

C PROGRAMMING

*Objectif de ce document ; guide de survie en C pour un débutant qui n'a jamais trop touché.
Idéalement quelques idées de programmation.*

Objectif de l'auteur du doc (Benjamin Suger) : expliqué à ma manière les concepts de base pour mieux les maîtriser, me pousser à faire des recherches plus poussées sur certains sujets, etc....

*Source ; Openclassroom, Cours du Dr Chuck, HackerNews, Google,
<https://www.cc4e.com/index.php>*

Version 1 du document. Potentiellement des mises à jours un jour si j'ai le temps d'ajouter du contenu. Ajout à faire ;

-plus d'exemple sur les variables constantes => utilisation pratique

-utilisation de GDB pour debugger

-ternaire explication + exemples

-GCC des .o => puis compilation de projet (explication objet)

-GCC avec des options en plus comme -Wextra -fmax-errors etc....

-Makefile avancé

-Linked list

*-liste de tips ; struct -> x, struct -> y / arithmetics pointers / pointers array ** / passer des arguments dans un script C depuis l'invite de commande (si vous voulez tout de suite le faire en urgence <https://c.developpez.com/faq/?page=Communication-avec-l-environnement#Comment-accéder-aux-paramètres-de-la-ligne-de-commandes>)*

-plus d'exemples de malloc

C est un langage de bas niveau. C'est-à-dire qu'il s'éloigne de notre langage qu'on utilise pour communiquer mais il est plus proche de celui de la machine.

D'autres langages de programmation se base sur C. Par exemple Python, qui est un langage de programmation plus haut niveau est lui-même « coder » en C.

Début d'un programme

Dans les lignes du début, il y a `#include <bibliothèque>` c'est pour inclure une librairie/bibliothèque. De base on parle de bibliothèque mais un anglicisme fait que parfois on parle de librairie.

Les bibliothèques contiennent des fonctions qu'on peut utiliser.

On peut en avoir plusieurs d'affilés dans un même programme;

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

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

Si on découpe `stdio.h` ;

-std pour standard

-io pour input output

-.h pour header

C'est donc la bibliothèque qui gère les input et output. On peut voir toutes les fonctions utilisables ici ; <https://www.ibm.com/docs/fr/i/7.5?topic=files-stdioh>

stdlib.h c'est pour « *standard library* »

Fonction main()

Un programme commence « toujours » par la fonction *main*. Par là je veux dire qu'il va exécuter ce qu'il trouve dans main, mais il peut y avoir du code avant et après.

Main qui veut dire que c'est la fonction principale.

(le toujours est entre guillemet, voir plus tard plus de détails sur les fonctions).

On la déclare ainsi ;

```
int main() {  
  
<instructions> ;  
  
return 0 ;  
  
}
```

main() est notre fonction principale. Entre parenthèse il n'y rien dans cette exemple. Ici c'est vide car on a pas de paramètre dans notre fonction, cela peut arriver (voir plus tard).

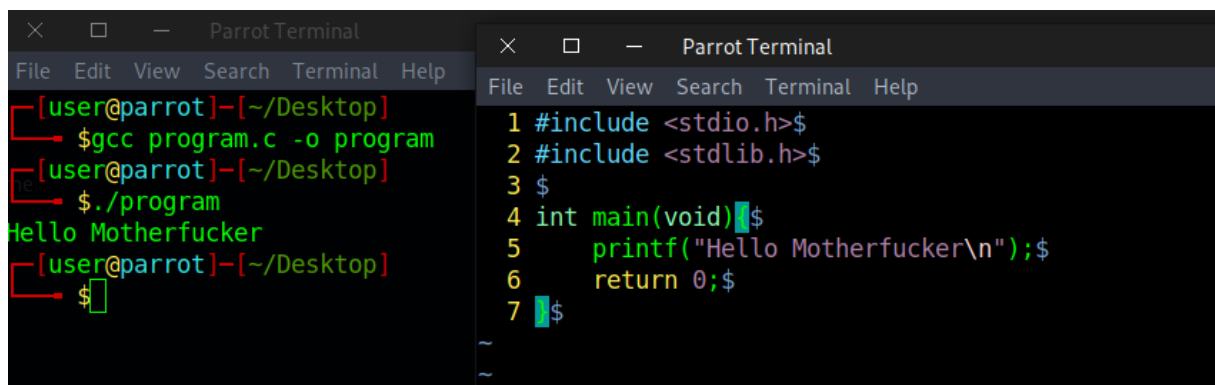
Entre les {} on va mettre les instructions. C'est-à-dire ce que doit faire le programme, chaque instruction se termine par un point-virgule ;

Ici à la fin on fait **return 0** ; car notre fonction est un **int**. C'est-à-dire que la fonction attend à la fin un *integer* (=nombre entier negatif ou positif +0).

On ne commence pas forcément par **int**, il y a différent types possible.

Exemple d'une fonction de base (décrite) ;

Ici on est sur une VM (virtual machine) avec Parrot OS. J'utilise VIM pour taper mon code.



```
File Edit View Search Terminal Help  
[user@parrot]~[/Desktop]  
$ gcc program.c -o program  
[user@parrot]~[/Desktop]  
$ ./program  
Hello Motherfucker  
[user@parrot]~[/Desktop]  
$  
~  
~
```

```
File Edit View Search Terminal Help  
1 #include <stdio.h>$  
2 #include <stdlib.h>$  
3 $  
4 int main(void)$  
5     printf("Hello Motherfucker\n");$  
6     return 0;$  
7 $  
~  
~
```

A droite ; le programme pour faire écrire « Hello Motherfucker » sur l'écran.

A gauche ; compilation + exécution du programme.

Description sur la gauche ;

Une fois que l'on a écrit notre code, dans `program.c`, il faut le compiler. *Qu'est-ce que compiler ?* C'est le moment où on traduit notre code en langage machine. L'ordinateur ne comprenant pas le code C qu'on a fait. L'ordinateur ne comprends que le binaire in fine.

Ici je compile avec `gcc`, comment marche la commande ?

`gcc <votre_code_en_point_c> -o <le_nom_que_vous_souhaitez_mettre_en_place>`

l'option `-o` n'est pas obligatoire mais ça vous permet de choisir le nom.

Pour lancer le programme on écrit `./<nom_du_programme>` car je suis déjà là où il faut, c'est-à-dire que je suis dans le directory où tout est présent.

```
$ls
program program.c README.license
[user@parrot]~[~/Desktop]
$
```

`ls` dans le shell veut dire *listing*, on voit que j'ai mon `program.c` puis `program` que j'ai compiler.

Lancement depuis un autre directory (à comprendre que c'est l'endroit dans lequel je me trouve avec mon terminal) ;

```
$pwd
/home/user/Documents
[user@parrot]~[~/Documents]
$./../Desktop/program
Hello Motherfucken
[user@parrot]~[~/Documents]
$
```

`Pwd` vous donne votre chemin d'accès, `pwd` = print working directory.

La ligne `./../Desktop/program` me permet de lancer le programme qui est dans `/Desktop/`

Attention potentiellement sur linux => il est possible que vous ne pouvez pas utiliser votre programme avec `./<nom_de_votre_programme>` car vous n'avez pas les permissions.

Vous devez taper `chmod +x <nom_de_votre_programme>`.

A droite ; on a notre code.

Les `#include` nous permet de faire appel aux librairies. Ici particulièrement `#include <stdio.h>` va nous permettre d'utiliser `printf()` qui vient de cette librairie.

`Printf()` est donc une instruction qu'on termine bien par un point-virgule ;

Sur l'utilisation de `printf()`, le string (= chaîne de caractère) doit être entre guillemet. On utilise `\n` pour le saut de ligne pour éviter quelque chose de moche comme ça ;

```
└─ $ ./program
Hello Motherfucker
└─ [user@parrot] - [~/Desktop]
└─ $ ./program
Hello Motherfucker └─ [user@parrot] - [~/Desktop] SANS LE \n
└─ $
```

On finit avec return 0 ; qui n'est pas obligatoire pour que cela marche, cependant d'après différentes sources cela est recommandé en bonne pratique.

Les variables

Pour déclarer une variable, il faut faire attention au nom. On ne peut pas commencer par un nombre et on ne peut pas mettre d'espace.

Attention à ce que cela soit cohérent, un petit exemple avec la nomenclature (à jours ? attention à vous renseigner) ici dans l'école 42 ; <https://github.com/Nqsir/The-Norm-42>

- Un nom d'union doit commencer par u_.
- Un nom d'enum doit commencer par e_.
- Un nom de globale doit commencer par g_.
- Les noms de variables, de fonctions doivent être composés exclusivement de minuscules, de chiffres et de '_' (Unix Case).
- Les noms de fichiers et de répertoires doivent être composés exclusivement de minuscules, de chiffres et de '_' (Unix Case).
- Le fichier doit être compilable.
- Les caractères ne faisant pas partie de la table ascii standard ne sont pas autorisés.

Partie conseillée

- Les objets (variables, fonctions, macros, types, fichiers ou répertoires) doivent avoir les noms les plus explicites ou mnémoniques. Seul les 'compteurs' peuvent être nommés à votre guise.

Les variables peuvent être de différent types ;

-int ; integer (voir plus haut) / -32 768 à 32 767

-long ; integer mais avec plus de capacité / -2 147 483 648 à 2 147 483 647

-float ; nombre décimaux / 1.17549×10^{-38} à $3.40282 \times 10^{+38}$

-double ; nombre décimaux / 2.22507×10^{-308} à $1.79769 \times 10^{+308}$

-signed char ; char c'est pour le texte / -128 à 127

-unsigned char ; 0 à 255

-unsigned long ; 0 à 4 294 967 295

-unsigned int ; 0 à 65 535

Il faut déclarer la variable pour l'utiliser ;

`<types_de_variable> <nom_de_la_variable> ;`

Exemple ;

```
1 int main(){
2     int chat;
3     float personne;
```

Pour affecter une valeur à une variable

`<nom_de_la_variable> = <valeur_de_la_valeur>;`

Exemple ;

```
chat = 2;$
personne = 7.3;$
```

N'oubliez pas que « l'ordinateur lit ligne par ligne » (c'est un peu imagé mais c'est bien fait ligne par ligne). Dans notre exemple je pourrais changer la valeur chat en chat = 3 plus bas il va donc prendre cette valeur en note.

On peut faire la déclaration et l'affectation sur la même ligne.

Exemple cas 1 = cas 2 :

Cas 1 ;

```
int chat;$
float personne;$
chat = 2;$
personne = 7.3;$
```

Cas 2 ;

```
3 int main(){
4     int chat = 2;
5     float personne = 7.3;
```

On arrive au même résultat mais nous fait gagner quelques lignes.

Il existe une possibilité de faire en sorte qu'une variable ne change jamais, c'est une **constante**.

Pour déclarer une constante ;

`const <type_de_variable> <nom_de_la_variable> = <valeur_de_la_variable>;`

Ici il faut tout de suite assigné une valeur.

Printf() d'une variable ;

Si on veut print une valeur d'une variable dans un texte, comme printf() voici la structure ;

`printf(« ceci est un texte de %d mots», <nom_de_la_variable_>);`

%d c'est pour un int, voici un tableau avec plus de détails ;

%d	Int
%u	Unsigned int
%ld	Long
%f	Float
%lf	Double
%s	String
%c	Single character
%p	Pointer

Comment faire si on en a plusieurs ?

```
printf("il y a %d chats, pour une moyenne de %f milliers d'habitant\n",chat,personne);$  
return 0;$
```

On fait la même chose mais on répète les variables après les virgules.

Récupérer une valeur de l'utilisateur ;

1. On doit déclarer la variable
2. On utilise scanf() qui vient aussi de stdio.h. Pour utiliser scanf()
scanf(« %d », &<nom_de_la_variable>);
On doit bien mettre le %d entre guillemet et mettre un & devant le nom de la variable

Exemple ;

```
3 int main(){$  
4     int nombre_de_chat;$  
5     printf("Combien vous avez de chat?\n");$  
6     scanf("%d",&nombre_de_chat);$  
7     printf("vous avez %d chats, ce n'est pas assez\n",nombre_de_chat);$  
8     return 0;$  
9 }$
```

```
[user@parrot] ~/Desktop  
$ ./program  
Combien vous avez de chat?  
7  
vous avez 7 chats, ce n'est pas assez  
[user@parrot] ~/Desktop
```

Les variables globales

Il faut savoir qu'une variable qui est dans une fonction, n'existe qu'au moment où la fonction est utilisée. Ensuite elle n'existe plus.

Si vous avez une fonction en dehors de main() (*allez voir la suite pour mieux comprendre ça*), vous ne pourrez pas l'utiliser une variable de cette fonction dans main() car elle n'existe plus.

Mais si vous voulez absolument avoir une variable en dehors de tout ça ; il suffit de la déclarer en dehors de la fonction et/ou de main().

```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $user@parrot|~|~/Desktop|
3 int a = 2;$am.c -o program
4 $user@parrot|~|~/Desktop|
5 int main(){
6     int b = 3;$
7     int resultat = a * b;$
8     printf("a est %d\n",a);$
9     printf("le résultat de a fois b est %d\n",resultat);$
10    return 0;$
11 }$
~|user@parrot|~|~/Desktop|
program.c
~|user@parrot|~|~/Desktop|
$ ./program
a est 2
le résultat de a fois b est 6
```

J'ai bien déclaré la variable a en dehors de main(), je peux la récupérer maintenant dans main() mais aussi dans d'autres fonctions du projet (voir la partie sur les projets).

Si je ne veux avoir que cette variable dans ce fichier lors de la déclaration je fais ;
static <type> <nom_de_la_variable> = <valeur> ;

Opérations Mathématiques

Les opérateurs mathématiques

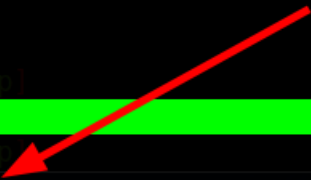
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Attention avec les division ;

Disclaimer=> si vous avez suivi le cours d'Openclasroom il y a une différence qui n'est pas précisé.

Si vous faites une division, il ne fait qu'une division euclidienne, c'est-à-dire sans chiffre après la virgule.

```
1 #include <stdio.h>$ [top]
2 $ gcc program.c -o program
3 int main(){ $ [~/Desktop]
4 - $ double a = 5/2;$
5 - $ double b = 6/4;$
6 - $ double c = 7/3;$ [top]
7 - $ printf("resultat de a %f\n",a);$
8 - $ printf("résultat de b %f\n",b);$
9 - $ printf("resultat de c %f\n",c);$
10 - $ return 0;$
11 }$ [user@parrot]~[~/Desktop]
$ gcc program.c -o program
[user@parrot]~[~/Desktop]
$ ./program
resultat de a 2.000000
résultat de b 1.000000
resultat de c 2.000000
[user@parrot]~[~/Desktop]
$
```



Quoi qu'il arrive on aura ça (*même si on change `int main()` pour `double main()` ou `float...`*) alors que j'ai bien mis du double en déclarant mes variables.

Cependant quelque chose qui n'est pas précisé dans le cours d'Openclassroom (en date de juillet 2024 et dernière mise à jours du cours fut le 14 février 2024), c'est que cela est différent si

le calcul est fait d'une autre manière.

```
1 #include <stdio.h>$
2 $ ./program
3 double main(){
4     double a =5;$
5     double b =2;$
6     double c =a/b;$
7     printf("resultat de c %f\n",c);$
8     return 0;$
9 }$
resultat de b 1.000000
resultat de c 2.000000
[user@parrot]~[~/Desktop]
$ gcc program.c -o program
[user@parrot]~[~/Desktop]
$ ./program
resultat de a 2.000000
resultat de b 1.000000
program.c 6,17
"program.c" 9L, 140C written
$ gcc program.c -o program
[user@parrot]~[~/Desktop]
$ ./program
resultat de c 2.500000
[user@parrot]~[~/Desktop]
$
```

Si on passe le calcul dans les variables ça change tout et on finit bien avec un double et pas un résultat euclidien.

Autre cas, si vous passez deux int pour une variable double/float

```
1 #include <stdio.h>$
2 $
3 int main(){
4     int a =5;$
5     int b =2;$
6     double c =a/b;$
7     printf("resultat de c %f\n",c);$
8     return 0;$
9 }$
```

Vous obtiendrez quand même 2.000000

Il faut passer des chiffres à virgules

Qu'est-ce que modulo ?

C'est le reste d'une division euclidienne.

5/2 => 2 en division euclidienne => il y a 2 fois 2 dans 5 => 2*2 =4 pour atteindre 5 il faut ajouter 1
5 = 2*2 +1
Donc 5%2 = 1

15/2 => 7 en division euclidienne => il y a 2 fois 7 dans 15 => 7*2 =14 pour atteindre 15 il faut ajouter 1
15 = 2*7 +1
Donc 15%2 = 1

Les raccourcis mathématiques

Très utile pour les boucles (voir plus bas), les raccourcis mathématiques permettent de faire des opérations d'incrémentatation ou de décrémentatation.

<variable>++; permet d'ajouter 1 à la valeur de la variable => qu'il faut déclarer + assigner une valeur en amont

<variable>--; permet de diminuer de 1 à la valeur de la variable => qu'il faut déclarer + assigner une valeur en amont

Pour les opérations ou vous voulez modifier votre variable de + - * / % x valeurs ;

<variable> <opérateur>= <valeur> ;

variable1 += 6 ; ajoute 6 à la variable1

variable1 -= 3 ; enlève 3 à la variable1

variable1 %=5 ; fait modulo 5 à la variable1

Pour faire plus de math, vous allez devoir puiser dans les librairies spécifiques.

Les conditions

Pour les conditions, il faut d'abord faire un test, voici les opérateurs ;

==	Egal
>	Supérieur
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
!=	Différent
&&	ET / AND
	OU / OR
!	NON / NO / NO (en espagnol)

Structure IF

```
if (<test>)  
{  
    <le_code_fait_des_choses>  
}  
else  
{  
    <le_code_fait_des_choses>  
}
```

La partie else n'est pas obligatoire c'est pour gérer les autres cas qui ne valident pas le premier test.

Exemple de code ;

```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 int main(){$
4     int chats;$
5     printf("combien vous avez de chat?\n",chats);$
6     scanf("%d",&chats);$
7     if (chats == 0){$
8         printf("Pourquoi ?\n");}$
9     else{$
10        printf("pas mal, bon début\n");$
11    }$
12    return 0;$
13 $
combien vous avez de chat?0
Pourquoi ?-[user@parrot]--[~/Desktop]
program.c
-[user@parrot]--[~/Desktop]
$ ./program
combien vous avez de chat?
0
Pourquoi ?
-[user@parrot]--[~/Desktop]
$ ./program
combien vous avez de chat?
4
pas mal, bon début
-[user@parrot]--[~/Desktop]
$
```

Oui les indentations sont pas terrible, fuck it

On peut faire aussi avec **else if** faire un second test. Voici la structure ;

```
If (<test>){
    <code>
}
else if(<second_test>){
    <code>
}
else{
    <code>
}
```

Exemple; avec un else if un peu spécial dans le test

```
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 int main(){
4     int chats;$
5     printf("combien vous avez de chat?\n",chats);$
6     scanf("%d",&chats);$
7     if (chats == 0){
8         printf("Pourquoi ?\n");}
9     else if(chats >= 1 && chats < 4){
10        printf("pas mal, bon début\n");}
11    else{
12        printf("vous avez une bonne escouade");}
13    return 0;$
14 }$
program.c
[user@parrot]~[~/Desktop]
$ ./program
combien vous avez de chat?
0
Pourquoi ?
[user@parrot]~[~/Desktop]
$ ./program
combien vous avez de chat?
1
pas mal, bon début
[user@parrot]~[~/Desktop]
$ ./program
combien vous avez de chat?
4
vous avez une bonne escouade
[user@parrot]~[~/Desktop]
$
```

Oui j'ai pas mis \n à la fin du dernier printf(), fuck it

[Cas particulier du !](#)

! veut dire « n'est pas ». Il est différent du != qu'on a dans les tests.

On le met avant un test *if* (!(<test>))

[Booléens](#)

1 = vrai / true

0 = faux / false

On peut attribuer un test à une variable => ce qui a pour conséquence de faire une valeur 1 ou 0 à la variable.

Il existe plein de manière de vérifier true or false. Une façon intéressante de faire est d'avoir une variable à qui on assigne un test d'une autre variable

```
3 int main(){  
4     int a = 2;  
5     int chats = a > 1;  
6     if (chats){  
7         printf("vrai");  
8     }  
9     else{  
10        printf("faux");  
11    }  
12    return 0;  
13 }
```

Ici le resultat va être vrai car $a > 1$, c'est peu intéressant dans cette exemple mais si on fait a avec une entrée d'utilisateur ça peut être sympa.

[SWITCH](#)

Quand vous avez plein de cas, vous n'avez pas forcément à faire une structure if qui pourrait être long. Imaginez 15 cas avec un if statement.....

Structure du switch ;

```
switch(<variable>)  
{  
    case <valeur_que_la_variable_peut_avoir> :  
        <instruction>;  
        break;  
    case <valeur_que_la_variable_peut_avoir> :  
        <instruction>;  
        break;  
    case <valeur_que_la_variable_peut_avoir> :  
        <instruction>;  
        break;  
    .....  
}
```

En gros l'idée c'est que la variable qu'on met dans switch() => chaque cas (case) prévoit une valeur de notre variable et du coup une instruction différente pour chaque cas.

Exemple venant de codewars ; site pour s'entraîner

```

switch (num) {
    case 1:
        return "Sunday";
    case 2:
        return "Monday";
    case 3:
        return "Tuesday";
    case 4:
        return "Wednesday";
    case 5:
        return "Thursday";
    case 6:
        return "Friday";
    case 7:
        return "Saturday";
    default:
        return "Wrong, please enter a number between 1 and 7";
}

```

default pour
les autres cas

[Le ternaire](#)

C'est une sorte de « one-liner » (comprendre une ligne de code) pour faire un statement if

Structure d'une expression ternaire ;

(<test>) ? <expression_si_vrai> : <expression_si_faux>;

Les boucles

Boucle = on répète une suite d'instructions plusieurs fois.

[While](#)

Un des types de boucle est « while » == « tant que »

Structure du while ;

```

while (<condition>)
{
    <instructions>;
}

```

ATTENTION AVEC LES BOUCLES. Il faut éviter les boucles infini qui pourrait casser, votre code, votre VM, votre ordinateur, votre vie.....

Dans les conditions de la boucle *while*, on a souvent quelque chose comme « tant que la variable n'est pas égal/supérieur/inférieur à cette valeur » **MAIS** si dans votre code à la fin de votre boucle *while*, vous ne changez pas la valeur de votre variable => **CATASTROPHE**

En gros si je devais écrire ce qui se passe ligne par ligne en utilisant notre langage ;

-valeur de ma variable = 0

-début de ma boucle while

-tant que ma variable n'est pas = 10 je continue à printf(« chats »). On appelle aussi la variable

un compteur

-fin de mon instruction, je retourne au début de ma boucle while

-tant que ma variable n'est pas = 10 je continue à printf(« chats ») **effectivement ma variable est toujours à 0 en dehors de ma boucle je n'ai pas fait de modification.**

-je printf(« chats »)

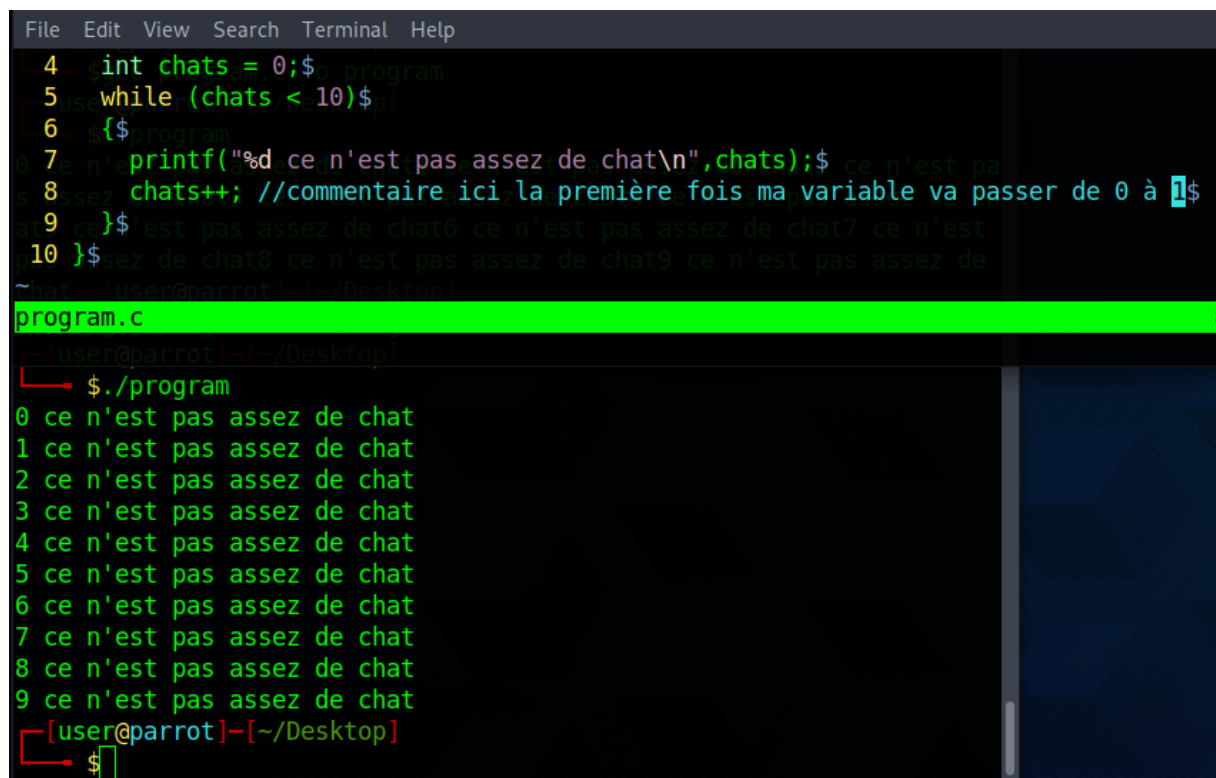
-fin de mon instruction, je retourne au début de ma boucle while

-tant que ma variable n'est pas = 10 je continue à printf(« chats ») **effectivement ma variable est toujours à 0 en dehors de ma boucle je n'ai pas fait de modification.**

-etc.... etc....

C'est ici qu'on utilise les raccourcis qu'on a appris pour incrémenter une valeur.

Exemple ;



```
File Edit View Search Terminal Help
4 int chats = 0;$
5 while (chats < 10){$
6 {
7 printf("%d ce n'est pas assez de chat\n",chats);$
8 chats++; //commentaire ici la première fois ma variable va passer de 0 à 1$
9 }$
10 }$
~chat-[user@parrot]-[~/Desktop]
program.c
~[user@parrot]-[~/Desktop]
$ ./program
0 ce n'est pas assez de chat
1 ce n'est pas assez de chat
2 ce n'est pas assez de chat
3 ce n'est pas assez de chat
4 ce n'est pas assez de chat
5 ce n'est pas assez de chat
6 ce n'est pas assez de chat
7 ce n'est pas assez de chat
8 ce n'est pas assez de chat
9 ce n'est pas assez de chat
~[user@parrot]-[~/Desktop]
$
```

Ici tout va bien, à noter ici quelques petits points ;

-ca commence forcément à 0

-avec mon test < 10 ca exclut donc 10 dans mon printf()

-mon code parle de chat

Second exemple ; mauvais code

```
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 int main(){$
4     int chats = 0;$ 'est pas assez de chat
5     while (chats != 10)$ pas assez de chat
6     {$ $ Home 7 ce n'est pas assez de chat
7         printf("%d ce n'est pas assez de chat\n",chats);$
8     }$ 9 ce n'est pas assez de chat
9 $ (user@parrot)~| ~/Desktop|
~ Trash $gcc program.c -o program
```

Ici pas bon ! A noter que j'ai fait un changement sur mon test qui est une autre façon de faire avec != mais qui arrive au même résultat excluant 10 dans mon printf().
Le plus gros problème ici, c'est qu'il ne va jamais finir la condition ! La variable ne va jamais atteindre 10.

[Do...while](#)

Dans une boucle *while* on fait un test puis on fait des instructions. Pour répéter tout ça car on a augmenté la valeur de notre variable dans le test.

Ici dans cette structure on va faire l'inverse. On va d'abord faire les instructions et ensuite le test avec *while*. **Attention**, on incrémente encore une fois bien notre variable (qu'on appelle aussi compteur) dans les instructions.

Voici la structure du do while

```
do
{
    <instructions>;
    <incrimente_mon_compteur>;
} while (<test>;
```

Cette structure permet d'exécuter au moins une fois les instructions. Ça peut être un objectif très précis et même rare mais au moins c'est arriver une fois. N'oubliez pas le programme va être « lu » par l'ordinateur ligne par ligne.

[For](#)

Structure de la boucle for ;

```
for (<variable> = <valeur>; <condition>; <incrementation_de_notre_variable>)
{
    <instructions>;
}
```

- Ici on assigne une valeur à notre variable dans le for statement
- Mais il faut bien déclarer notre variable avant de commencer notre for
- <condition> est identique que dans while, on utilise < == > != etc...
- ici au moins on ne peut pas se tromper pour une boucle infini car l'incrémentation doit être faite tout de suite.


```

1 #include <stdio.h>$
2 $
3 int main(){$
4     int chats = 0;$
5 $
6     for(chats=0;chats<10;chats++){
7         printf("il n'y a que %d chats\n",chats);$
8     }$
9 $
10 }$

```

Les fonctions

Plus de détails sur l'écriture de fonction. Pour le moment on a toujours fait nos opérations dans main() qui est la fonction principale du programme.

Mais dans main(), vous allez faire référence à d'autres fonctions que vous allez écrire en dehors de main().

Voici la structure des fonctions ;

```

<type> <function_name>(<parameters>)
{
    instructions ;
}

```

<type> c'est le type de données qu'on va avoir en sortie, en résultat, de notre fonction. Cela peut être du int, double, float....Mais aussi void si elle ne renvoie rien.

<function_name>(<parameters>) c'est le nom de la fonction, puis des paramètres si cela est nécessaire.

Dans les paramètres, on met encore une fois le <type> <nom_du_paramètre>

L'ordre

On a dit au tout début de ce document, que le programme commence « toujours » par main().

Le « toujours » est entre guillemet car c'est plus compliqué que ça. Le programme va effectivement utiliser main() en premier. Mais si dans main() vous faites référence à une fonction qui est en dehors, l'ordinateur qui lit ligne par ligne le code va vous dire ; « mais de quoi tu parles mec ? moi j'ai jamais entendu parler de ça ? t'es ouf ! ».

Exemple de ce problème, expliqué ligne par ligne ;

```

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int a = 2;$
6     int b = 3;$
7     int resultat = multiplication(a,b);$
8     printf("%d\n",resultat);$
9     return 0;$
10 }$
11 $
12 $
13 $
14 int multiplication(int a, int b){$
15     return a * b;$
16 }$

```

L'ordinateur fait ;

-je lis la ligne 1 : ok

-je lis la ligne 4 : ok

-ah ! ligne 4 c'est mon programme à faire, je me mets au travail : ok

-ligne 5 + ligne 6 j'ai 2 variables : ok

-ligne 7 je déclare une variable...ok....ok....elle est égale à.....multiplication(a,b).....Mais c'est quoi ca ? **JE N'AI RIEN LU A CE SUJET AAAHHHH DU COUP JE TRAVAIL PLUS VOILA erreur de compilation**

```

[user@parrot]~/Desktop
$ gcc program.c -o program
program.c: In function 'main':
program.c:7:20: warning: implicit declaration of function 'multiplication' [-Wimplicit-function-declaration]
7 |     int resultat = multiplication(a,b);
  |                    ^

```

Pour pallier à ce problème, on va faire un *prototype*. Après nos #include des librairies, on va mettre la première ligne de notre fonction,

```

1 #include <stdio.h>$
2 $
3 int multiplication(int a, int b);$
4 $
5 int main(){$
6     int a = 2;$
7     int b = 3;$
8     int resultat = multiplication(a,b);$
9     printf("%d\n",resultat);$
10    return 0;$
11 }$
12 $
13 $
14 $
15 int multiplication(int a, int b){$
16     return a * b;$
17 }$

```

La c'est bon

Si on recommence par lire, ligne par ligne ;

-je lis la ligne 1 : ok

-je lis la ligne 3, oh ! c'est une fonction pour plus tard d'accord.

-je lis la ligne 5, c'est mon programme cool

-ligne 6 et 7, des variables : ok

-ligne 8, une variable qui est le résultat de la fonction multiplication(a,b).....ah ! je connais cette fonction ! C'est bon je vais la retrouver !

A noter on a mis un ; sur la ligne 3 c'est ce qui fait comprendre à la machine que c'est pas la déclaration de la fonction mais juste un prototype.

Les projets

Un logiciel n'est pas simplement un fichier qu'on a compilé et qu'on exécute, c'est plus souvent des projets avec plusieurs fichiers.

Dans votre projet, [vous allez avoir des fichiers sources en .c et des headers en .h](#)

Les fichiers en .h c'est les prototypes des fonctions alors que les fichiers en .c c'est le code des fonctions.

Dans votre fichier en .c, vous allez le prévenir qu'il y a les prototypes, mais pas de la manière dont on fait ! Dans la précédente partie on a fait ça car c'est un petit programme mais imaginez bien des programmes avec des milliers de lignes !

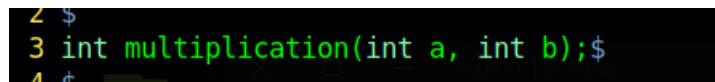
On va le mettre à la suite des #include



```
1 #include <stdio.h>;
2 #include "multiplication.h";
3
4
5 int main(){
```

Ici dans le #include on met cette fois entre guillemet « »

Dans votre fichier en .h c'est là où vous allez retrouver les prototypes comme on a fait au préalable ;



```
2
3 int multiplication(int a, int b);
4
```

[GCC](#)

On a vu gcc pour compiler notre code pour le rendre exécutable. Mais on peut le faire aussi sur des projets entiers avec différents fichiers.

Exemple ;

```

[user@parrot]--[~/Desktop/project]
$ pwd
/home/user/Desktop/project
[user@parrot]--[~/Desktop/project]
$ ls
main.c multiplication_division.c multiplication_division.h
[user@parrot]--[~/Desktop/project]
$

```

pwd => print working directory => c'est à dire où je suis dans ma machine linux

ls => listing => montre tout ce qu'il y a de visible dans ce fichier

On retrouve donc dans mon dossier « project », 3 fichiers (main.c, multiplication_division.c et multiplication_division.h).

Les fichiers en .c c'est là où il y a le code alors que le .h c'est le header.

Mon main.c ;

```

1 #include <stdio.h>$
2 #include "multiplication_division.h"$
3 $
4 int main(){$
5     int a, b;$
6     printf("le premier chiffre\n");$
7     scanf("%d",&a);$
8     printf("le second chiffre\n");$
9     scanf("%d",&b);$
10    multiplication_division(a,b);$
11    return 0;$
12 }$

```

On peut voir en ligne 2 que je fais appel au header multiplication_division.h entre guillemet.

Sur la ligne 10, j'ai une fonction multiplication_division() qui n'est pas dans main.c =>

l'ordinateur va la retrouver grâce au header. Donc ici je n'ai pas fait de prototype comme on a pu voir avant.

Dans le fichier multiplication_division.c ;

```

1 #include <stdio.h>$
2 #include "multiplication_division.h" //on inclut aussi ici notre .h$
3 $
4 $
5 int multiplication_division(int a, int b){$
6     int multiplication = a*b;$
7     int division = a/b;$
8     printf("le resultat de la multiplication est %d, le resultat de la division est %d\n",multiplication,division);$
9     return 0;$
10 }$

```

Simplement, il nous fait une multiplication et une division pour finir par un printf().

Enfin, multiplication_division.h ; (le premier include est pas obligatoire)

```
1 #include <stdio.h>$
2 int multiplication_division(int a, int b);$
```

J'ai mon prototype de ma fonction.

Maintenant avec gcc on va tout compiler pour mettre ca ensemble et l'exécuter.

```
[user@parrot]~/Desktop/project
$gcc main.c multiplication_division.c multiplication_division.h -o project
[user@parrot]~/Desktop/project
$ls
main.c multiplication_division.c multiplication_division.h project
[user@parrot]~/Desktop/project
$./project
le premier chiffre
2
le second chiffre
3
le resultat de la multiplication est 6, le resultat de la division est 0
[user@parrot]~/Desktop/project
$
```

Description ;

-ligne 1 = la commande qui me permet de compiler l'ensemble.

gcc <fichier1.c> <fichier2.c>.....<fichier_headers>.... -o <nom_que_je_veux>

rappel -o c'est pour output.

-ligne 2 = ls pour listing et voir si j'ai bien mon « project » qui est créer mais comme je n'ai pas fait d'erreur (sinon bloque à la compilation) tout est bon.

-ligne 3 = je lance mon programme tout fonctionne bien. **Ce qu'il faut comprendre** => je n'ai pas fait de prototype dans main.c => il vient directement de multiplication_division.h que j'ai include => je pourrais faire ca avec plus de fichiers (des dizaines, des centaines....) => bien sur ici c'est un peu léger c'est pour l'exemple mais imaginez sur des milliers de ligne de code.

Après le beau temps, vient la pluie.

Imaginons, que je dois ajouter quelque chose dans multiplication_division.c (une fonctionnalité, une phrase...). Si je modifie ce fichier, je vais devoir TOUT recompiler. Refaire la commande gcc avec tout dedans. Ce qui n'est pas très grave dans notre cas car c'est un petit projet mais imaginez encore une fois sur des milliers de ligne de code => cela peut être long.

Surtout que le reste je n'y touche pas, c'est juste imaginons une faute de frappe quelque part.

Après la pluie, le beau temps

Pour ça il y a une autre manière de faire avec make et makefile

[Make, mafile](#)

Pour que l'outil make fonctionne, il nous faut un « makefile »

[Makefile simple](#)

```

program: program.o weatherstats.o
    gcc -std=c11 -Wall -fmax-errors=10 -Wextra program.o weatherstats.o -o program

program.o: program.c weatherstats.h
    gcc -std=c11 -Wall -fmax-errors=10 -Wextra -c program.c -o program.o

weatherstats.o: weatherstats.c
    gcc -std=c11 -Wall -fmax-errors=10 -Wextra -c weatherstats.c -o weatherstats.o

clean:
    rm -f weatherstats.o program.o program

```

Structure d'une commande

<target> : **<component1>** **<component2>**
<gcc_command>

<target> c'est ce qu'on cherche à compiler. Vous pouvez voir dans certaines lignes des fichiers qui finissent par .o

Il s'agit d' **object files**, il s'agit d'une étape intermédiaire dans la compilation que l'on peut faire avec la commande gcc. Jusqu'à présent on n'en avez pas besoin car on fait des petits programmes.

<component1> **<component2>** on retrouve ici ce qui est nécessaire comme fichier pour créer notre target. Il peut y avoir plusieurs fichiers comme un seul.

<gcc command> ici on a des options en plus qui servent à avoir plus d'informations s'il y a un problème.

On affiche au maximum 10 erreurs avec fmax

Wall va nous donner des avertissements + des variables non utilisées

.....

Donc quand on va faire la commande **make program** pour la première fois il va faire plusieurs opérations ;

-compiler program.o

-compiler weatherstats.o

-compiler program grâce aux deux autres

Si on se rend compte qu'on a besoin de faire une modification dans program, on va refaire **make program**. Mais imaginez qu'il ne s'agit que d'une modification qui ne concerne qu'un printf() qui n'est pas en lien avec weatherstats.

Il ne va pas faire d'update de weatherstats ! Il sait que ce ne le concerne pas et il a déjà le weatherstats.o nécessaire.

C'est la force de make / makefile il va faire la compilation la ou il y a des choses à faire et pas des choses inutiles.

Nous sommes dans un cas simple et un peu bête mais globalement ca peut déjà faire gagner du temps.

make clean est une commande qui va supprimer nos fichiers (ici dans cette exemple c'est pas super malin de supprimer ca mais au moins on est tranquille).

Makefile normal

[SORTIE POUR VERSION 2]

Les pointeurs

La définition est la suivante ; « **un pointeur pointe vers l'adresse dans la mémoire de l'ordinateur d'une variable** ».

Revenons déjà sur les variables.

Quand on assigne une valeur à une variable ; **int chats = 10** ; l'ordinateur va garder en mémoire cette information. L'information à une adresse dans l'ordinateur ! Il ne va pas conserver l'information « chats » mais une adresse.

Quand je veux retrouver l'information, j'utilise le nom de la variable **chats**. Mais l'ordinateur lui en réalité. Il va chercher à l'adresse qu'il a enregistré l'information.

On peut déjà sans utiliser de pointeur, chercher l'adresse de cette variable. Pour ça il suffit de mettre un & devant le nom de la variable.

Il faut aussi faire une modification dans printf(), on va utiliser %p.

```
1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int chats = 10;$
6     printf("la valeur de la variable chats est de %d\n",chats);$
7     printf("l'adresse de la variable chats est la suivante %p\n",&chats);$
8 }$
9 $
10 $
```

```
➤ $ ./program
la valeur de la variable chats est de 10
l'adresse de la variable chats est la suivante 0x7ffd2e077dec
```

Maintenant, on peut regarder les pointeurs. Le pointeur va pointer la direction d'une adresse dans la mémoire.

Dans notre cas, on va pointer l'adresse de la variable chats.

Comment déclarer un pointeur ?

int *<nom_du_pointeur> ;

Ici c'est la création simple. Comme pour une variable on indique un type. La seule différence c'est le fait qu'on mette une * devant le nom de notre pointeur.

int *<nom_du_pointeur> = NULL ;

En ajoutant = NULL, on crée notre pointeur mais on lui indique de nous réserver une case vide de mémoire dans l'ordinateur.

int *<nom_du_pointeur> = &<nom_de_la_variable> ;

Enfin ici on retrouve ce qu'on vient de faire &<nom_de_la_variable> pour lui dire que c'est l'adresse de cette variable qui nous intéresse.

```

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int chats = 10;$
6     printf("la valeur de la variable chats est de %d\n",chats);$
7     printf("l'adresse de la variable chats est la suivante %p\n",&chats);$
8 $
9     int *pointeur = &chats;$
10    printf("l'adresse qui est enregistré dans notre pointeur est la suivante %p\n", pointeur);$
11 $
12 }$

```

Ligne 9, je lui demande d'enregistrer l'adresse de la variable chats.

En résultat ;

```

[user@parrot]--[~/Desktop]
└─ $ ./program
la valeur de la variable chats est de 10
l'adresse de la variable chats est la suivante 0x7ffde7eb0444
l'adresse qui est enregistré dans notre pointeur est la suivante 0x7ffde7eb0444
[user@parrot]--[~/Desktop]

```

On voit bien que l'adresse (qui est exprimé en hexadecimal) est identique. SUCESS !

Maintenant si je souhaite interroger mon pointeur, sur la valeur qui est dans cette adresse ?

Dans mon printf(), je dois mettre %d de nouveau car il s'agit d'un int et mettre
*<nom_du_pointeur>

```

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int chats = 10;$
6     printf("la valeur de la variable chats est de %d\n",chats);$
7     printf("l'adresse de la variable chats est la suivante %p\n",&chats);$
8 $
9     int *pointeur = &chats;$
10    printf("l'adresse qui est enregistré dans notre pointeur est la suivante %p\n", pointeur);$
11    printf("la valeur qui est enregistré dans pointeurs est la même que chats %d\n", *pointeur);$
12 }$
13 $ la suivante 0x7ffde7eb0444
14 $

```

```

[user@parrot]--[~/Desktop]
└─ $ ./program
la valeur de la variable chats est de 10
l'adresse de la variable chats est la suivante 0x7ffdb1116ba4
l'adresse qui est enregistré dans notre pointeur est la suivante 0x7ffdb1116ba4
la valeur qui est enregistré dans pointeurs est la même que chats 10
[user@parrot]--[~/Desktop]
└─ $

```

Il faut bien comprendre que les deux sont en lien !

Si je fais une modification sur la variable chats ;


```

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int chats = 10;$
6     printf("la valeur de la variable chats est de %d\n",chats);$
7     printf("l'adresse de la variable chats est la suivante %p\n",&chats);$
8     $ la suivante 0x7ffc773ff61d
9     int *pointeur = &chats;$
10    printf("l'adresse qui est enregistré dans notre pointeur est la suivante %p\n", pointeur);$
11    printf("la valeur qui est enregistré dans pointeur est la même que chats %d\n", *pointeur);$
12 $
13    //maintenant je vais une opération sur la variable chats$
14    chats +=2;$
15    printf("la valeur de la variable chats est de %d\n",chats);$
16    printf("la valeur qui est enregistré dans pointeur est la même que chats %d\n", *pointeur);$
17 }$ 0x700f354

```

```

$ ./program
la valeur de la variable chats est de 10
l'adresse de la variable chats est la suivante 0x7ffd9799f354
l'adresse qui est enregistré dans notre pointeur est la suivante 0x7ffd9799f354
la valeur qui est enregistré dans pointeur est la même que chats 10
la valeur de la variable chats est de 12
la valeur qui est enregistré dans pointeur est la même que chats 12

```

J'ai modifié la valeur de la variable chats. Du coup logiquement, comme pointeur n'est qu'une redirection. J'ai retrouvé la même chose

Je peux aussi manipulé la variable chats, à partir du pointeur. Ils sont liées ensemble donc si je modifie l'un ca modifie l'autre.

```

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int chats = 10;$
6     printf("la valeur de la variable chats est de %d\n",chats);$
7     printf("l'adresse de la variable chats est la suivante %p\n",&chats);$
8     $ la suivante 0x7ffde9274b4
9     int *pointeur = &chats;$
10    printf("l'adresse qui est enregistré dans notre pointeur est la suivante %p\n", pointeur);$
11    printf("la valeur qui est enregistré dans pointeur est la même que chats %d\n", *pointeur);$
12 $
13    //maintenant je vais une opération sur le pointeur$
14    *pointeur+=2; //attention ici je fais *pointeur$
15    printf("la valeur de la variable chats est de %d\n",chats);$
16    printf("la valeur qui est enregistré dans pointeur est la même que chats %d\n", *pointeur);$
17 }$ 72ade944

```

ATTENTION je fais *pointeur

```

$ ./program
la valeur de la variable chats est de 10
l'adresse de la variable chats est la suivante 0x7ffd72ade944
l'adresse qui est enregistré dans notre pointeur est la suivante 0x7ffd72ade944
la valeur qui est enregistré dans pointeur est la même que chats 10
la valeur de la variable chats est de 12
la valeur qui est enregistré dans pointeur est la même que chats 12

```

J'ai de nouveau la même valeur.

Si on revient en arrière dans le cours, on a parlé de scanf(), on avait déjà utilisé & sans savoir qu'il s'agit d'une adresse de la variable ;

```
Parrot Terminal
File Edit View Search Terminal Help

1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5 ++++$
6 int a = 0;$
7 int b = 0;$
8 printf("premier chiffre\n");$
9 scanf("%d",&a);$
10 printf("second chiffre\n");$
11 scanf("%d",&b);$
12 printf(" a = %d, et b = %d\n",a,b);$
13 }$
program.c
[user@parrot]~[~/Desktop]
$ gcc program.c -o program
[user@parrot]~[~/Desktop]
$ ./program
premier chiffre
8
second chiffre
4
a = 8, et b = 4
[user@parrot]~[~/Desktop]
$
```

Si on examine ce code ;

```
1 #include <stdio.h>$
2 $
3 int addition(int a, int b);$
4 $
5 int main(){$
6 ++++$
7     int a = 0;$
8     int b = 0;$
9     printf("premier chiffre\n");$
10    scanf("%d",&a);$
11    printf("second chiffre\n");$
12    scanf("%d",&b);$
13    printf(" a = %d, et b = %d\n",a,b);$
14    addition(a,b);$
15    int resultat;$
16    printf("le resultat de a + b est  %d\n",resultat);$
17 }$
18 $
19 int addition(int a, int b){$
20     int resultat = a+b;$
21     return 0;$
22 }
```

On peut voir qu'il ne va pas fonctionner comme je veux ;

```
➤ $ ./program
premier chiffre
8
second chiffre
2
a = 8, et b = 2
le resultat de a + b est 0
```

Pourquoi ? Parce que la variable résultat dans la fonction main() et dans la fonction addition() ne sont pas les mêmes. Celle de la fonction addition(), n'existe que pendant l'exécution de celle-ci. On peut le « prouver » avec un printf() dans la fonction addition().

```
1 #include <stdio.h>$
2 $
3 int addition(int a, int b);$
4 $
5 int main(){$
6 ++++$
7     int a = 0;$
8     int b = 0;$
9     printf("premier chiffre\n");$
10    scanf("%d",&a);$
11    printf("second chiffre\n");$
12    scanf("%d",&b);$
13    printf(" a = %d, et b = %d\n",a,b);$
14    addition(a,b);$
15    int resultat;$
16    printf("le resultat de a + b est  %d\n",resultat);$
17 }$
18 $
19 int addition(int a, int b){$
20     int resultat = a+b;$
21     printf("la variable resultat de la fonction addition est égale à %d\n",resultat);$
22     return 0;$
23 }$
```

```
[user@parrot]~[~/Desktop]
➤ $ ./program
premier chiffre
8
second chiffre
2
a = 8, et b = 2
la variable resultat de la fonction addition est égale à 10
le resultat de a + b est 0
```

Moi je souhaite trouver résultat dans main(). Je pourrais bien sur la créer dans main() mais mon objectif c'est de l'avoir avec un pointeur.

Pour faire ça, il faut changer la fonction.

Il faut lui indiquer qu'on va prendre un paramètre en plus.

```

1 #include <stdio.h>$
2 $
3 int addition(int a, int b, int *resultat);$
4 $
5 int main(){
6     int a = 0;
7     int b = 0;
8     printf("premier chiffre\n");
9     scanf("%d",&a);
10    printf("second chiffre\n");
11    scanf("%d",&b);
12    printf("a = %d, et b = %d\n",a,b);
13    int resultat; //changement de place pour le mettre avant la fonction
14    addition(a,b,&resultat); //ici j'ai mis & pour indiquer que je met l'adresse de la variable resultat dans main()
15    printf("le resultat de a + b est %d\n",resultat);
16 }
17 $
18 $
19 int addition(int a, int b, int *resultat){
20     *resultat = a+b; //ici j'ai mis de nouveau * pour le pointeur resultat
21     return 0;
22 }

```

J'ai mis quelques commentaires pour montrer ce que j'ai changé.

En gros l'idée est la suivante ; suivre nos premiers exemple de modification à partir du pointeur.
Ma fonction addition(), prend 3 paramètres ;

- a
- b
- *resultat

Qu'est-ce que je dis à l'ordinateur ? Voici ma conversation avec l'ordinateur ;

- Monsieur l'ordinateur, j'ai une fonction qui prend 3 paramètres
- ok
- les 2 premiers sont des int
- ok
- Le troisième est une **ADRESSE d'une variable**. Variable que j'ai déjà déclaré (int resultat), je vous le signale par le symbole &
- ok.....donc en gros ***pointeur = &resultat**. Ici le pointeur vous l'avez aussi appelé « resultat ».
- En soit c'est comme si vous faisiez cette ligne la pour déclarer un pointeur.
- c'est ca monsieur l'ordinateur !
- donc si vous faites « *resultat = a + b » comme c'est lié à la variable resultat dans main() ca va modifier aussi la valeur. Comme vous avez fait dans l'exemple plus haut avec une modification du pointeur (attention à bien mettre *).
- Exactement, je vois qu'on se comprend, merci beaucoup bonne journée !
- Bonne journée

On va voir un second cas, qui va nous permettre de faire quelque chose qui jusqu'à présent était impossible !

A partir d'une fonction, avoir 2 résultats ! Jusqu'à présent, on ne pouvait avoir qu'un seul résultat dans le return. Mais avec ce qu'on a découvert pour les pointeurs dans les fonctions, on va pouvoir faire le même types de liens que dans l'exemple précédent.

```

1 #include <stdio.h>$
2 $
3 int multiplication_division(int a, int b, int *pointeur1, int *pointeur2);$
4 $
5 int main(){$(user@parrot)~(~/Desktop)
6 ++++$ gcc program.c -o program
7 int a = 0;$parrot~(~/Desktop)
8 int b = 0;$program
9 int multiplication, division;$
10 printf("premier chiffre\n");$
11 scanf("%d",&a);$
12 printf("second chiffre\n");$
13 scanf("%d",&b);$
14 printf(" a = %d, et b = %d\n",a,b);$
15 multiplication_division(a,b,&multiplication,&division);$
16 printf("le resultat de la multiplication est %d\n",multiplication);$
17 printf("le resultat de la division est %d\n",division);$
18 }$
19 $ ./program
premier chiffre
20 8
21 int multiplication_division(int a, int b,int *pointeur1, int *pointeur2){$
22 *pointeur1 = a * b;$
23 *pointeur2 = a / b;$
24 return 0;$
25 }$
le resultat de la multiplication est 16

```

Bref explication (je vous laisse bien décortiqué), mais dans l'idée j'ai envoyé des adresses de variable dans une fonction. La fonction est bien déclarée avec des pointeurs (comme on a pu voir avec `int *<nom_du_pointeur>`), ce qui correspond à écrire dans l'idée (c'est pas tout à fait ça attention) `*pointeur1 = &multiplication`. Du moins c'est ce que comprend la machine

```

$ ./program
premier chiffre
8
second chiffre
2
a = 8, et b = 2
le resultat de la multiplication est 16
le resultat de la division est 4

```

Les tableaux / array

Les tableaux sont une suite de variable de même types. On appelle cela aussi un array.

Il faut comprendre que ce tableau est un pointeur lui aussi. Si vous cherchez son adresse, vous allez avoir l'adresse du premier élément qu'il index.

Comment déclarer un tableau ?

<types> <nom_tableau>[<nb_valeur>];

<types> comme quand on déclare une variable `int / double / long /`

<nb valeur> on doit lors de la déclaration de notre tableau, lui donner l'information en amont de sa longueur. Car l'ordinateur va réserver des places dans sa mémoire pour le tableau.

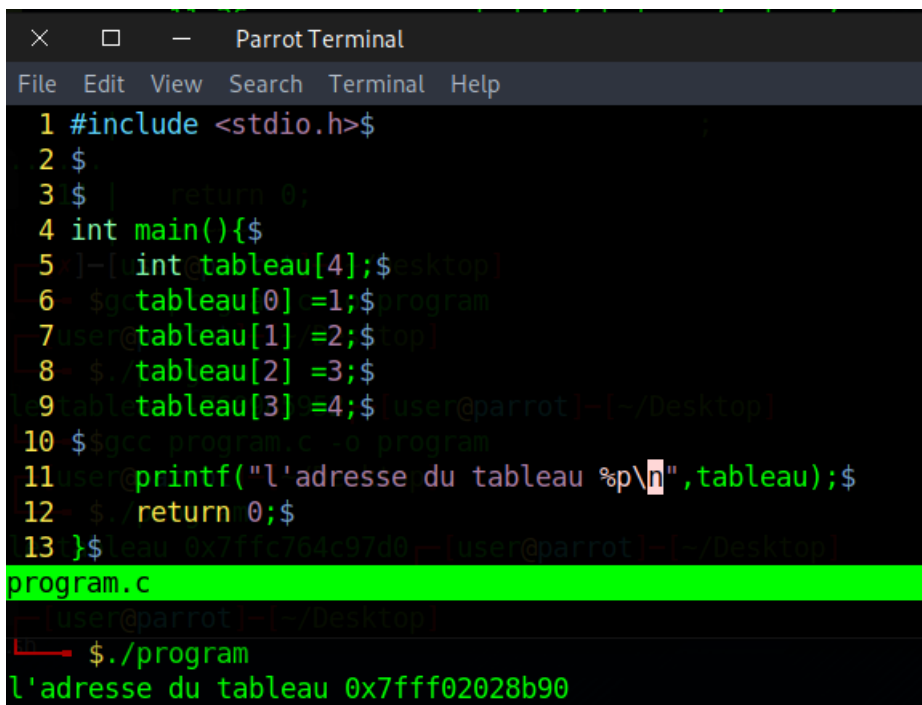
Ensuite pour assigner des valeurs dans notre tableau ;

<nom_tableau>[<espace>] = <valeur>;

<espace> c'est « le nombre de case » de mémoire qu'on veut remplir. Attention on commence comme d'habitude par 0.

```
1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int tableau[4];$
6     tableau[0] = 1;$
7     tableau[1] = 2;$
8     tableau[2] = 3;$
9     tableau[3] = 4;$
10    return 0;$
11 }$
12 $
```

On peut voir l'adresse du tableau, qui correspond en réalité à l'adresse de la première case. ;



```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 $
4 int main(){$
5     int tableau[4];$desktop]
6     tableau[0] = 1;$program
7     tableau[1] = 2;$top]
8     tableau[2] = 3;$
9     tableau[3] = 4;$[user@parrot]~[~/Desktop]
10 $gcc program.c -o program
11 user@parrot~$ printf("l'adresse du tableau %p\n",tableau);$
12 $./program
13 l'adresse du tableau 0x7ffc764c97d0,[user@parrot]~[~/Desktop]
program.c
[user@parrot]~[~/Desktop]
$./program
l'adresse du tableau 0x7fff02028b90
```

On peut aussi printf() les valeurs du tableau

```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 $
4 int main(){
5     int tableau[4];
6     tableau[0] =1;
7     tableau[1] =2;
8     tableau[2] =3;
9     tableau[3] =4;
10 }
11 printf("la valeur du premier élément du tableau %d\n",*tableau);
12 printf("la troisième valeur du tableau %d\n", tableau[2]);
13 return 0;
14 }
15 $ gcc program.c -o program
16 $ ./program
17 ~ adresse du tableau 0x7fff02028b90
18 ~ [user@parrot]~[~/Desktop]
program.c
"program.c" 18L, 284C written
$ ./program
la valeur du premier élément du tableau 1
la troisième valeur du tableau 3
```

On peut aussi initialiser un tableau d'une manière différente ;

int <tableau>[<espace>] = {0}

Dans ce cas toutes les valeurs sont à 0

```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>$
2 $
3 $
4 int main(){
5     int tableau[4] = {0};
6     printf("la troisième valeur du tableau %d\n", tableau[2]);
7     return 0;
8 }
9 $ gcc program.c -o program
10 $ ./program
11 ~ [user@parrot]~[~/Desktop]
program.c
~ [user@parrot]~[~/Desktop]
$ ./program
la troisième valeur du tableau 0
~ [user@parrot]~[~/Desktop]
$
```

int <tableau>[<espace>] = {1} Attention lorsqu'il ne s'agit pas de 0, vous allez obtenir 1 en première valeur et 0 aux autres ;

```

× □ — Parrot Terminal
File Edit View Search Terminal Help
2 $ quatrième valeur du tableau 0
3 $ er@parrot:~[~/Desktop]
4 int main(){ $
5 user@ int tableau[4] = {1}; $
6 $ ./program
7 trois printf("la première valeur du tableau %d\n", tableau[0]); $
8 trois printf("la seconde valeur du tableau %d\n", tableau[1]); $
9 trois printf("la troisième valeur du tableau %d\n", tableau[2]); $
10 trois printf("la quatrième valeur du tableau %d\n", tableau[3]); $
11 } $ quatrième valeur du tableau -1092566576
12 $ er@parrot:~[~/Desktop]
program.c
"program.c", 15L, 318C written
$ ./program
la première valeur du tableau 1
la seconde valeur du tableau 0
la troisième valeur du tableau 0
la quatrième valeur du tableau 0

```

On peut donc créer un tableau de cette manière

```
Int <tableau>[] = {<Valeur>,<Valeur>.....<valeur>}
```

Rappel ; je mets int car je veux un tableau avec des int, il est possible de mettre d'autres types en fonction de ce qu'on veut faire.

```

1 #include <stdio.h>$ tableau 0
2 $troisième valeur du tableau -1092566576
3 $er@parrot|~|~/Desktop|
4 int main(){$m.c -o program
5 user@int tableau[] = {101,45,8,666};$
6 $$. ./program
7 premierprintf("la première valeur du tableau %d\n", tableau[0]);$
8 secondprintf("la seconde valeur du tableau %d\n", tableau[1]);$
9 troisièprintf("la troisième valeur du tableau %d\n", tableau[2]);$
10 quatrièprintf("la quatrième valeur du tableau %d\n", tableau[3]);$
11 )}$er@parrot|~|~/Desktop|
program.c [+]
"program.c" 15L, 328C written
$→ $. ./program
la première valeur du tableau 101
la seconde valeur du tableau 45
la troisième valeur du tableau 8
la quatrième valeur du tableau 666

```

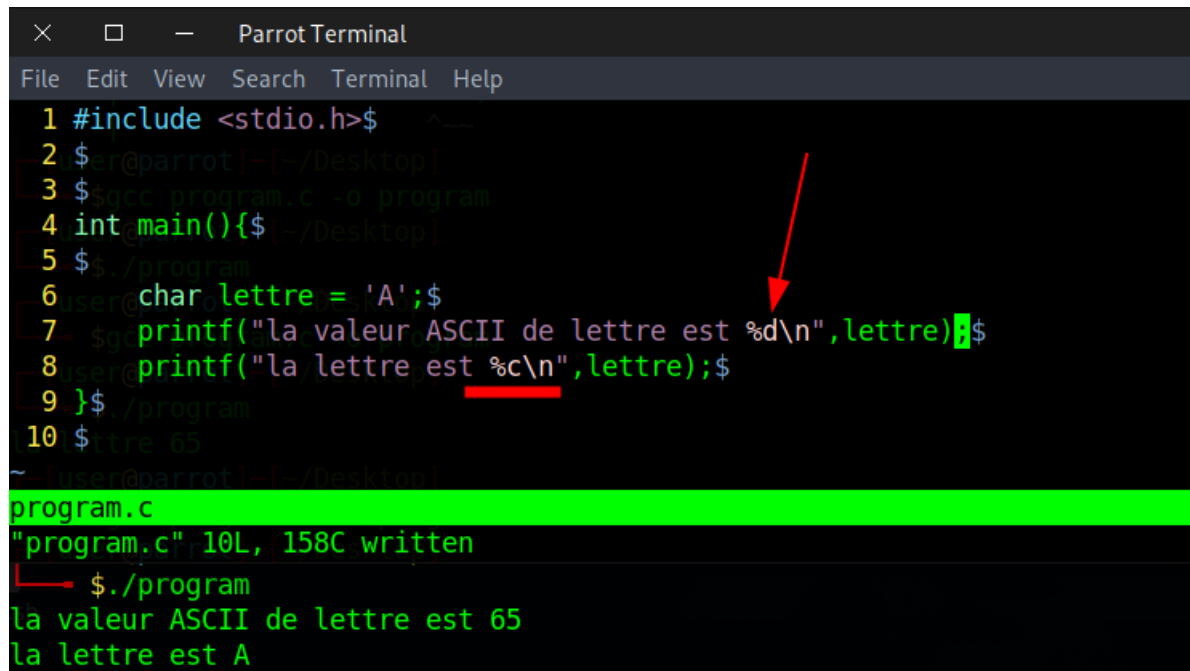

CHAR

Attention, il ne peut enregistrer qu'une lettre !

Attention, lors de la déclaration de la variable il faut utiliser " et pas « »

Si vous essayez de sortir avec le printf() %d => vous aurez la valeur ASCII de la lettre. N'oubliez pas l'ordinateur n'enregistre pas vraiment la lettre mais la valeur ASCII de celle-ci.

Pour avoir la lettre « lisible » / « convertit » => il faut utiliser %c



```
1 #include <stdio.h>
2
3 $gcc program.c -o program
4 int main(){
5     ./program
6     char lettre = 'A';
7     printf("la valeur ASCII de lettre est %d\n", lettre);
8     printf("la lettre est %c\n", lettre);
9 }
10 $./program
    lettre 65
~$ ./program
"program.c" 10L, 158C written
$ ./program
la valeur ASCII de lettre est 65
la lettre est A
```

<https://www.ascii-code.com/fr>

65	101	41	01000001	A	A	A majuscule
----	-----	----	----------	---	-------	-------------

Chaîne de caractères

Comment faire pour enregistrer un mot ou un texte, si on ne peut pas faire qu'une lettre ?

Grace au tableau ! Une chaîne de caractère = un tableau

C'est ici qu'on peut utiliser « » et non pas " pour enregistrer un mot/phrase/texte

```
Parrot Terminal
File Edit View Search Terminal Help
1 #include <stdio.h>
2 $-[user@parrot]~[~/Desktop]
3 $gcc program.c -o program
4 int main() {
5     char mot[] = "chats";
6     printf("le mot est %s\n", mot);
7 }
8 $gcc program.c -o program
9 $[user@parrot]~[~/Desktop]
10 $./program
11 le mot est chats
12
program.c
"program.c" 9L, 98C written
[user@parrot]~[~/Desktop]
$./program
le mot est chats
```

A noter que pour printf() notre chaîne de caractère => on utilise %s

ATTENTION il y a un piège avec les chaînes de caractères !!!

L'ordinateur enregistre une valeur en plus ! il enregistre \0 en dernier élément pour indiquer que la chaîne de caractère s'arrête ici.

Dans la méthode que je vous ai montré, vous n'avez pas à le mettre. Mais si vous enregistrez une chaîne de caractère lettre par lettre, il faut penser à la dernière \0

```
File Edit View Search Terminal Help
5 $[user@parrot]~[~/Desktop]
6 $./char_mot[5]; //ici je met 5
7 char mot[5];
8 mot[0] = 'c';
9 mot[1] = 'h';
10 mot[2] = 'a';
11 mot[3] = 't';
12 mot[4] = '\0'; //je signale que c'est la fin
13 printf("le mot est %s\n", mot);
14 printf("le dernier char est %c\n", mot[4]);
15 $gcc program.c -o program
16
program.c
$./program
le mot est chat
le dernier char est
```

En soit cela n'est pas obligatoire mais recommandé. Pourquoi ? Pas obligatoire car j'ai bien dit mot[6] donc il y a l'espace qui suffit. Par contre si je met mot[5] et pas de \0

```
Parrot Terminal
File Edit View Search Terminal Help
5 $ er@parrot:~[~/Desktop]
6 - $. char mot[4]; //ici je met 4$
7 mot mot[0] = 'c'; $
8 dern mot[1] = 'h'; $
9 user mot[2] = 'a'; $ ktop
10 - $g mot[3] = 't'; $ program
11 user //je ne signale pas que c'est la fin$
12 $ $. /program
13 mot printf("le mot est %s\n", mot); $
14 } $ nier char est
15 $ er@parrot:~[~/Desktop]
program.c
"program.c" 15L, 217C written
$ ./program
le mot est chatpe 
V
```

Pour plus d'opération sur des chaînes de caractères, il faut utiliser la bibliothèque `string.h`

`#import <string.h>`

Listes non exhaustive de fonction ;

strlen pour calculer la longueur d'une chaîne.

strcpy pour copier une chaîne dans une autre.

strcat pour concaténer 2 chaînes.

strcmp pour comparer 2 chaînes.

strchr pour rechercher un caractère.

strpbrk pour rechercher le premier caractère de la liste.

strstr pour rechercher une chaîne dans une autre.

sprintf pour écrire dans une chaîne.

Préprocesseur

Directive du préprocesseur = lignes qui commencent avec #

#define permet d'associer un mot à une valeur. Cela va s'appliquer à tout votre fichier, sans prendre de place en mémoire à l'inverse d'une constante.

Vous pouvez aussi faire des calcul dans **#define** à partir d'autres valeurs que vous avez aussi **#define**.

Dans le préprocesseur, il existe aussi des constantes déjà existantes ;

__LINE__ donne le numéro de la ligne actuelle.

__FILE__ donne le nom du fichier actuel.

__DATE__ donne la date de la compilation.

__TIME__ donne l'heure de la compilation.

Macros

Vous pouvez définir des macros => c'est-à-dire remplacer une fonction avec des paramètres par un autre

```
#define <fonction()> <fonction_qui_existe_avec_paramètre> ;
```

STRUCTURE & ENUMERATION

Une structure est un assemblage de variables qui peuvent avoir différents types. Pour l'utiliser on le déclare dans un fichier header .h (voir plus haut sur les projets).

Comment déclarer une structure ?

```
struct <nom_structure>{  
    int <variable>;  
    float <variable>;  
    char <variable>;  
};
```

A noter, vous pouvez mettre des tableaux/array dans les structures. On a aussi terminer notre structure par un ; à ne pas oublier (ce n'est pas en lien avec le fait que j'ai fait cette erreur...je vois pas de quoi vous parler)

Exemple ;

Fichier en .h

```
1 #include <stdio.h>$  
2 struct test_structure{$  
3     int hauteur;$  
4     int longueur;$  
5     char lettre;$  
6 };$
```

Notre program.c qui est le main()

```
1 #include <stdio.h>$  
2 #include "structure.h"$  
3 $  
4 int main(){  
5     struct test_structure variable_structure; //ici on crée une variable "variable_structure" qui n'est donc pas int/float/long/double....mais un struct$  
6     variable_structure.hauteur = 10;//ici on va mettre des valeurs dans notre structure, a noter la forme <structure>.<element>+;$  
7     variable_structure.longueur = 1;$  
8     variable_structure.lettre = 'A';$  
9 $  
10    printf("la hauteur est de %d, la longueur est de %d et la lettre est %c\n", variable_structure.hauteur, variable_structure.longueur, variable_structure.lettre);$  
11 $  
12 $  
13 $
```

Resultat

```
[x]-[user@parrot]-[~/Desktop]  
$ gcc program.c structure.h -o program  
[user@parrot]-[~/Desktop]  
$ ./program  
la hauteur est de 10, la longueur est de 1 et la lettre est A  
[user@parrot]-[~/Desktop]  
$
```

Pour passer des valeurs dans la structure, on peut aussi ainsi ;

```

1 #include <stdio.h>$
2 #include "structure.h"$
3 $
4 int main(){
5     struct test_structure variable_structure = {10,1,'B'}; //ici on passe les valeur des la créations
6     printf("la hauteur est de %d, la longueur est de %d et la lettre %c\n", variable_structure.hauteur, variable_structure.longueur, variable_structure.lettre);$
7     $
8     $
9 }$
10 $

```

Résultat ;

```

[user@parrot] [~/Desktop]
$ gcc program.c structure.h -o program
[user@parrot] [~/Desktop]
$ ./program
la hauteur est de 10, la longueur est de 1 et la lettre B
[user@parrot] [~/Desktop]
$

```

Structure and array :

On peut bien sur utiliser un array dans une structure.

```

[user@parrot] [~/Desktop/project]
$ gcc test.c -o test
[user@parrot] [~/Desktop/project]
$ ./test
3
2
[user@parrot] [~/Desktop/project]
$

```

```

1 #include <stdio.h>$
2 $
3 struct point{
4     int a;$
5     int b;$
6 };$
7 $
8 int main(){
9     struct point test[5]; //un array dans une structure$
10    test[0].a = 1;$
11    test[0].b = 2;$
12    test[1].a = 3;$
13    test[1].b = 4;$
14    printf("%d\n", test[1].a);$
15    printf("%d\n", test[0].b);$
16    return 0;$
17 }$

```

Typedef

Typedef permet d'éviter de faire une aussi longue declaration dans main().

Il faut l'indiquer dans votre fichier .header.

`typedef struct <nom_de_la_structure> <alias_de_la_structure> ;`

```

1 #include <stdio.h>$
2 typedef struct test_structure structuro;$
3 $
4 struct test_structure{
5     int hauteur;$
6     int longueur;$
7     char lettre;$
8 };$

```

Dans main()

```

1 #include <stdio.h>$
2 #include "structure.h"$
3 $
4 int main(){$
5     struct variable_structure = {10,1,'B'}; //ici c'est structuro maintenant, plus besoin de mettre struct$
6     printf("la hauteur est de %d, la longueur est de %d et la lettre %c\n", variable_structure.hauteur, variable_stru$
7 $
8 $

```

Enumerations

L'énumération ressemble à la structure dans l'organisation => c'est-à-dire à mettre dans un fichier .h et la définition (même l'utilisation de typedef).

Mais l'énumération est juste une liste de valeur possible.

Comment le déclarer ?

```

enum <nom_enumération> {
    <valeur1>, <valeur2>, <valeur3>....<valeur n>
}

```

MANIPULATION DE FICHIER

En plus de stdio.h, on doit utiliser stdlib.h

Avant d'ouvrir un fichier il faut déclarer un pointeur sur FILE. Pourquoi ? Parce que c'est comme ça qu'on le retrouve grâce à stdio.h.

```
FILE *fichier = NULL ;
```

Ensuite on va pouvoir utiliser fopen() qui est la fonction qui permet d'ouvrir notre fichier ;

```
fichier = fopen(« <nom_du_fichier> », « <mode> »);
```

Mode ici permet de déterminer ce qu'on va faire avec ce fichier ;

r pour read only => le fichier doit déjà exister

w pour écriture seulement => si le fichier n'existe pas il est créé

a pour ajout => ajoute à la fin du document les informations

a+ pour ajout en lecture / écriture à la fin => écriture + lecture à la fin

r+ pour lecture et écriture => le fichier doit déjà exister

w+ pour lecture et écriture + suppression du contenu au préalable

Nom_du_fichier c'est s'il est dans le même dossier/directory, sinon il faut mettre le chemin d'accès (attention il faut mettre 2 \ sinon le compilateur va penser à \n).

Ecrire dans un fichier

On peut le faire à travers différentes fonctions ;

fputc() pour mettre un seul caractère

fputs() pour mettre une chaîne de caractère

fprintf() pour mettre une chaîne formatée comme printf() qu'on connaît

Fermer un fichier

Il faut fermer le fichier une fois qu'on a fait nos modifications avec `fclose(<fichier>);`

```

1 #include <stdio.h>$
2 #include <stdlib.h>$
3 $
4 int main(){
5     FILE *fichier = NULL; // le pointeur$
6     fichier = fopen("test.txt","r+");//on ouvre le fichier test.txt$
7     printf("How many cats?\n");$
8     int cats;$
9     scanf("%d",&cats);$
10    fprintf(fichier,"le gars il a %d chats",cats);//on print directement dans le fichier$
11    fclose(fichier);$
12    return 0;$
13 }$

```

```

[user@parrot]--[~/Desktop]
└─$ cat test.txt
[user@parrot]--[~/Desktop]
└─$ gcc program.c -o program
[user@parrot]--[~/Desktop]
└─$ ./program
How many cats?
5
[user@parrot]--[~/Desktop]
└─$ cat test.txt
le gars il a 5 chats
[user@parrot]--[~/Desktop]
└─$

```

Allocation dynamique / Malloc()

Attention, il faut aller dans la section Astuces pour voir l'utilisation de sizeof() avant de continuer cette section.

Pourquoi faire de l'allocation dynamique de mémoire ?

Quand on crée une variable, l'ordinateur nous donne un endroit dans la mémoire pour enregistrer ensuite une valeur, un char.....

```

1 #include <stdio.h>$
2 //le variable int est de 4 octet
3 //le en octet d'une variable long est de 8 octet
4 int main()$
5 {
6     //le en octet d'une variable double est de 8 octet
7     //le en octet d'une variable char est de 1 octet
8     int variable; //ici l'ordinateur libère un espace pour cette variable$
9     //on peut d'ailleurs déjà aller chercher son adresse$
10    printf("%p\n",&variable);$
11    $rm program.exe
12    //on peut aussi savoir combien il prend d'espace$
13    printf("la variable prend %d bytes\n",sizeof(variable));$
14 }$
15 ./program
16 ~x7ffecf6947c --(user@parrot)~(~/Desktop)
program.c
~(user@parrot)~(~/Desktop)
$ ./program
0x7fff5314961c
la variable prend 4 bytes
~(user@parrot)~(~/Desktop)

```

Rappel ; bytes = octets, 1 (un) (uno) (eins) (yí) (๑) octet = 8 bits

Cette mémoire n'est pas illimité, même si on en a normalement beaucoup sur notre ordinateur.

Le cas le plus fréquent, c'est si on a un array/tableau dont on ne connaît pas la taille à l'avance (à partir de maintenant je dirais array).

Comment utiliser malloc ?

```

1 #include <stdio.h>$
2 #include <stdlib.h> //c'est ici qu'est malloc$
3 $
4 int main(){$
5     int nb; // nb de case$
6     printf("combien de case dans mon tableau ?\n");$
7     scanf("%d",&nb);$
8     $
9     int *tableau; // pointeur qui va me permettre de faire mon array plus tard, il va contenir des int$
10    tableau = malloc(nb*sizeof(int)); // je libère de l'espace pour mon tableau$
11    $
12    free(tableau); // on doit free l'allocation de mémoire done$
13    $
14    return 0;$
15 }$
16 $

```


ASTUCES

Convertir un int en float/double

On peut cast c'est-à-dire changer le type de variable d'un int vers double/float

```
1 #include <stdio.h>$
2 $
3 int main(){$
4     int a= 1;$
5     double b;$
6     printf("%d\n",a); // int$
7     b = (double) a; // cast => transformation de notre int en double$
8     printf("%lf\n",b);$
9 }
```

```
[user@parrot]--[~/Desktop]
└─$ ./program
1
1.000000
```

On peut faire ça dans l'autre sens float => vers int de la même manière.

On peut aussi le faire dans un calcul (mais attention au arrondi dans ce cas la)

resultat = (int) money/price ;

Ici resultat est int, money et price sont double.

Enregistrer plusieurs valeurs par un scanf()

On peut tout de suite mettre plusieurs valeurs dans un scanf()

```
1 #include <stdio.h>$
2 $
3 int main(){$
4     int a, b, c;$
5     printf("entre 3 valeurs\n");$
6     scanf("%d %d %d",&a,&b,&c);$
7     printf("valeur a %d\nvaleur b %d\nvaleur c %d\n",a,b,c);$
8 $
9 }
```

```

[user@parrot]--[~/Desktop]
$ ./program
entre 3 valeurs
45
58
12
valeur a 45
valeur b 58
valeur c 12
[user@parrot]--[~/Desktop]
$ ./program
entre 3 valeurs
45 12 63
valeur a 45
valeur b 12
valeur c 63

```

A noter on peut mettre les valeurs soit en appuyant sur *entrée* soit en les séparant par un espace

Enregistrer depuis un scanf() une valeur dans un tableau / array

```

1 #include <stdio.h>
2 $
3 int main(){
4     int ingredient[10], id, i;
5     for(i=0;i<10;i++){
6         scanf("%d",&ingredient[i]);
7     }
8     scanf("%d",&id);
9     printf("%d",ingredient[id]);
10 }

```

Pas besoin de passer par une variable en plus qui ferait dans notre exemple ;

```
scanf("« %d »",&variable_temporaire);
```

```
ingredient[i] = variable_temporaire;
```

Sizeof()

Sizeof() est un opérateur qui permet de connaître la taille en octet d'une variable ou d'un type de variable.

Celui-ci est exprimé en octet (bytes est le mot équivalent en anglais), c'est-à-dire 8 bits.

```

int main()
{
    printf("la taille en octet d'une variable int est de %d octet\n",sizeof(int));
    printf("la taille en octet d'une variable long est de %d octet\n",sizeof(long));
    printf("la taille en octet d'une variable double est de %d octet\n",sizeof(double));
    printf("la taille en octet d'une variable char est de %d octet\n",sizeof(char));
}

```

```
[user@parrot]--[~/Desktop]  
└─ $./program  
la taille en octet d'une variable int est de 4 octet  
la taille en octet d'une variable long est de 8 octet  
la taille en octet d'une variable double est de 8 octet  
la taille en octet d'une variable char est de 1 octet  
[user@parrot]--[~/Desktop]
```