# Mise à niveau en langage C

Infres 1A

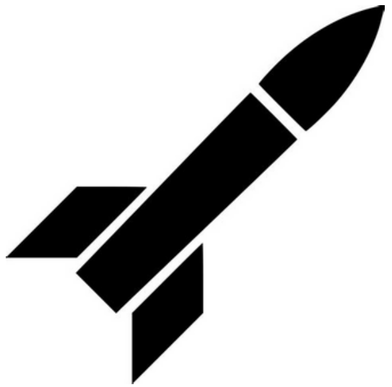
Enseignant: P. BERTIN-JOHANNET

# Introduction

# Le langage C

- Le langage C est utilisé lorsqu'on cherche à minimiser :
  - La vitesse d’execution
  - L'utilisation mémoire
  - le contrôle des instructions éxécutées

- Il est courament utilisé dans :
  - Les systèmes d’exploitation
  - Les drivers
  - Les navigateurs
  - Les microcontroleurs
  - Les jeux vidéos
  - Les programmes embarqués




# Pourquoi apprendre la programmation en C ?

- Le C est utilisé dans pratiquement tous les composants informatiques

# Pourquoi apprendre la programmation en C ?

- Le C est utilisé dans pratiquement tous les composants informatiques
- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)

# Pourquoi apprendre la programmation en C ?

- Le C est utilisé dans pratiquement tous les composants informatiques
- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)
- Selon les GAFAM, plus de 70% des failles de sécurité critiques originent de programmes écrits en C

# Pourquoi apprendre la programmation en C ?

- Le C est utilisé dans pratiquement tous les composants informatiques
- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)
- Selon les GAFAM, plus de 70% des failles de sécurité critiques originent de programmes écrits en C
- Un programme en C est plus rapide et utilise moins de mémoire en général qu'un programme ecrit dans un autre langage

# Pourquoi apprendre la programmation en C ?

- Le C est utilisé dans pratiquement tous les composants informatiques
- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)
- Selon les GAFAM, plus de 70% des failles de sécurité critiques originent de programmes écrits en C
- Un programme en C est plus rapide et utilise moins de mémoire en général qu'un programme ecrit dans un autre langage
- Une majeure partie des langages utilisés aujourd'hui sont inspirés du C

# Exemple de comparaison

- On considère ici un programme qui demande un nombre à l'utilisateur et affiche deux de ses facteurs

| Langage | Temps d'execution | Nombre d'appels système | Mémoire utilisée (virtuelle) | Mémoire disque nécéssaire |
|---|---|---|---|---|
| C | 8 ms | 52 | 2.5 Mo | 24 Ko |
| Python | 1 400 ms | 670 | 13 Mo | 6 000 Ko |
| Java | 86 ms | 185 | 623 Mo | 124 000 Ko |

# Particularités de la programmation en C

- On contrôle précisément la mémoire que l'on souhaite utiliser ainsi que les instructions executées

- Il faut donc souvent se poser les questions suivantes :
  - Quelles instructions binaires sont executées et quel est leur cout
  - Quelle quantité de mémoire est utilisée
  - Quels appels systèmes sont effectués
  - Comment sont organisées les données dans la mémoire

Cela demande donc un effort supplémentaire

# Les bases du C

# Compilation

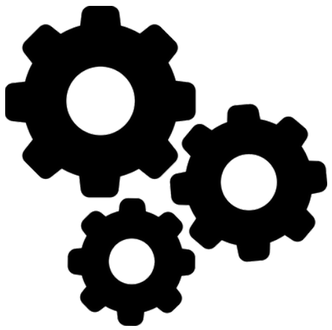
- Le langage C peut être traduit en langage binaire pour être éxécuté.
- On appelle ça la compilation. Le programme qui traduit est appelé le compilateur.

```c
#include <stdio.h>

main()
{
  int i=0;

  if (i == 1)
  {
    puts ("i is equal to one\n");
  }
  else
  {
    puts("i is NOT equal to one");
  }
}
```

Programme en C



Compilateur

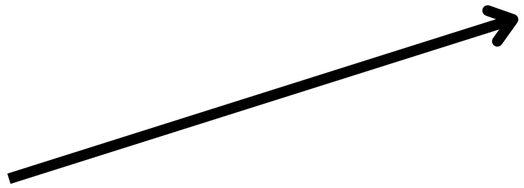
```
01000010 01101001 01101110
01100001 01110010 01111001
00100000 01000011 01101111
01100100 01100101 00100000
01010100 01110010 01100001
01101110 01110011 01101100
01100001 01110100 01101111
01110010
```

Fichier executable

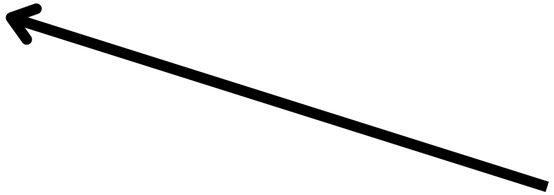
# Compilation

- Le compilateur que nous utiliserons dans ce cours s'appelle **gcc**
- On lui passe le nom du fichier à compiler puis le nom du fichier à créer ainsi :

*gcc main.c -o main.out*



Nom du fichier à compiler



Nom du fichier executable que l'on souhaite créer

# Les variables

- En C, une variable est définie par :
  - Un nom
  - Un type
- Elle permet d'associer un nom à un emplacement mémoire.
- Pendant l'execution du programme, on pourra modifier la valeur située dans cet emplacement mémoire
- Exemple: une variable ***α*** de type entier qui contiendra d'abord 5 puis 4 :

```
int a = 5;
a = a - 1;
```

# Types de données

- En C nous sommes obligés de préciser le type de chaque variable utilisée par notre programme
- Le type d'une variable indique :
  - La quantité de mémoire qui lui est reservée
  - Les opérations qu'on peut lui appliquer

# Les types de base

- Dans cette séance, nous allons utiliser les types suivants :

| **Nom** | **Représentation mémoire** | **Mémoire utilisée** | **Intervale de valeurs** |
|---|---|---|---|
| int | Nombre entier | minimum 2 octets | si 4 octets : -4 Milliards ; +4 Milliards |
| char | Nombre entier | 1 octet | 0 ; 255 |
| float | Nombre à virgule | 2 octets | - |

# Déclaration de variable

- Afin d'utiliser une variable il faut toujours la déclarer et lui donner un type, on écrit :
- type nom_variable;

- Pour une variable apellée **b** et de type entier on écrira :
- int b ;

- Pour une variable apellée **c** et de type charactère on écrira :
- char c ;

# Déclaration de variable

- On peut aussi donner une valeur à notre variable au moment de la déclaration :
- type nom_de_la_variable = valeur;

- Par exemple :
- ip ip_client = 192.168.1.1;
- int age = 52;

# Exemple :

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire :

# Exemple :

Programme:

```
> int a = 5;
  int b;
  char c = 10;
  float f = -1.8;
  b = a + c;
```

Mémoire :

# Exemple :

Programme:

```
int a = 5;
> int b;
char c = 10;
float f = -1.8;
b = a + c;
```

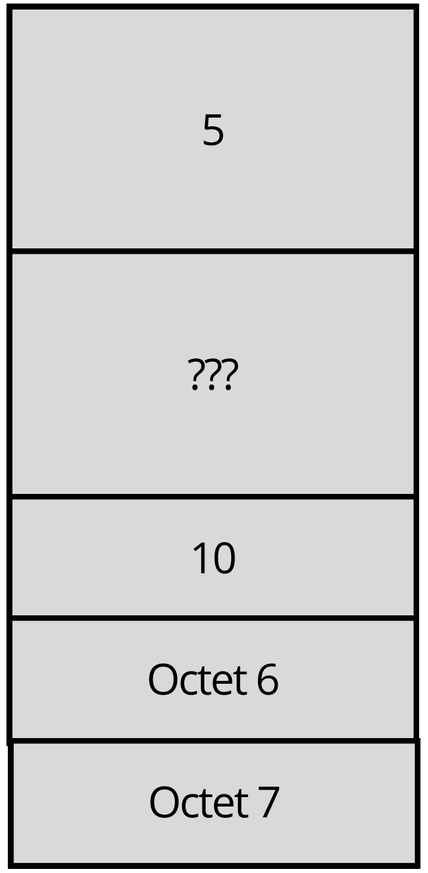
Mémoire :

# Exemple :

Programme:

```
int a = 5;
int b;
> char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire :

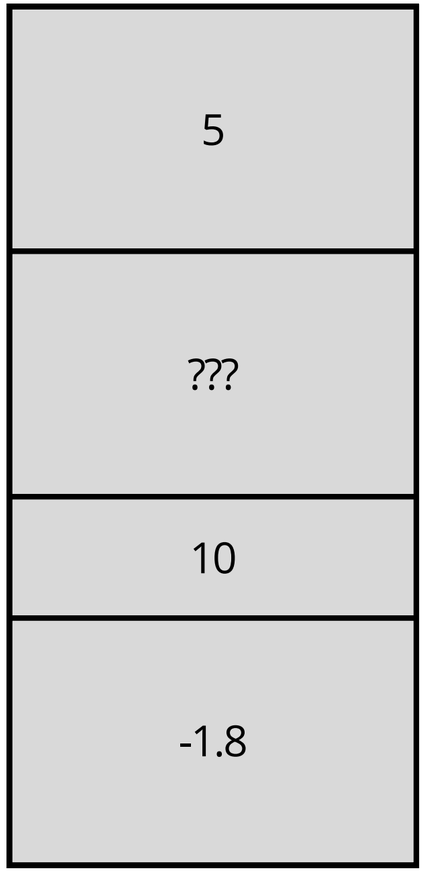


# Exemple :

Programme:

```
int a = 5;
int b;
char c = 10;
> float f = -1.8;
b = a + c;
```

Mémoire :

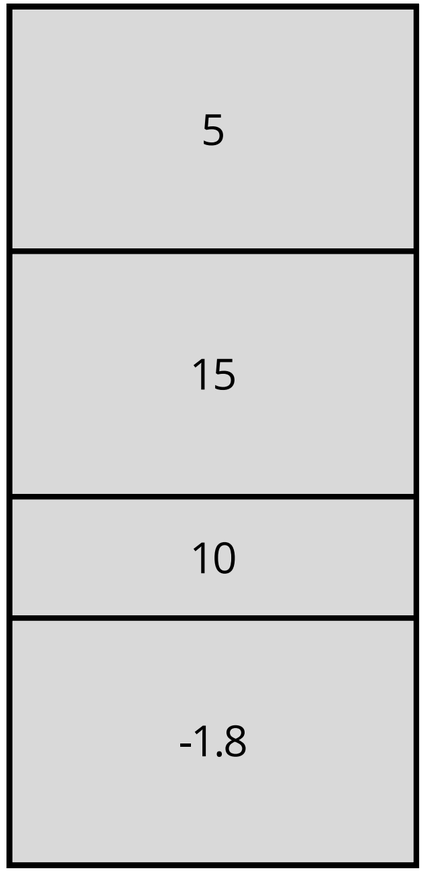


# Exemple :

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;
> b = a + c;
```

Mémoire :



# Les charactères

- Le type char permet aussi de représenter des charactères (lettre/symbole)
- On l'écrit donc entre deux apostrophes
- `char a = 'i';`
- `char b = '$';`
- `char c = '*';`

# La table ASCII (American Standard Code for Information Interchange)

- La table ASCII associe un nombre à tous les caractères usuels (états-uniens) sur 7 bits (128 positions)
- Les variables de types char utilisent cette table

| | | | | |
|---|---|---|---|---|
| 32 | | | 56 | 8 |
| 33 | ! | | 57 | 9 |
| 34 | " | | 58 | : |
| 35 | # | | 59 | ; |
| 36 | $ | | 60 | < |
| 37 | % | | 61 | = |
| 38 | & | | 62 | > |
| 39 | ' | | 63 | ? |
| 40 | ( | | 64 | @ |
| 41 | ) | | 65 | A |
| 42 | * | | 66 | B |
| 43 | + | | 67 | C |

# Exemple :

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4;
char d = 'e';
```

Mémoire :

# Exemple :

Programme:

```
> char a = 56;
  char b = '8';
  char c = 'a' + 4;
  char d = 'e';
```

Mémoire :

# Exemple :

Programme:

```
    char a = 56;
>   char b = '8';
    char c = 'a' + 4;
    char d = 'e';
```

Mémoire :

# Exemple :

Programme:

```
  char a = 56;
  char b = '8';
> char c = 'a' + 4;
  char d = 'e';
```

Mémoire :

# Exemple :

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4;
> char d = 'e';
```

Mémoire :

# Opérateurs sur les entiers

- Le C propose les opérateurs suivants sur les entiers :
- Les opérations classiques : (*, +, /, -)
- Le reste de la division euclidienne (ou "modulo") : %
- Exemple : 18 % 14 = 4
- Des opérateurs binaires qui appliquent une opération booléenne à chaque bit des deux entiers donnés (|, &, ^)
- Exemples :
- 1 | 2 = 0 31 & 16 = 16 5 ^ 30 = 25
- Des opérateurs qui déplacent les bits d'une variable dans un sens (>>, <<)
- 3 << 2 = 12
- Le complément à 1(négation bit à bit) ou "tilde" ~

# Afficher dans le terminal

- Pour afficher dans le terminal on utilise la fonction printf (pour print format)
- La syntaxe est la suivante :

```
printf(format, variables) ;
```

- Exemple: afficher un jour et une température

```
int jour = 27;
float temperature = 25.4;
printf("il fait %f le %d\n", temperature, jour);
```

# Chaine de format printf

- Durant cette séance, nous utiliserons les formats suivants :
  - %d : int
  - %f : float
  - %c : char (affiche le charactère dans la table ASCII)
- /!\ Attention : Sur la plupart des environnements, le texte ne sera pas affiché tant que vous n'aurez pas envoyé un saut de ligne : "\n"

# Exemple :

Programme:

```
char a = 56;
float b = 8,1;
int c = 4;
printf("%c -- %f -- %d\n", a, b, c);
```

# Exemple :

Programme:

```
char a = 56;
float b = 8,1;
int c = 4;
printf("%c -- %f -- %d\n", a, b, c);
```

Console :

8 -- 8.1 -- 4

# Lire dans le terminal

- Pour lire dans le terminal on utilise la fonction scanf (pour scan format)
- La syntaxe est la suivante :

```
scanf(format, &variable) ;
```

- Exemple: lire un age et une taille

```
int age;
float taille;
scanf("%d %f", &age, &taille);
```

# Chaine de format scanf

- Les formats de scanf sont les mêmes que printf :
  - %d : int
  - %f : float
  - %c : char (scanne le charactère dans la table ASCII)

- /!\ Attention : un espace dans scanf permet de sauter tous les charactères *"blancs"* (tabs, espaces, sauts de lignes) jusqu'au prochain charactère non "blanc"

# La condition if

- L'instruction if (en anglais : "si") permet d'executer du code uniquement si une condition est vraie.

```
if(condition){
        code
}
```

- Les lignes de code entre les accolades ne seront executées que si la condition est vraie

# Exemple :

```
int age;
printf("quel est votre age ?\n");
scanf("%d", &age);
printf("vous pouvez acheter : \njus de fruit\n");
if (age > 17){
    printf("vodka\n");
}
```

# La condition else

- L'instruction else ("sinon" en anglais) placée après un if permet d'executer du code si la condition était fausse.

```
if(condition){
        code_a
}
else{
        code_b
}
```

- Le code code_b sera executé uniquement si la condition est vraie

# Exemple :

```
if (age > 17){
    printf("Tarif : 8€ \n");
}
else {
    printf("Tarif : 6€ \n");
}
```

# Conditions

- En C, une condition est un entier. On la considère comme vraie ou fausse ainsi :
  - 0 => FAUX
  - N'importe quoi d'autre => VRAI
- Les opérations suivantes renvoient 1 ou 0 pour vrai ou faux :
  - Comparaison de nombres : >, <, ==, >=, <=
  - Algebre de bool : || (ou), && (et), ! (not)

# Opérateurs d'affectation

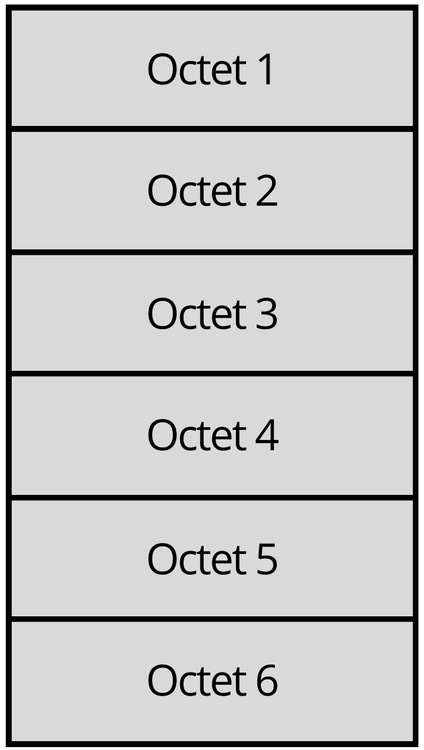
- Les opérateurs d'affectation permettent de changer la valeur d'une variable et de renvoyer le résultat
- c = b => affecte la valeur de b à c et renvoie la nouvelle valeur
- c += b => ajoute b à c (existe aussi pour *, /, -)
- c++ => ajoute 1 à c et renvoie l'ancienne valeur
- c-- => ajoute 1 à c et renvoie l'ancienne valeur
- ++c => ajoute 1 à c et renvoie la nouvelle valeur
- --c => enlève 1 à c et renvoie la nouvelle valeur

# Exemple

Programme :

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire :

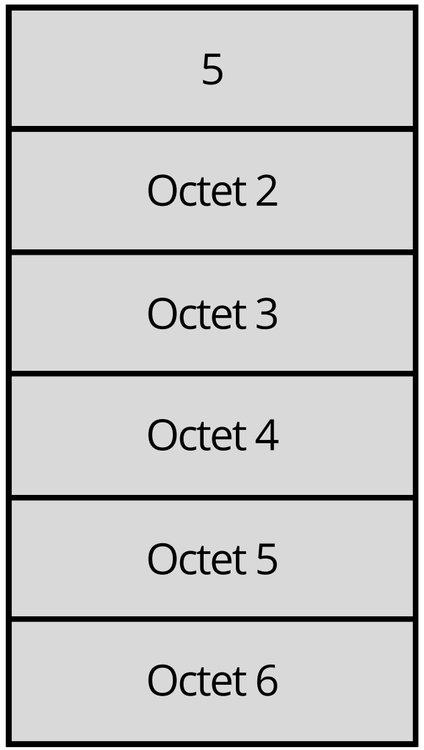


# Exemple

Programme :

```
> char a = 5;
  char b = (a = 2);
  b *= 2;
  a = --b + 4;
  char c = (b = a) + 1
```

Mémoire :

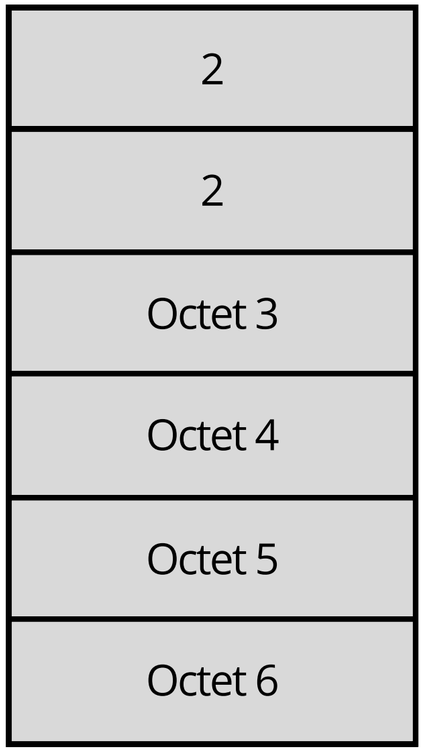


# Exemple

Programme :

```
  char a = 5;
> char b = (a = 2);
  b *= 2;
  a = --b + 4;
  char c = (b = a) + 1
```

Mémoire :

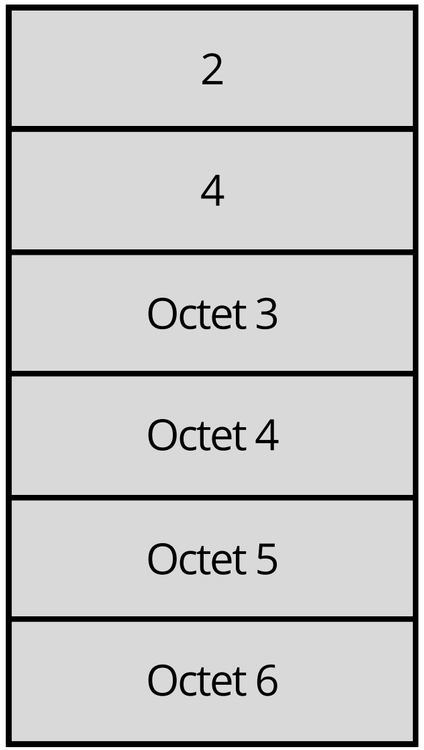


# Exemple

Programme :

```
char a = 5;
char b = (a = 2);
> b * =2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire :

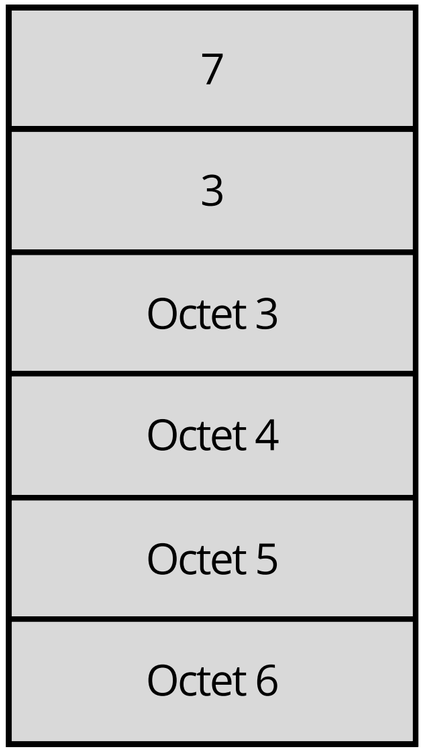


# Exemple

Programme :

```
char a = 5;
char b = (a = 2);
b *= 2;
> a = --b + 4;
char c = (b = a) + 1
```

Mémoire :



# Exemple

Programme :

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
> char c = (b = a) + 1
```

Mémoire :

# Priorité des opérateurs

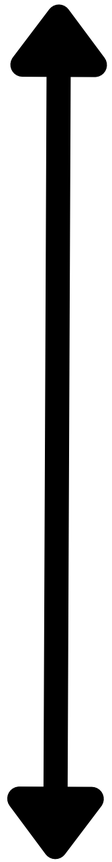
- Les opérateurs avec la priorité la plus élevée sont évalués en premier

- Exemple dans 2 + 3 * 5 on évalue la multiplication en premier : 2 + (3 * 5)

- Tous les opérateurs C ont une priorité associée

# Priorité des opérateurs

Priorité élevée



Priorité faible

| | |
|---|---|
| Parenthèses | () |
| Opérateurs unaires | ++ -- - ~ |
| Multiplication et division | / * % |
| Addition et soustraction | + - |
| Déplacement de bits | << >> |
| Comparaison | > >= < <= |
| Égalité | == != |
| Binaires (chacun une priorité différente) | & puis ^ puis \| |
| Logique booléenne | \|\| && |
| Affectation | = += -= *= /= |

# Programme complet en C

- Dans cette séance les programmes seront de la forme suivante :

```
#include "stdio.h"
int main(){
    code
}
```

# Exemple :

```
#include "stdio.h"
 int main(){
        printf("C'est la fin ! \n");
 }
```

# Mise en pratique

Si vous n'avez pas d'environnement linux vous pouvez utiliser :

https://www.onlinegdb.com/online_c_compiler