveillé, il ne peut y avoir de certitude que la fonction renvoie quelque chose.

CONFORMITÉ

C89, C99, POSIX.1-2001.

LSB déconseille l'utilisation de **gets**(). POSIX.1-2008 marque **gets**() comme étant obsolète. ISO C11 retire la specification de **gets**() du langage C et, depuis la version 2.16, les fichiers d'en-tête glibc n'exposent pas la déclaration de fonction si la macro de test de fonctionnalités **_ISOC11_SOURCE** est définie.

BOGUES

N'utilisez jamais **gets**(). Comme il est impossible de savoir à l'avance combien de caractères seront lus par **gets**(), et comme celui-ci écrira tous les caractères lus, même s'ils débordent du tampon, cette fonction est extrêmement dangereuse à utiliser. On a déjà utilisé ce dysfonctionnement pour créer des trous de sécurité. UTILISEZ TOUJOURS **fgets**() À LA PLACE DE **gets**().

Pour plus d'informations, consultez CWE-242 (document intitulé sous le nom « Use of Inherently Dangerous Function ») à l'adresse http://cwe.mitre.org/data/definitions/242.html

VOIR AUSSI

Manual page gets(3) line 1 (press h for help or q to quit)

man gets !

Ouverture d'un nouveau flux : fopen

- FILE * fopen(const char * chemin, const char * mode)
- r : ouvre le fichier en lecture, le flux est positionné au début du fichier
- r+ : ouvre le fichier en lecture/écriture, le flux est positionné au début du fichier
- w : ouvre le fichier en écriture, supprime toutes les données si le fichier existe et le crée sinon, le flux est positionné au début du fichier
- w+ : ouvre le fichier en lecture/écriture, supprime toutes les données si le fichier existe et le crée sinon, le flux est positionné au début du fichier
- a : ouvre le fichier en écriture, crée le fichier s'il n'existe pas, le flux est positionné à la fin du fichier
- a+ : ouvre le fichier en lecture/écriture, crée le fichier s'il n'existe pas, le flux est positionné à la fin du fichier.

Ecritures & gestion du tampon
Fichier d'entête <stdio.h>

- int fprintf(FILE *stream, const char *format, ...)
  - Ecriture dans un flux : fprintf(stdin, …) ⇔ printf(...)
- int feof(FILE *stream)
  - Renvoie VRAI si la fin de fichier a été atteinte
- int fflush(FILE *stream)
  - Force l'écriture de toutes les données se trouvant dans les tampons de l'espace utilisateur (ce qui se produit normalement lorsque des retours chariots sont insérés dans le flux)
  - Pour les flux de sortie du processus uniquement !
- int fseek(FILE *stream, long offset, int whence)
  - Change la position dans le flux
- long ftell(FILE *stream)
  - Indique la position dans le flux

centralelille

AUDACE • EXIGENCE • RESPECT

Exercice : ex_cesar

Chiffrage par substitution mono-alphabétique

Cf. Trame séance 1 - 2022-2023

TEA S1

- TEA Séance 1 (2022-2023)
- Travail d'approfondissement sur les pointeurs
- Non évalué
  - Sera évalué à l'occasion du test de la séance 2

Nous ferons l'hypothèse que tous ces sujets sont parfaitement maîtrisés pour commencer la séance 2, ce dont le test de début de séance tiendra compte.

centralelille

AUDACE • EXIGENCE • RESPECT

Code Couleur

Légende des textes

- mot-clé important, variable, contenu d'un fichier, code source d'un programme
- chemin ou url, nom d'un paquet logiciel
- commande, raccourci
- commentaire, exercice, citation
- culturel, optionnel

Culturel / Approfondissement

- A ne pas connaître intégralement par coeur
  - Donc, le reste… est à maîtriser parfaitement !
- Pour anticiper les problématiques que vous rencontrerez en stage ou dans d'autres cours
- Pour avoir de la conversation à table ou en soirée...

Exemples ou Exercices

- Brancher le cerveau
- Participer
- Expérimenter en prenant le temps...

Bonnes pratiques, prérequis

- Des éléments d'organisation indispensables pour un travail de qualité
- Des rappels de concepts déjà connus