Programmation C : Cahier d'Exercices

Table des matières

Chapitre 1

Compilation, structures de contrôle, opérations sur les types, opérations bits-à-bits

Où l'étudiant essaiera plus ou moins péniblement de se rappeler comment exécuter un programme C très simple, puis se familiarisera avec ses concepts de base.

1 Compiler, lier, exécuter un programme C

On commence par quelques exercices permettant de compiler et exécuter un programme simple, ce que vous savez normalement faire après les cours.
Quelques rappels tout d'abord :

— Pour fabriquer un exécutable à partir d'un fichier source grâce à GCC :

Compilation `gcc -c -Wall -Werror -o <outfile.o> <file.c>`

Édition de lien `gcc -o executable <tous-vos-fichiers-objets.o`

— Pour afficher un message sur la sortie standard `printf("A message that is printed out\n");`
Si vous voulez afficher un nombre entier :

```
int my_variable = -42;
printf("My signed integer is: %d\n", my_variable);
```

Si vous voulez afficher un nombre entier non signé :

```
unsigned int my_variable = 42;
printf("My unsigned integer is: %u\n", my_variable);
```

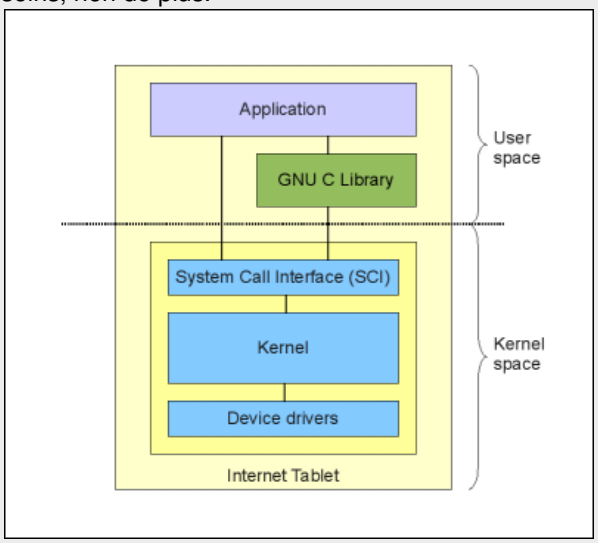
On reviendra sur `printf()` (et ses copines utilitaires) dans le Chapitre 3 mais pour l'instant, ces utilisations toutes simples devraient suffire. Attention cependant à bien comprendre dans quelle partie du logiciel se trouve la fonction `printf()`.

Appel à la libC vs. appel système

La confusion est souvent faite entre un appel à une fonction faisant partie de la *librairie C* et *appel système*. Un appel système est un service rendu par le *noyau* du système d'exploitation. Cela implique des mécanismes complexes :

- changement d'espace d'adressage, impact sur les
- sauvegarde et restauration des registres processeurs
- changement de mode d'exécution du processeur
- gestion de la concurrence d'accès à certaines variables partagées
- etc.

Vous ne pouvez pas appréhender aujourd'hui ces mécanismes mais vous pouvez par contre accepter l'idée qu'ils sont coûteux en temps, et qu'ils n'ont rien à voir avec un simple appel à une fonction de la librairie C. Un appel à une telle fonction se passe exactement comme si vous appeliez une fonction écrite par vos soins, rien de plus.



QUESTION 1 ► Grâce à un éditeur de texte soigneusement choisi, écrivez un programme qui affiche votre âge, votre prénom et votre chanson préférée.

QUESTION 2 ► Écrivez un Makefile, ou adaptez-en un (il y en a un sur Moodle) pour qu'il fabrique, avec les options de compilation données ci-dessus, l'exécutable correspondant à votre programme. Puis, exécutez-le.

2 Types entiers, casts, et promotion

Les questions ci-dessous vous permettent de vous familiariser, de réviser, ou pour quelques rares spécimens parmi vous, de revoir pour la 50e fois les types de base en C.
Je ne savais pas où mettre ça mais c'est important donc je vous le dis tout de suite : l'opérateur `sizeof()` permet de connaître la taille, en octets, du type passé en paramètre. Un autre truc important, c'est de maîtriser ces différentes manières d'écrire un entier :

décimal 42

hexadécimal 0x2A

binaire 0b101010. Cette manière de noter est valide chez GCC mais appartient à une *extension*. GCC râle si l'option *-pedantic* est indiquée.

(les 3 valeurs données ci-dessus sont les mêmes)

QUESTION 3 ► Le type `char` stocke une valeur entière, toujours sur un seul octet. Attention, cette valeur peut être signée ou non, cela dépend de votre machine, du compilateur, de la version du compilateur, et sûrement de l'âge du capitaine. Par contre, les types `signed char` et `unsigned char` permettent de préciser. Écrivez un petit programme grâce auquel vous saurez, sur votre ordinateur, si un `char` est signé ou non.

QUESTION 4 ► `int` est équivalent à `signed int`, mais le type `unsigned int` permet de ne stocker que des valeurs positives. Par contre, la taille de ces deux types n'est pas spécifiée et dépend de votre machine, du compilateur, et du sens du vent. Écrivez un programme affichant sur la sortie standard :

— la taille, les valeurs min et max d'un `int`

— la taille et la valeur max d'un `unsigned int`

QUESTION 5 ► Écrivez un petit programme qui affecte respectivement aux variables `super`, `bof`, `naze`, `superman`, `sousleau` les valeurs 17 en notation hexadécimale, 10 en notation binaire, 8 en notation décimale, 22 en notation binaire, 2 en notation hexadécimale. Choisissez le type que vous voulez pour ces variables.

QUESTION 6 ► Lorsqu'il est important de maîtriser la taille d'un type entier, nous utilisons les types suivants :

— `int8_t` et `uint8_t` codées sur un octet et stockant des valeurs entières respectivement signées et non-signées

— `int16_t` et `uint16_t` codées sur deux octets et stockant des valeurs entières respectivement signées et non-signées

— `int32_t` et `uint32_t` ... bla... 4 octets bla...

— `int64_t` et ... `uint64_t` !

Quel type entier, à taille spécifiée ou non, utiliseriez-vous pour stocker les informations suivantes :

— un caractère ASCII

— le nombre d'atomes de la terre

— le numéro dans une adresse

— le nombre de cartons de vin dans le hangar d'un viticulteur

QUESTION 7 ► Écrivez un programme affichant sur combien d'octets le type `long` est codé. Idem pour le type `long long`.

QUESTION 8 ► QUESTION 9 ► Attention au dépassement de capacité ! Que se passe-t'il lorsque vous ajoutez 1 à un entier qui a la valeur maximum autorisée par son type ?

QUESTION 10 ► Dans l'exemple ci-dessous, un dépassement de capacité sur l'addition n'implique pas forcément un dépassement de capacité sur `a`. Vérifiez-le en affectant de grandes valeurs à a, b et c. Pour comprendre ça complètement, il faut vous rappeler ou attendre vos cours d'architecture (complément à deux).

```
uint_32 a,b,c,d;
a = (b + c) - d;
```

QUESTION 11 ► Attention aux comparaisons entre entiers signés et non-signés :

```
int main()
{
  if (sizeof(int) < -1)
      printf("Bizarre, bizarre ... ??\n");
  else
      printf ("Tout semble normal\n");
}
```

QUESTION 12 ► Les nombres à virgules sont représentés au moyen des types `float`, et `double`. Vous verrez avec Florent de Dinechin comment les nombres à virgules sont codés, mais écrivez tout de même tout de suite un programme qui affiche la variable `pi` ainsi que le périmètre d'un cercle, en donnant à la variable `pi` :

— le type entier et la valeur 3.14 ;

— le type `float` et la valeur et la valeur 3.14 ;

— le type `float` et la valeur et la valeur 3.14159265358979323846264338327

— le type `double` et la valeur 3.14159265358979323846264338327

.

QUESTION 13 ► Améliorez la précision de l'affichage de `pi` en utilisant le format `%0.28f` dans `printf` à la place de simplement `%f` (cela indique d'afficher jusqu'à 28 décimales). Cela suffit-il à retrouver la valeur que vous avez affectez à `pi` ? Comment corriger cela ?

QUESTION 14 ► Le type `bool` peut être utilisé en incluant `<stdbool.h>` et stocke l'information "vrai ou faux". Attention, le nombre d'octets utilisés n'est ni inférieur à 1 (hum...) et ne vaut pas forcément 1. Écrivez un petit programme affichant sur la sortie standard :

— la taille d'un `bool`

— la valeur entière codant l'information *vrai*

— la valeur entière codant l'information *faux*

QUESTION 15 ► Le type `char` est un type entier, comme int, et peut donc être additionné à une autre valeur entière. Comprenez, compilez et exécutez ce petit exemple.

```
char mychar = 'A';
int val = mychar + 10;
printf("val = %d", val);
```

QUESTION 16 ► L'exemple précédent marche car la variable `mychar` est *convertie* par le compilateur en type `int` avant l'addition. Ce genre de conversion marche souvent tout seul, mais pas toujours, et les erreurs de compréhension liées aux conversion de types sont sources de gros gros bugs. Dans l'exemple ci-dessous, le type du résultat de la soustraction est-il un `uint16_t` ?

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

int main(int argc, char ** argv)
{
    uint16_t a = 413;
    uint16_t b = 64948;

    fprintf(stdout, "%u\n", (a - b));

    return 0;
}
```

QUESTION 17 ► Vous avez répondu "Non" à la question précédente ? Bravo, vous êtes trop fort. Comprenez tout de même ce qui se passe en lisant l'encart ci-dessous (*Conversion implicite de type*).

Conversion implicite de type

Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont converties automatiquement dans un type commun. Lors d'une affectation, la donnée à droite du signe d'égalité est convertie dans le type à gauche du signe d'égalité. Dans ce cas, il peut y avoir perte de précision si le type de la destination est plus faible que celui de la source. De même, lors de l'appel d'une fonction, les paramètres sont automatiquement convertis dans les types déclarés dans la définition de la fonction.
Dans n'importe quelle expression entière, si un `int` peut représenter toutes les valeurs de cette expression (par exemple `uint16_t`), alors il est converti implicitement par le compilateur en `int`. Si cela ne suffit pas, la conversion est faite vers `unsigned int`, si possible.
Dans le cas où une opération artithmétique est efectuée entre deux expressions qui n'ont pas le même type après la conversion entière décrite plus haut, alors la conversion de type continue selon les règles suivantes :

- si les deux opérandes sont de type flottant, alors l'opérande ayant le rang le plus bas est converti vers un type de même rang que l'autre opérande, l'ordre des rangs étant définie comme suit : float -> double -> long double.
- si un seul des opérande est est de type flottant, le type entier est converti dans ce type flottant.
- si les deux opérandes sont de type entier, alors :
  - si les deux opérandes sont tous deux de de type non-signé, ou tous deux de type signé, le type de plus petit rang est converti dans l'autre, la hiérarchie de rang chez les entiers étant la suivante :
    1. long long int, unsigned long long int
    2. long int, unsigned long int
    3. int, unsigned int
    4. short int, unsigned short int
    5. signed char, char, unsigned char
  - sinon si le type T_u d'un des opérande est non signé et a un rang égal ou supérieur au type T de l'autre opérande, alors T est convertie dans T_u.
  - sinon, si le type T_s de l'un des opérande est signé et peut représenter toutes les valeurs de l'autre opérande, alors T est converti en T_s
  - sinon les deux opérandes sont converties dans le type entier non signé correspondant au type de l'opérande qui a un type signé.

Si une perte de précision est possible lors de la conversion implicite de type, le compilateur peut génèrer un warning.
Exemples :

int	2 + 3	int 2 + int 3 → int 5
double	2.2 + 3.3	double 2.2 + double 3.3 → double 5.5
mix	2 + 3.3	int 2 + double 3.3 → double 2.0 + double 3.3 → double 5.3
int	1 / 2	int 1 / int 2 → int 0
double	1.0 / 2.0	double 1.0 / double 2.0 → double 0.5
mix	1 / 2.0	int 1 / double 2.0 → double 1.0 / double 2.0 → double 0.5

Maintenant, corrigez votre programme pour qu'il affiche bien la différence entre `a` et `b`.

QUESTION 18 ► Pour illustrer la conversion de type, exécutez le programme ci-dessous, et indiquez pour quelle raison le résultat n'est pas correct, c'est à dire quelle règle de conversion a été appliquée.

```
#include <stdio.h>

int
main()
{
    long a = 8000000002;
    float b = 2;
    long c = a / b;
    printf("res : %li\n", c);
}
```

QUESTION 19 ► Allez encore une question sur la conversion des types entiers pour être sûr que vous avez compris. Quelle conversion de type va effectuer le compilateur si vous demandez l'addition entre un `float f` et un `long l` ?

QUESTION 20 ► Allez encore une question sur la conversion des types entiers, pour être vraiment sûr. Quelle conversion de type va effectuer le compilateur lors de l'affectation suivante : `int i = 1.618` ?

3 Conversion explicite de type

QUESTION 21 ► Un *type* est finalement une indication au compilateur sur la façon dont peut et doit être manipulée une donnée. Jsuqu'à présent, vous n'avez indiqué le type d'une donnée que lors de la déclaration d'une variable. Or, il est possible de d'indiquer un type à tout opérande d'une expression. Ceci s'appelle un *cast* explicite, ou *conversion de type* explicite, ou forcée. Vous pouvez convertir n'importe quel type entier dans un autre en indiquant `(<type>)` avant l'opérande. Les valeurs des éventuelles variables dont le type est convertie ne change pas.
Une conversion de type forcée est parfois *exigé* par le compilateur quand les informations peuvent être perdues durant la conversion, ou quand la conversion peut échouer pour d'autres raisons.

QUESTION 22 ► Regardez comme ça se passe bien :

- Convertissez un `char` valant 42 dans un `int` puis affichez ce dernier.
- Convertissez un `int` valant 42 dans un `char` puis affichez-le. Le code de formattage d'un `char` est `%c`.

QUESTION 23 ► Faut quand même faire attention à ce qu'on fait :

- Convertissez un `int` valant -42 dans un `unsigned int` puis affichez-le. Le code de formattage d'un `unsigned int` est `%u`.
- Convertissez (explicitement) une variable de type `int` valant 0x2a3b vers une variable de type `char` puis affichez-le. Quelle partie de l'entier est conservée ?

Une conversion de type explicite donne le message suivant au compilateur : "je sais ce que je fais". Donc sachez ce que vous faites à chaque fois que vous effectuez une telle conversion ou prenez vos précautions et testez précautionneusement.

QUESTION 24 ► Compilez puis corrigez le programme ci-dessous à l'aide d'une conversion de type explicite.

```
char A=3;
int B=4;
float C = A/B;
```

4 Opérations bits-à-bits

Passons un peu de temps à manipuler les bits constituant d'une valeur entière.
Considérez un foyer de jeunes travailleurs, avec piscine et jacouzi, possédant 32 chambres. Chacune peut être libre ou occupée. Pour stocker cette information, vous allez utiliser un entier dont le *i*-ème bit indique si la chambre *i* est vide (bit à 0) ou occupée (bit à 1). La série de question ci-dessous vous guide dans l'implémentation d'une petite bibliothèque de fonctions permettant au logiciel de gestion du foyer de tenir à jour quelles chambres sont vides, lesquelles sont occupées. Pour cela, il va vous falloir utiliser certains `opérateurs bits-à-bits` :

shift (*décalage* en français) à droite : `>>`, à gauche : `<<`. Exemple : `a = b << 5;`.

xor (*ou exclusif* en français) : `^`.

complément à 1 : `~`. Exemple : `~0b0101` vaut `0b1010`.

and (nan, je traduirai pas) se fait grâce à l'esperluette : `&`. Exemple : `0b0111 & 0b1101` vaut `0b0101`

or : `|`. Exemple : `0b1000 | 0b0001` vaut `0b1001`.

Attention, les opérateurs `&&`, `||` et == sont des opérateurs implémentant respectivement le “et”, le “ou” et l'égalité *logiques*.

QUESTION 25 ► Implémentez la fonction `void printRoomsState(uint32_t allRooms)` qui affiche l'état de toutes les chambres. N'hésitez pas à vous en servir dans la suite pour vérifier vos implémentations ; si vous préférez utiliser gdb, `print /t <valeur>` vous donnera une valeur en format binaire.

QUESTION 26 ► Implémentez la fonction `uint32_t roomGoesOccupied(uint32_t allRooms, unsigned int roomN` qui marque comme occupée la chambre dont le numéro est passée en paramètre, et renvoie l'état de toutes les chambres.

QUESTION 27 ► Implémentez la fonction `bool isOccupied(uint32_t allRooms, unsigned int roomNum);` qui renvoie *vrai* si la chambre dont le numéro est passé en paramètre est occupée, *faux* sinon.

QUESTION 28 ► Implémentez la fonction `uint32_t roomGoesEmpty(uint32_t allRooms, unsigned int roomNum)` qui marque comme vide la chambre dont le numéro est passée en paramètre, et renvoie l'état de toutes les chambres.

5 Créer un type

Un truc tout bête mais hyper méga pratique c'est de créer un type, un type C, grâce à `typedef`. Par exemple la ligne `typedef unsigned int size_t;` crée le type `size_t`. Ce nouveau type n'est ensuite qu'un synonyme de `unsigned int`, mais c'est pas pareil : il est clair lorsque vous manipulez une variable de type `size_t` qu'elle doit être manipulée comme une taille (elle ne peut être négative par exemple). Et en plus c'est + concis.
D'autres fois, il est utile de créer un type dit *opaque* : l'utilisateur (d'une bibliothèque par exemple) n'a pas forcément à connaître sa définition précise, la bibliothèque pourra la changer sans que le programme l'utilisant doive changer.
Pour reprendre l'exemple du type booléen : le type `bool` est définie par un `typedef` dans le fichier d'entête `stdbool.h`.

QUESTION 29 ► Créez un type `ensemble_t` (le suffixe `_t` est souvent utilisé pour tous les types créés grâce à `typedef`) pour l'ensemble des chambres de l'hôtel considéré précédemment.

QUESTION 30 ► Le foyer pour jeunes travailleurs s'agrandit ! Une extension pour réfugiés va être construite, contenant une vingtaine de chambres. Votre bibliothèque de gestion de l'ensemble des chambres doit donc évoluer. Proposez une nouvelle définition pour le type `ensemble`.

QUESTION 31 ► Modifiez votre bibliothèque de fonctions pour qu'elle fonctionne avec votre nouvelle défition du type `ensemble`

6 Énumérations

Une *énumération* est un *type* entier que vous pouvez créer. Ce type ne pourra prendre que certaines valeurs entières. Les exemples suivants montrent différentes manière de définir un type de scrutin :

```
enum scrutin_e {MAJORITAIRE_UNI, MAJORITAIRE_PLURI, PROPORTIONNEL, MIXTE};
// ou
enum scrutin_e {MAJORITAIRE=2, MAJORITAIRE_PLURI=8, PROPORTIONNEL=42, MIXTE=12};
// ou
enum scrutin_e {MAJORITAIRE=4, MAJORITAIRE_PLURI, PROPORTIONNEL, MIXTE};

typedef enum scrutin_e scrutin_t;
```

Puis vous pouvez déclarez une variable de ce type : `scrutin_t presidentiel = MAJORITAIRE_UNI;`. Vous pouvez afficher sa valeur entière : `printf("%d", presidentiel);`

QUESTION 32 ► Pour chacune des façon de définir une énumération, affichez les valeurs entières correspondant à chacune des valeurs que peut prendre un `scrutin_t`.

QUESTION 33 ► Proposez une énumération pour les jours de la semaine.

7 “Switch-case” pour conversion shadok

QUESTION 34 ► Exécutez pas à pas le programme ci-dessous et indiquez pourquoi il est erroné

```
void
appreciate()
{
  switch (note) {
    case 12:
        printf("12 ! Passable ");
        break;
    case 18:
        printf("18 ! Super ! ");
    case 6:
        printf("6 ! Naze ");
        break;
    case 8:
        printf("8 ! Mieux que naze ");
        break;
    case 10:
        printf("10 ! Presque passable ");
        break;
    default:
        printf("%d ! Pas prevu par le de-qui-corrige", note);
        break;
    }
    printf("\n");
}

void
main()
{
  appreciate(6);
  appreciate(12);
  appreciate(18);
  appreciate(14);
}
```

Pour plus de détail sur les switch-case, voyez `http://blog.robertelder.org/switch-statements-statement-expressions/`

QUESTION 35 ► Vous allez écrire un programme qui affiche un nombre à la manière des shadoks, c’est à dire en base 4 :

- "Ga" pour 0
- "Bu" pour 1
- "Zo" pour 2
- "Meu" pour 3
- "Bu Ga" pour 4
- ...
- "Zo Meu" pour 11
- ...

Vous utiliserez une boucle à l’intérieur de laquelle vous placerez un switch-case pour tester le reste de la division euclidienne. Si vous avez des problèmes à trouver l’algorithme qui permet de faire cela, interrogez votre prof. Ce n’est pas grave si votre programme affiche le bon résultat à l’envers ("Meu Zo" au lien de "Zo Meu").

QUESTION 36 ► Testez votre programme sur les nombres 3, 6, 11, 42

Chapitre 2

Pointeurs, tableaux, allocation statique, fonction main paramétrée

Où l'étudiant, déjà, référencera et déréférencera à tout va.

1 Je déréférence ou je pointe ?

Chaque variable d'un programme C possède une adresse. Une adresse est un entier correspondant à l'emplacement du premier octet de la mémoire occupée par la variable. Un *pointeur* est une variable contenant une adresse.
L'opérateur `&` renvoie l'adresse de l'objet indiqué à sa suite. Un *objet* est un emplacement mémoire identifiable. Vous pouvez par exemple demander l'adresse de n'importe quel symbole : variables locales, variables globales, fonction... Exemple : `&aVariable;`. Il s'agit du *référencement* d'une variable. Mais vous ne pouvez pas demander l'adresse de l'expression `5 + 1`, ça n'a pas de sens en C.

QUESTION 1 ▶ Écrivez un programme qui affiche l'adresse de votre fonction `main()`. Dans quelle section de la mémoire (Figure 2.1) se trouve cette fonction ?

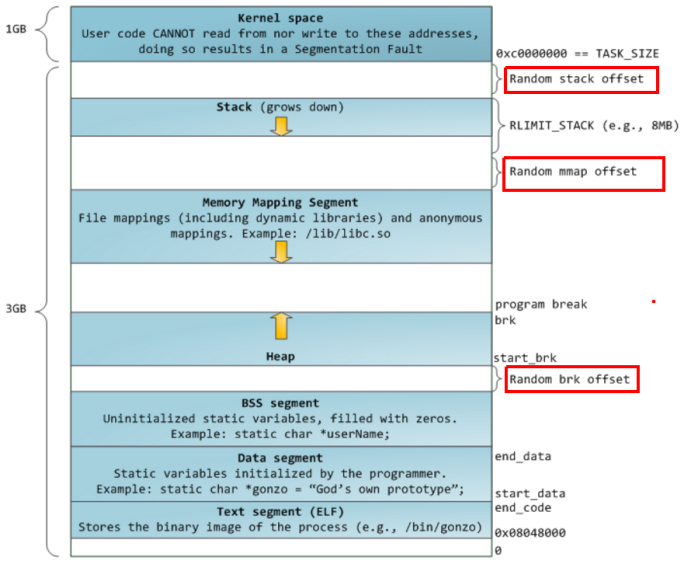


FIGURE 2.1 – Espace d'adressage d'un processus sous Linux et sur une architecture de type i386 (32 bits)

QUESTION 2 ▶ Écrivez un programme qui :

- affecte à une variable globale `intNotInitialized` non initialisée| la valeur 42
- affecte à une variable globale `intInitialized` initialisée à zéro la valeur 5
- affiche les valeurs et l'adresse de ces variables ainsi que l'adresse de la variable.

À quelle section de la Figure 2.1 chacune de ses variable appartient-elle ? Vérifiez que ces adresses ainsi que l'adresse de la fonction `main()` sont ordonnées correctement, c'est à dire qu'une adresse appartenant à la section `.text` n'est pas supérieure à une autre.
Une adresse peut être stockée dans une variable mais cette variable doit avoir un type adéquat. Ce type est indiqué par le type de l'élément en mémoire suivi du caractère `*`. On l'appelle alors un *pointeur*. Exemple de déclaration d'un pointeur : `int* adress;`. Exemple d'affectation de ce pointeur : `adress = &aVariable`

QUESTION 3 ► Modifiez votre programme pour que les adresses des variables globales soit stockées dans des variables locales avant affichage.
S'il est possible de référencer une variable, il est possible de *déréférencer* un pointeur, grâce à l'opérateur `*`. Exemple : `*mavariable`. `*v` peut être interprété comme « contenu de l'adresse mémoire pointée par v ».

QUESTION 4 ► Modifiez votre programme pour que, après les affichages déjà effectués, il modifie et affiche le contenu des deux variables `intNotInitialized` et `intInitialized` grâce au déréférencement des deux pointeurs correspondant.

QUESTION 5 ► Modifiez votre programme pour que les variables globales soient maintenant locales. Dans quelle partie de la mémoire sont stockée ces variables locales ?

2 Tableaux

On déclare un tableau de la manière suivante : `<type> variable[<taille>];` Exemple, un tableau de 3 entiers : `int mesentiers[3];`
Deux manières d'initialiser les valeurs d'un tableau :

```
int mesentiers[3];
mesentiers[0] = 7;
mesentiers[1] = 21;
mesentiers[2] = 63;
```

ou

```
int mesentiers[3] = {7, 21, 63};
```

Tableaux de table variable

Dans la norme C99, un tableau pouvait être de tableau variable, ce qui correspond au cas du tableau `all_elements` ci-dessous :

```
#include <stdlib.h>

int
main(int argc, char* argv[])
{
  int i = 0;
  int number_of_elements = atoi(argv[1]);
  int all_elements[number_of_elements];

  while (i < number_of_elements) {
    all_elements[i] = 42;
  }
}
```

MAIS la norme C11 les a rendu optionnels, ce qui fait qu'un compilateur C respectant la norme C11 n'est pas tenu de les supporter. La raison, il me semble, c'est que placer des objets de taille arbitrairement grande peut mener à des dépassements de pile non anticipés.

QUESTION 6 ► Bien qu'utiliser des tableaux de taille variable ne soit pas une bonne idée (voir l'encadré) on va, pour une fois, utiliser les tableaux de taille variable pour comprendre des trucs.
`sizeof` est un *opérateur*, au même titre que +, -, * etc. Si la taille de l'opérande est connue à la compilation, le compilateur remplace le `sizeof(<operand>)` par la taille. Sinon , le compilateur génère les instructions qu'il faut pour que, à l'exécution la taille soit calculée.

En vous inspirant du programme donné dans l'encadré sur les tableaux de taille variable, écrivez un programme vous permettant de savoir si `sizeof`, appliqué à un tableau (type ou variable) renvoie le nombre d'éléments du tableau, le nombre d'octets occupés par ce tableau, ou le nombre d'octets occupés par l'adresse du tableau.

QUESTION 7 ► Définissez un type pour des matrices de taille `MAT_SIZE x MAT_SIZE`. Deux choses à observer :

1. Comment déclarer un nouveau type tableau, (perturbe parfois les gens) ;
2. Comment déclarer `MAT_SIZE`. Essayez via une variable globale de type `unsigned int` puis `const unsigned int`, puis grâce à un `#define`.

QUESTION 8 ► Implémentez une procédure `affiche_matrice()` qui prend en paramètre une matrice et l'affiche (ligne par ligne). Testez là.

3 Chaînes de caractères

Une chaîne de caractères, en C, est simplement un tableau rempli de caractères ASCII, plus précisément un tableau de `char`, terminé par le caractère de code 0, c'est à dire `'\0'`.

QUESTION 9 ► Dans un main(), affectez à une variable `my_string` la chaîne de caractère `"<-_->"`.

QUESTION 10 ► Dans votre main parcourez cette chaîne de caractère et affichez-les sur la sortie standard séparés par des '.'

QUESTION 11 ► Modifiez votre programme pour qu'il utilise la fonction `size_t strlen(const char *s);` qui renvoie la taille de la chaîne passée en paramètre.

QUESTION 12 ► Pour lire une chaîne de caractère sur l'entrée standard, la fonction int scanf(const char *format, ...) ; `#include <stdio.h>` Écrivez un programme qui convertit un paramètre pris sur l'entrée standard en `int` grâce à la fonction `strtoimax` (`man strtoimax` pour les détails !) et en retourne le reste de la division euclidienne par 4.

QUESTION 13 ► La fonction `int strcmp(const char *s1, const char *s2);`, utilisable en incluant `<string.h>`, est une fonction (de la librairie C) qui compare les deux chaînes passées en paramètre et renvoie 0 si elles sont égales, un entier infèrieur à 0 si `s1` est plus petite que `s2`, un entier supérieur à 0 sinon. Implémentez `int TP_strcmp(const char *s1, const char *s2);` qui fait la même chose (et interdiction d'inclure `<string.h>` ! je vous vois venir). Retournez une valeur dès que le code ASCII du *i*-ème caractère de `s1` est infèrieur/supérieur au *i*-ème de `s2`.

4 Équivalence pointeur / tableau

Tableau ou pointeur, la différence n'est que syntaxique. Dans certains cas il est plus clair, pus lisible, d'utiliser l'un que l'autre c'est tout. Il y a tout de même 2-3 subtilités.

QUESTION 14 ► Affichez l'adresse pointée par `tab_addr, tab_point, tab_point, elt` à la suite des instructions ci-dessous.

```
int tab[5][7];
int* tab_addr = &tab;
int* tab_point = tab;
int* tab_point_elt = &tab[0];
```

QUESTION 15 ► Écrivez un petit programme vous permettant de savoir quel opérateur est le plus prioritaire entre `[]` et `*`.

QUESTION 16 ► L'expression `*(tab_point_elt + 12)` permet-elle d'accèder à `tab[1][5]` ou à `tab[2][2]` ?

5 Structures

On peut en C, créer des types de données non entiers, les *structures*. On met ce qu'on veut dedans du moment que le type de ce qu'on met dedans est connu.

QUESTION 17 ► On veut créer un type permettant de stocker la date naissance de quelqu'un. Proposez une structure pour cela ; stockez

QUESTION 18 ► On veut créer un type permettant de stocker l'identifiant (entier) d'un étudiant, sa date de naissance le nom de sa boisson préférée (sur 20 caractères max). Proposez une structure pour cela

QUESTION 19 ► Déclarez un tableau de 3 `students fetards`

6 Retour sur le cast

Rappelez-vous : un *cast* est une indication, pr le programmeur, à destination du compilateur, qu'une donnée est à manipuler comme un type donné. Vous n'avez pour l'instant effectuez des conversions de type entier mais cette notion est applicable aux autre types : pointeurs de n'importe quoi, tableaux, structures.

7 Arithmétique de pointeurs

Petit manuel ultra-rapide d'utilisation de GDB

1. Compiler avec l'option `-g`
2. Lancer gdb : en ligne de commande, ben... `gdb <exec_file>`
3. Indiquer que gdb doit s'arrêter quelque part :
   - s'arrêter à l'entrée d'une fonction : `break <function_name>`. Exemple : `break main`
   - `break <filename.c>:<line_num>` pour s'arrêter à une ligne de code donnée d'un fichier C
4. Démarrer l'exécution du programme : `run`
5. `step` exécute la prochaine ligne du programme et descend dans les appels de fonctions s'il y en a.
6. `next` exécute la prochaine ligne du programme sans descendre dans les appels de fonctions.
7. `cont` reprend l'exécution jusqu'au prochain point d'arrêt ou jusqu'à la fin du programme
8. `print <expression>` pour afficher un truc. Exemple : `print monint + 1`. Un truc pratique c'est de préciser le format d'affichage, par exemple hexadécimal : `print/x ''un truc''`.
9. `watch <mem_expr>` indique à gdb de s'arrêter quand le contenu de la mémoire donné par l'expression change. Exemple : `watch mavariable`
10. Pour une interface plus agréable : tapez `layout src` puis `focus cmd`
11. donner à gdb des instructions à exécuter depuis un fichier lorsqu'il démarre : `gdb -x file.gdb <exec>`. Par exemple, plus besoin de taper `break main` puis `layout src` puis `focus cmd` puis `run` à chaque début de session debug si vous mettez ces 4 instructions dans le fichier `file.gdb`.

QUESTION 20 ► Soit le programme suivant :

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Sur papier (sans programmer pour l'instant), notez le résultats que vous attendez des expressions suivantes :

- `*P+2`
- `*(P+2)`
- `&P+1`
- `&A[4]-3`
- `A+3`

— `&A[7]-P`

— `P+(*P-10)`

— `*(P+*(P+8)-A[7])`

QUESTION 21 ► Lisez l'encadré sur GDB puis à l'aide de GDB stoppé juste après `P=A;`, vérifiez vos réponses.

QUESTION 22 ► Grâce à la question précédente, vous devriez être en mesure de répondre à ces questions : 1) Quel est l'opérateur prioritaire entre `*` et `+` ? 2) Quel est l'opérateur prioritaire entre `&` et `+` ?

8 Fonction `main()` paramétrée

Le point d'entrée d'un programme C est la fonction `main()`, et vous savez sûrement que cette fonction se déclare comme ceci : `int main(int argc, char* argv[])`. Armés de vos nouvelles connaissances concernant les pointeurs, vous êtes maintenant en mesure d'appréhender le type de `argv`.

QUESTION 23 ► Affichez le nombre de paramètres tous les paramètres passés à votre fonction `main()`.

QUESTION 24 ► À quoi correspond le premier paramètre (indice 0 du tableau `argv`) ?

QUESTION 25 ► Affichez *x* fois le caractère '*', où *x* est passé en paramètre. Vous devrez pour cela utilisez la fonction `strtol()` de la libC. L'explication, avec des exemples est donnée grâce à `man strtol()`.

9 r-value et l-value

lvalue and rvalue

Un *objet* est un emplacement mémoire identifiable (par exemple une variable locale `int locale`). Contre-exemple : le résultat d'une opération arithmétique n'est pas un objet. Une *l-value* est une expression identifiant un objet ; elle a donc un emplacement mémoire identifiable. Elle peut donc être affectée. Par exemple : `locale = 4`. Une l-value peut donc être :

- le nom d'une variable (de n'importe quel type)
- un emplacement d'un tableau (via l'opérateur `[]`)
- un déréférencement (via l'opérateur `*`)
- un accès à un membre d'une structure
- une l-value entre parenthèse

Une *r-value* est une valeur qui ne persiste pas après son usage dans une expression. Par exemple la valeur 4. Elle n'a pas d'emplacement mémoire identifiable.
Une l-value peut être utilisée comme une r-value mais l'inverse n'est pas vrai !

QUESTION 26 ► Pour illustrer les concepts de l-value et r-value, pour chacun des programmes ci-dessous, indiquez s'il est correct ou non, puis vérifiez en l'exécutant.

— `foo() = 2;`

— `int var = 0; 4 = var;`

— `int var = 0; (var + 1) = 4;`

—
```
int arr[] = {1, 2};
int* p = &arr[0];
*(p + 1) = 10;
```

—
```
int var = 10;
int* addr = &(var + 1);
```

—
```
enum color { red, green, blue };
color c;
c = green;
blue = green;
```

— `&var = 40;`

10 Pointeurs de fonction

QUESTION 27 ► Écrire une fonction `fois_deux` qui multiplie par deux un entier

QUESTION 28 ► Écrire une fonction `appliquer_tableau(int pfunc(int), int* tab, int size)` qui applique une fonction `pfunc` au tableau `tab` de taille `size`.

QUESTION 29 ► Écrire une fonction main qui teste `appliquer_tableau`

QUESTION 30 ► Voici un algorithme de tri des éléménts d'un tableau dans l'ordre croissant :

```
void
tri_croissant(int* t, int n)
{
  int i, i_min, j, tmp;

  for(i=0; i < n - 1; i++) {
    i_min=i;

    for(j=i; j<n; j++) {
      if(t[j] < t[i_min]) {
        i_min = j;
      }
    }
    tmp = t[i_min];
    t[i_min] = t[i];
    t[i] = tmp;
  }
}
```

Voici un algorithme de tri des éléménts d'un tableau dans l'ordre décroissant :

```
void
tri_decroissant(int* t, int n)
{
  inti, i_max, j, tmp;

  for(i=0 ; i < n-1 ; i++) {
    i_max=i;
    for(j=i; j<n; j++) {
      if(t[j]>t[i_max]) {
        i_max=j;
      }
    }
    tmp = t[i_max];
    t[i_max] = t[i];
    t[i] = tmp;
  }
}
```

Ces deux algorithme sont identiques, à l'exception de l'opérateur de comparaison. Afin d'éviter de dupliquer du code, nous allons utiliser les pointeurs de fonctions.

— Écrire une fonction `superieur(int a, int b)` qui renvoie 1 si a est supérieur à b , 0 s'ils sont égaux et -1 sinon

— Écrire une fonction `inferieur(int a, int b)` qui renvoie 1 si a est inférieur à b , 0 s'ils sont égaux et -1 sinon

— Écrire une fonction `tri`, qui tri un tableau `t` de taille `size` selon une fonction `compare`

— Écrire une fonction `main` qui teste `tri`

11 Jouons aux cartes

On veut programmer un jeu permettant de jouer à deux joueurs à un jeu de cartes dont le principe est le suivant. Au début de la partie, chaque joueur reçoit 10 cartes puis à chaque tour de jeu :

— un nombre de carreaux de chocolat artisanal bio équitable solidaire est mis en jeu par un généreux donateur

— chaque joueur choisit une carte qui sera confrontée à celle de l'autre joueur

— le joueur ayant la carte de plus haute valeur remporte

Le but est de remporter le plus de carreaux pour faire une fondue.

QUESTION 31 ► Proposez une énumération pour les couleurs d'une cartes (coeur, carreau, pique, trèfle).

QUESTION 32 ► Proposez une énumération pour la valeur d'une carte (sept, huit, neuf, dix, valet, dame, roi, as).

QUESTION 33 ► Déclarez maintenant une structure `struct carte_s` pour une carte.

QUESTION 34 ► Déclarez une variable globale `jeu_complet` de type tableau contenant les 32 cartes dont on aura besoin pour jouer, puis implémentez la fonction `void init_jeu()` qui l'initialise.

QUESTION 35 ► Écrivez une fonction `void affiche_carte(struct carte_s carte)` qui permet d'afficher une carte. Remarque : pour simplifier la tâche, on pourra déclarer les deux tableaux suivants :
`char* chaines_couleur[4] = {"Coeur", "Carreau", "Pique", "Trefle"};`
`char* chaines_valeur[8]={"7", "8", "9", "10", "Valet", "Dame", "Roi", "As"};`

QUESTION 36 ► Écrivez une fonction `void distribue_main(struct carte_s* jeu)` qui prend en argument un jeu de carte et deux tableaux de 10 cartes et qui initialise ceux-ci avec 10 cartes prises au hasard dans le jeu. Rappel : on peut tirer un entier au hasard entre 0 et une valeur donnée avec la fonction `random()` déclarée dans l'en-tête <stdlib.h> :

```
#include <stdlib.h>
...
srandom(time(NULL)); /* initialise la graine d'aléa  */
int nb_entre_0_et_42  = random() % 42;
```

Pour enlever une carte du jeu chaque fois que vous en tirez une, vous pouvez simplement copier des zéros à la place (attention, arrangez-vous pour ne pas autoriser votre programme à tirer une telle "carte"...).

QUESTION 37 ► Implémentez le jeu : vous affichez le jeu de l'utilisateur sur la sortie standard ; vous demandez à l'utilisateur de choisir quelle carte il veut jouer ; l'ordinateur joue au hasard.

12 C'est noël !

QUESTION 38 ► C'est noël ! Écrivez un programme permettant de dessiner un sapin en fixant la hauteur du cône, la largeur et la hauteur du tronc. Par exemple, pour un cône de hauteur 6, un tronc de hauteur 2 et de largeur 3 on doit obtenir un affichage du type :

```
     *
    ***
   *****
  *******
 *********
***********
    |||
    |||
```

QUESTION 39 ► Maintenant, l'utilisateur du programme peut, en ligne de commande, spécifier un ÉTAGE et une place à cet étage où placer une boule de noêl. Si l'utilisateur spécifie 2 3, le sapin précédent sera redessiné comme ceci :

```
     *
    ***
   *****
  *******
 **0******
***********
    |||
    |||
```

QUESTION 40 ► Écrivez maintenant une fonction `generate_pine()` qui affiche un sapin dont la taille est aléatoire et les boules sont placées aléatoirement.

QUESTION 41 ► Proposez une structure pour un sapin de noël.

QUESTION 42 ► Implémentez la fonction `void populate_forest()` qui prend en paramètre un pointeur vers un sapin et qui l'initialise aléatoirement.

QUESTION 43 ► Implémenter la fonction `print_pine()` qui prend en paramètre un pointeur vers un sapin et qui affiche ce sapin.

QUESTION 44 ► Déclarez un tableau de 8 sapins de noël `forest`, puis initialisez ces sapines grâce à `populate_forest()` puis affichez-les grâce à `print_pine`.

Chapitre 3

Entrées-sorties

1 Language de bois

Le président de la République cherche un nouveau directgeur de la communication. Vue la réputation de l'INSA de Lyon et les cours qui y sont prodigués, son équipe a tout naturellement pensé à vous et, plein de bonne volonté, ou plus vraisemblablement pensant à votre carrière, vous avez accepté. Après quelques temps à écrire les discours, les réponses aux interviews truquées, et les gazouillis [1], vous vous êtes vaguement rappelés que beaucoup beaucoup de tâches sont automatisables à l'aide d'un programme informatique. Vous avez donc décidé d'implémenter un un générateur de langue de bois.
Pour cela vous allez avoir besoin des fonctions suivantes :

- `FILE *fopen(const char *pathname, const char *mode);`
- `int fclose(FILE *stream);`
- `int fgetc(FILE *stream);`
- `char *fgets(char *s, int size, FILE *stream);`
- `int fputc(int c, FILE *stream);`
- `int fputs(const char *s, FILE *stream);`

Pour les utiliser, vous devez inclure `<stdio.h>`, et lire à quoi elles servent grâce à la commande `man` (dans un terminal). Exemple : `man fgets`.
Deux petits détails :

- Vous verrez dans la documentation des références au symbole `EOF`. C'est juste un entier défini dans la librairie `stdio` et utilisé
- Si vous voulez écrire (resp. lire) sur la sortie standard (resp. entrée standard), vous pouvez utiliser le symbole `stdout` (resp. `stdin`), de type `FILE*`.

QUESTION 1 ► Lisez le fonctionnement de `fgets()` grâce à `man`. Quels sont les 3 cas qui conduisent cette fonction à arrêter de lire dans le fichier et à vous retourner un résultat ?

QUESTION 2 ► Commencez par écrire un programme qui parcourt un fichier texte et l'affiche sur la sortie standard. Vous pouvez lire/écrire depuis/dans la sortie standard exactement comme dans un fichier grâce aux descripteurs de fichiers `stdin` et `stdout` (ou utiliser `printf()`)

QUESTION 3 ► Votre générateur de langue de bois va implémenter le tableau 1. Sur Moodle vous trouverez, grâce à un généreux étudiant, le texte de la case cases de la colonne 1 dans un fichier `colonne1.txt` (1 case par ligne), celui de la colonne 2 dans un fichier `colonne2.txt` etc. Écrivez maintenant un programme qui parcourt chacun de ces 4 fichiers et copie une ligne du fichier prise au hasard sur la sortie standard.

QUESTION 4 ► Adaptez maintenant votre programme pour qu'il génère un petit discours de 5 phrases dans un fichier appelé `discours.txt`. N'oubliez pas d'appeler `fclose()` quand vous n'avez plus besoin d'accéder à un fichier (c'est surtout vrai en écriture bien sûr...).

1. `https://fr.wikipedia.org/wiki/Tweet`

1	2	3	4
Mesdames, messieurs,	la conjoncture actuelle	doit s'intégrer à la finalisation globale	d'un processus allant vers plus d'égalité.
Je reste fondamentalement persuadé que	la situation d'exclusion que certains d'entre vous connaissent	oblige à la prise en compte encore plus effective	d'un avenir s'orientant vers plus de progrès et plus de justice.
Dès lors, sachez que je me battrai pour faire admettre que	l'acuité des problèmes de la vie quotidienne	interpelle le citoyen que je suis et nous oblige tous à aller de l'avant dans la voie	d'une restructuration dans laquelle chacun pourra enfin retrouver sa dignité.
Par ailleurs, c'est en toute connaissance de cause que je peux affirmer aujourd'hui que	la volonté farouche de sortir notre pays de la crise	a pour conséquence obligatoire l'urgente nécessité	d'une valorisation sans concession de nos caractères spécifiques.
Je tiens à vous dire ici ma détermination sans faille pour clamer haut et fort que	l'effort prioritaire en faveur du statut précaire des exclus	conforte mon désir incontestable d'aller dans le sens	d'un plan correspondant véritablement aux exigences légitimes de chacun.
J'ai depuis longtemps (ai-je besoin de vous le rappeler ?), défendu l idée que	le particularisme dû à notre histoire unique	doit nous amener au choix réellement impératif	de solutions rapides correspondant aux grands axes sociaux prioritaires.
Et c'est en toute conscience que je déclare avec conviction que	l'aspiration plus que légitime de chacun au progrès social	doit prendre en compte les préoccupations de la population de base dans l'élaboration	d'un programme plus humain, plus fraternel et plus juste.
Et ce n'est certainement pas vous, mes chers compatriotes, qui me contredirez si je vous dis que	la nécessité de répondre à votre inquiétude journalière, que vous soyez jeunes ou âgés,	entraîne une mission somme toute des plus exaltantes pour moi :	l'élaboration d'un projet porteur de véritables espoirs, notamment pour les plus démunis

TABLE 3.1 – Commencez avec n'importe quelle case de la colonne 1 puis n'importe laquelle en colonne 2, puis colonne 3 puis 4.

2 Printf(), on l'implémente !

Vous allez implémenter votre version de `printf()`, intitulée `understood_printf()`. Le produit fini ne va servir à rien, comme tous les TPs, mais vous aurez compris plusieurs trucs, comme après la majorité des TP.
D'abord, la table 3.2 vous donne les codes de formattage indiquant comment doivent être représentées les paramètres de printf. C'est la vraie table, vous pourrez vous y référer pour future utilisation de `printf()`.

Famille de type	Code de conversion	Type précis
	%d (or %i)	(signed) int
	%u	unsigned int
	%o	int in octal
Integers	%x, %X	int in hexadecimal (%X uses uppercase A-F)
	%hd, %hu	short, unsigned short
	%ld, %lu	long, unsigned long
	%lld, %llu	long long, unsigned long long
	%f	float in fixed notation
	%e, %E	float in scientific notation
Floating-point	%g, %G	float in fixed/scientific notation depending on its value
	%f, %lf (printf), %lf (scanf)	double : Use %f or %lf in printf(), but %lf in scanf().
	%Lf, %Le, %LE, %Lg, %LG	long double
Character	%c char	
String	%s	string

TABLE 3.2 – Codes de formattage de chaîne de caractère

Ensuite, la fonction `printf()` contient un nombre variable de paramètres. Une telle fonction est dite *variadique* (*variadic* in english) et se déclare comme ceci : `void understood_printf(char *to_be_printed, ...)`. Pour accéder aux paramètres, à l'intérieur de cette fonction, la libc met à disposition (incluez `stdarg.h`) quelques outils :

- Le pointeur `va_list` est un type "pointeur sur paramètre". Vous allez déclarer au début de votre fonction : `va_list param_pointer;`
- un appel à `va_start()` commence à parcourir la liste (chaînée) d'arguments en paramètre de la fonction. Typiquement : `va_start(param_pointer, to_be_printed);`
- ensuite, chaque appel à `type va_arg(va_list ptr, type)` renvoie le paramètre courant pointé par le pointeur donné et passe au paramètre suivant. Typiquement : `va_arg(param_pointer)`.
- un appel à `void va_end(va_list ptr)` termine le parcours.

Enfin, pour afficher des chaînes de caractères, vous allez devoir utiliser une fonction qui sait afficher... 1 caractère. Celle-ci, on est trop sympa, on vous la donne ci-dessous :

```
#include <unistd.h>

void putCh(char c)
{
    write(STDOUT_FILENO, &c, 1);
}
```

QUESTION 5 ► La fonction `understood_printf()` vous est donnée ci-dessous.

```
void
understood_printf(char *yet_to_be_printed, ...)
{
    va_list param_pointer;

    va_start(param_pointer, yet_to_be_printed);

    for (; *yet_to_be_printed != 0 ; yet_to_be_printed++)
    {
        if (*yet_to_be_printed == '%') {
```

```
            yet_to_be_printed++;
            format_t formatting_code;
            formatting_code = get_formatting_code(yet_to_be_printed);
            printParam(param_pointer, formatting_code);
        } else {
            putCh(*yet_to_be_printed);
        }
    }
    va_end(param_pointer);
    putCh('\n');
}
```

Dans la suite, vous allez devoir implémenter les fonctions `void printParam(va_list param_pointer, format_e fo` et `format_e get_formatting_code(char* str)`. Commencez par :

1. déclarer ces 2 fonctions, avec un corps vide pour le moment ;
2. définir l'énumération `format_e` ;
3. comprendre ce bout de code en l'exécutant pas à pas dans GDB. Notamment, assurez-vous de comprendre ce que représente `str` (et le contenu pointé) dans `get_formatting_code()` et `printParam()`.

QUESTION 6 ► Pour le moment, ne gérez que les code de formattage à 1 caractère. Implémentez `format_e get_formatting_code(char* str)`, qui analyse le code pointé par `str`, détermine si le paramètre courant doit s'afficher sous forme d'entier, de caractère, de chaîne etc., et retourn le bon `format_e` correspondant.

QUESTION 7 ► Vérifiez que votre fonction `get_formatting_code()` fonctionne correctement en la testant sur plusieurs exemples.

QUESTION 8 ► Implémentez maintenant `void printParam(va_list param_pointer, format_e format)`

QUESTION 9 ► Votre ne gère que les code de formattage à un caractère. Vous allez faire en sorte que `get_formatting_code()` sache aussi détecter les code à 2 caractères mais un problème va se poser : le pointeur `yet_to_be_printed` doit être avancé. Le plus simple, c'est que `get_formatting_code()` renvoie aussi cette information. Pour ce faire, on va changer son comportement ; le nouveau prototype de la fonction est maintenant : `char* get_formatting_code(char* str, format_e code)`. Elle renvoie la position dans la chaîne et remplit le contenu pointé par `code` avec le bon `format_e`.

Chapitre 4

Allocation de mémoire

Où l'étudiant découvrira les différentes manière d'allouer de la mémoire en C, de deux manières : statiquement et dynamiquement
Jusque là vous utilisiez de la mémoire allouée *statiquement* ; soit des variables gloables, soit des variables locales allouées sur la pile d'exécution. Voyez le schéma TODO pour vous rafraîchir la mémoire. Mais cela arrive souvent que vous, programmeur, ne sachiez pas à l'avance de combien de mémoire vous allez avoir besoin. Dans ce cas, votre programme va allouer de la mémoire *dynamiquement*, en fonction de ce qui se passe à l'exécution.
La manière portable d'allouer de la mémoire à un processus est d'utiliser la fonction `void* malloc(size_t size)` qui alloue dans le tas (le *heap* en anglais) `size` octets et renvoie un pointeur vers le premier de ces octets. Cette fonction appartient à la librairie C ; y faire appel ne réalise pas un appel système ; en tous cas pas à tous les coups ; vous comprendrez, on l'espère, en cours de TP.

1 Allouer

QUESTION 1 ► Implémentez un programme qui va chercher tous les mots passés en paramètres et les range dans un bout de mémoire alloué dynamiquement, puis les affiche. Pour copier chaque mot, utilisez la fonction `void *memcpy(void *dest, const void *src, size_t n)` qui copie à l'adresse pointée par `dest` les `n` premières octets pointés par `src`.

QUESTION 2 ► Maintenant, tant que l'utilisateur tape des mots, lorsqu'il tape 'entrée', votre programme doit les ajouter à la liste. Pour agrandir l'espace alloué au stockage des mots, utilisez `void *realloc(void *ptr, size_t size)` de `stdlib.h`, qui modifie la taille allouée à la zone pointée par `ptr` pour la positionner à `size`.

QUESTION 3 ► `realloc()` renvoie un type `void*`. C'est important car `realloc()` peut allouer une zone mémoire non consécutive à celle que vous lui avez passée en paramètre. Pourquoi ? Dans quels cas ? Prenez bien soin de vous servir du pointeur retourné.

QUESTION 4 ► Lorsque l'utilisateur tape '!', vous ré-initialisez la liste. Grâce à la fonction `void *memset(void *s, int c, size_t n)`, remplissez la zone mémoire que vous aviez allouée avec une valeur reconnaissable.

QUESTION 5 ► (facultatif) Maintenant implémentez la fonction `void *TP_memcpy(void *dest, const void *src, si` qui fait exactement la même chose que `memcpy(....)`.

2 Allocation Dynamique : listes chainées

Après votre passage au ministère dans l'équipe de communication, vous êtes ecœurés, et décidez que finalement tout ça n'a que peu de sens et virez révolutionnaire. Pour commencer, vous décidez d'écrire un petit programme d'analyse de discours. Pour cela, vous avez besoin d'un programme capable de parcourir un texte et ranger les mots dans une structure de données avant d'analyser celle-ci. Cette structure de données sera, pour le moment, une liste chaînée.
A priori, vous savez ce qu'est une liste chaînée. Si ça ne vous rapelle rien, expliquez-vous avec votre prof – ou votre ancien prof – d'algo puis demandez des détails à votre prof d'amphi ou à votre chargé de TD/TP.

QUESTION 6 ► Définissez la structure `word_s` qui permet de stocker un mot de taille variable, contenant donc une chaîne de caractère et la taille de celle-ci.

QUESTION 7 ► Définissez la structure `node_s` qui représente un maillon de la liste chaînée. Chaque maillon contient une valeur `value` et un pointeur vers le mot suivant. Pour la valeur portée par chaque nœud, utilisez un type `value_type_t` que vous définissez comme un `struct word_s`grâce à `typedef`.

QUESTION 8 ► Parcourez le discours stocké dans le fichier `discours.txt` et construisez la liste chaînée correspondant au discours. Pour cela, il vous faut implémenter la fonction `struct node_s* insert_node_end(struct node_s* list_head, value_type_t* value)` qui ajoute un nœud portant la valeur donnée à la liste dont le premier élément est pointé par `list_head`. Si `list_head` vaut NULL, alors le premier élément de la liste est alloué. Dans tous les cas, cette fonction renvoie un pointeur vers l'élément de la liste nouvellement alloué.

QUESTION 9 ► Implémentez la fonction `int getLength(struct node_s* list_head)` qui renvoie la taille de la liste pointée par `list_head`. Combien de mots contient le discours donné ?

QUESTION 10 ► Combien d'occurence du mot "je", du mot "nous", et du mot "vous" contiennent chacun de ces discours ?

QUESTION 11 ► Quels sont les mots d'au moins 5 caractères les plus utilisés ? Vous pouvez commencer par supprimer toutes les majuscules

QUESTION 12 ► Votre petit logiciel vous a attiré la sympathie de vos voisins ; vous montez maintenant une association de quartier. Vous aller faire évoluer votre petite bibliothèque de gestion de liste chaînée pour qu'elle puisse stocker toutes les informations realtives aux adhérents de l'asso, c'est à dire pour chacun d'eux :

- Nom ;
- Prénom ;
- Investissement parmi : relation avec la presse, relation avec la mairie, organisation des réunions mensuelles, maintenance du site web, organisation de la fête annuel de quartier ;
- Boisson préférée.

Il vous faut en premier lieu créer la structure `struct member_s`pour ces informations et redéfinir le type `value_t`.

QUESTION 13 ► Testez votre fonction sur les 3 adhérents suivants :

- Kevin Marquet, relation presse, Chouffe.
- Juan, organisation réunions, Clara
- Adamo Bocceli, maintenance site, San Pellegrino

QUESTION 14 ► Mais Juan change de boulot, et achète une villa dans un quartier bourgeois, il se désinscrit donc de l'asso. Implémentez donc :

- implémentez la fonction `bool are_same_people(struct member_s* a_member, struct member_s* an_othe` qui renvoie `true` si les deux membres portent les même valeurs.
- la fonction `struct node_s** remove_node()` qui prend en paramètre :
  - un pointeur `struct member_s` nommé `to_be_removed`
  - un pointeur de fonction du prototype de `bool are_same_people(struct member_s* a_member, struct m` Nommez ce paramètre `equals_fct`

  qui supprime de la liste le premier élément `current_node` pour lequel la fonction `equals_fct(value_to_be_removed, current_node)` renvoie `true.`, et qui renvoie la tête de la liste (éventuellement nouvelle).

QUESTION 15 ► (facultatif) Faites en sorte que l'insertion se fasse pas ordre alphabétique

QUESTION 16 ► Écrivez la fonction `struct_node_s* invert(struct node_s* list_head)` qui inverse la liste chaînée dont le premier élément est passé en paramètre et qui renvoie un pointeur vers le nouveau premier élément

QUESTION 17 ► Implémentez la fonction `void print_list(struct node_s* list_head)` qui affiche un par un sur la sortie standard les éléments de la liste passé en paramètre.

QUESTION 18 ► Que renvoie, à votre avis l'expression `******head` ? Vérifiez dans gdb et/ou en utilisant `printf()`

3 Libérer, délivrer

QUESTION 19 ► Que fait l'OS de (toute) la mémoire allouée à un processus lorsque ce processus se termine ? Pile d'exécution, variables globales, etc.

QUESTION 20 ► C'est pareil pour le tas, qui n'est qu'un bout de mémoire alloué par l'OS et la libc à un processus. Du coup, pour la majorité des programmes, libérer la mémoire allouée dynamiquement n'est pas très utile. MAIS ! Vous allez quand même systématiquement libérer, grâce à `free()` (par exemple `free(<monpointeur>)` une portion de mémoire allouée grâce à `malloc()` (par exemple `int* monpointeur = malloc(42);`). Donc modifiez votre programme précédent pour libérer, au fur et à mesure tous les bouts de mémoire allouée. Si c'est déjà fait, c'est cool, vous êtes beaux, vous êtes bons, changez rien (mais retenez pourquoi c'est import

QUESTION 21 ► Écrivez un programme `fat.c` dont l'exécution est plus lente ou ne finit pas si sa mémoire allouée dynamiquement n'est pas libérée dynamiquement.

QUESTION 22 ► Si vous ne l'avez pas fait, appelez `free()` quand nécessaire dans votre bibliothèque de gestion des listes chaînée.

4 malloc()

After having been using it, you are going to implement your version of `malloc()`, called `TP_malloc()`. You (should) remember that `malloc()` allows to allocate data on the heap. The principle is the following :

- the user asks for a memory block via a call to `malloc()` ;
- `malloc()` itself asks for a memory block to the kernel via `sbrk()`. To avoid the cost of this system call, `malloc()` asks a "big" memoy block to `sbrk()` and use only a fraction of it to satisfy the request of the user. A data structure is maintained inside the allocation library to keep track of the unused memory (see below) ;
- During future calls to `malloc()`, the remaining memory (that was obtained by `sbrk()`) is used ;
- the memory blocks used by `free()` are kept in the user memory address space and will be used also to satisfy future requests. The memory is never given "back" to the kernel before the end of the program execution.

The data structure you will use to keep track of the free memory inside your library is a linked list. Each node of the list contains a size, a pointer to the next block, and the memory space itself.
The function `void* sbrk(int incr);` increases the size of the heap by `incr` bytes and returns a pointer to the beginning of the newly allocated region.
When a call to `TP_malloc()` is performed, your library must walk through the linked list until it find a free block big enough. This algorithm is called *first-fit* (one chose the first that suits), by opposition to *best-fit*. If the free block found is exacly the size requested, it is removed from the linked list. If it is too big, the remainder of the block is kept in the linked list. And if no block of sufficient size has been found, your library must asks for the heap to be increased.

QUESTION 23 ► Which information must your library keep between two calls to `TP_malloc()` ?

QUESTION 24 ► Implement `TP_malloc()` and `TP_free()`.

5 Révisons un peu de tout : une bibliothèque d'accès bit-à-bit

La bibliothèque C standard offre de nombreuses fonctions d'accès à un fichier. Ce sont, entre autres, les fonctions fopen, fscanf, fprintf, ... dont les prototypes sont fournis dans stdio.h. Ces fonctions permettent d'ouvrir un ficher en lecture ou écriture puis d'y écrire ou lire des caractères, des entiers, des flottants, etc. Malheureusement, la bibliothèque standard n'offre pas de moyen de lire ou d'écrire un seul bit à la fois dans un fichier.
Le but de cette question est d'écrire une bibliothèque permettant d'effectuer des entrées/sorties bit-à-bit dans un fichier. Une telle bibliothèque pourrait se révéler utile pour implémenter des algorithmes de compression par exemple. Le problème qui se pose lors de la réalisation de fonctions d'accès bit-à-bit aux fichiers est que nous devons les réaliser en nous servant des fonctions standard pour lesquelles la taille minimale d'un accès est l'octet (8 bits).
Une réalisation possible pour les fonctions d'écriture de cette bibliothèque consisterait à maintenir un ensemble de bits en attente d'écriture (dans une variable, qui nous servirait ainsi de mémoire tampon) et, dès que ces bits sont au nombre de huit, de réaliser l'écriture de l'octet correspondant à l'aide d'une des fonctions de la bibliothèque standard. Pour les fonctions de lecture, nous pouvons utiliser la même idée en maintenant un octet lu à l'avance dans le fichier duquel nous extrayons les bits les uns après les autres au fur et à mesure des demandes.
Vous aurez besoin des fonctions suivantes pour lire/écrire dans/depuis le fichier :

- `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);`

QUESTION 25 ► Vous allez implémenter les fonctions de cette librairie dans un fichier `bblib.c`. Mettez à jour votre Makefile de manière à ce qu'il compile de manière séparée les fichiers `main.c` et `bblib.c`.

QUESTION 26 ► Choisissez une structure de données permettant de maintenir toutes les informations qui vous seront utiles pour accéder à un fichier un bit à la fois. Définissez un type associé que nous appellerons BFILE dans un fichier `bblib.h` (et incluez ce fichier dans `bblib.c`.

QUESTION 27 ► Réalisez l'implémentation correspondant à la spécification suivante :

```
/*
   bstart
   description : démarre un accès bit à bit au fichier dont le descripteur est
                 passé en paramètre (le fichier doit avoir été préalablement
                 ouvert).
                 Pour conserver la cohérence des données, aucun accès non bit à
                 bit au même descripteur ne doit être fait jusqu'au prochain
                 bstop.
   paramètres : descripteur du fichier, le mode dans lequel le descripteur est
                ouvert.
   valeur de retour : pointeur vers une structure BFILE allouée par bstart
                      ou NULL si une erreur est survenue
   effets de bord : aucun (outre l'allocation)
*/
BFILE *bstart(FILE *fichier);

/*
   bstop
   description : termine un accès bit à bit à un fichier préalablement démarré
                 par bstart (termine les E/S en attente et libère la structure
                 BFILE).
   paramètres : pointeur vers une structure BFILE renvoyée par bstart
   valeur de retour : 0 si aucune erreur n'est survenue
   effets de bord : écrit potentiellement dans le fichier
*/
int bstop(BFILE *fichier);

/*
   bitread
   description : lit un bit dans un fichier sur lequel un accès bit à bit
                 a été préalablement démarré a l'aide de bstart.
   paramètres : pointeur vers une structure BFILE renvoyée par bstart
   valeur de retour : bit lu ou -1 si une erreur s'est produite ou si la
```

```
                      fin de fichier a été atteinte
   effets de bord : la valeur de retour dépend du contenu du fichier
                    lit potentiellement dans le fichier
*/
char bitread(BFILE *fichier);

/*
   bitwrite
   description : écrit un bit dans un fichier sur lequel un accès bit à bit
                 a été préalablement démarré à l'aide de bstart.
   paramètres : pointeur vers une structure BFILE renvoyée par bstart, bit à
                écrire
   valeur de retour : -1 si une erreur s'est produite
   effets de bord : écrit potentiellement dans le fichier
*/
int bitwrite(BFILE *fichier, char bit);

/*
   beof
   description : retourne 1 si un accès en lecture préalable a échoué pour
                 cause d'atteinte de la fin d'une séquence de bits (fin de
                 fichier ou fin de séquence codée dans le fichier), 0 sinon.
                 L'accès bit à bit doit avoir été préalablement démarré à
                 l'aide de bstart.
   paramètres : pointeur vers une structure BFILE renvoyée par bstart
   valeur de retour : 1 ou 0
   effets de bord : aucun.
*/
int beof(BFILE *fichier);
```

Chapitre 5

Exercices facultatifs

1 Système électoral de Condorcet

Implémenter un système de vote de condorcet, ou autre. Votes dans un fichier.

2 extern, volatile, static

3 #pragma

4 Usage avancé de GDB

- script gdb : breakpoint conditionnel, fonctions, variables...

5 Alignement en mémoire

- regarder les adresses des champs d'une structure - regarder les adresses retournées par malloc pour des petites tailles

6 Ramasse-miette

— quelques classes bien définies

— racine d'atteignabilité ?

— gc copiant