Introduction à GCC

- `-std=c99` : version de C
- `-pedantic` : Norme ISO du C
- `-Wall` : Affiche les warnings
- `-Wextra` : Affiche encore plus de warnings
- `-Werror` : transforme les warnings en erreurs
- `-Wvla` : Interdit les tableaux à tailles variables
- `-fstack-protector` : active la protection de la pile contre les d ́ebordements

Introduction à GCC : exemple de compilation

Makefile

Langage C

Langage C : les commentaires

```
// Un commentaire sur une ligne

/*
Un commentaire
sur plusieurs
lignes
*/
```

Langage C : les types

Nom	Taille (en bits)
char / unsigned char	8
short / unsigned short	16
int / unsigned int	32
long / unsigned long	32 ou 64
float	32
double	64
long double	128

Langace C : les variables

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

int main(int argc, char **argv) {
    int my_int = 10;
    char my_char;
    long double my_double = 1.12;
    return EXIT_SUCCESS;
}
```

Langace C : les constantes

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

int main(int argc, char **argv) {
    const int MY_CONST = 10;
    return EXIT_SUCCESS;
}
```

Langace C : alias de type

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

typedef unsigned char color_param_t;

int main(int argc, char **argv) {
    unsigned char color_red = 255;
    color_param color_blue = 255;
    return EXIT_SUCCESS;
}
```

Langace C : les tableaux

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

int main(int argc, char **argv) {
    int my_array[100];
    printf("La premiere case est : %d\n", my_array[0]);

    int my_array2[] = {25, 50, 75, 100};
    printf("La premiere case est : %d\n", my_array2[0]);

    return EXIT_SUCCESS;
}
```

Langace C : les `if` - `else`

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

int main(int argc, char **argv) {
    int a = 11;
    int b = 32;

    if (a > b) {
        printf("A is greater than b");
    } else if (a < n) {
        printf("A is lower than b");
    } else {
        printf("A is equal to b");
    }
    return EXIT_SUCCESS;
}
```

Langace C : les `if` - `else`

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

int main(int argc, char **argv) {
    int a = 11;
    int b = 32;

    if (a > b)
        printf("A is greater than b");
    else if (a < n)
        printf("A is lower than b");
    else
        printf("A is equal to b");

return EXIT_SUCCESS;
}
```

Langage C : les `switch`

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

int main(int argc, char** argv) {
int dice = 6;

switch (dice) {
case 1:
    printf("The dice roll is 1");
    break;
case 2:
    printf("The dice roll is 2");
    break;
case 3:
    printf("The dice roll is 3");
    break;
case 4:
    printf("The dice roll is 4");
    break;
case 5:
    printf("The dice roll is 5");
    break;
case 6:
    printf("The dice roll is 6");
    break;
default:
    printf("This is not a valid dice roll");
    break;
}
return EXIT_SUCCESS;
}
```

Langage C : boucle `while`

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

int main(int argc, char **argv) {
    int n = 0;
    while (n < 10) {
        if (n == 3)
            continue;
        if (n == 8)
            break;
        n++;
    }
    return EXIT_SUCCESS;
}
```

Langage C : boucle `while`

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

int main(int argc, char **argv) {
    int n = 0;

    while (n < 10)
        n++;

    return EXIT_SUCCESS;
}
```

Langage C : boucle `do while`

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

int main(int argc, char **argv) {
    int n = 0;
    do {
        if (n == 3) {
            continue;
        }
        if (n == 8) {
            break;
        }
        n++;
    } while (n < 10);

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

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

int main(int argc, char **argv) {

    for (int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

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

int main(int argc, char** argv) {

    for (int i = 0; i < 10; i++)
        printf("%d\n", i);

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

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

int main(int argc, char** argv) {

    for (int _ = 0; _ < 10; _++)
        printf("Loop");

    return EXIT_SUCCESS;
}
```

Langage C : fonctions et procédures

```c
#include <stdlio.h>

void say_hello() {
    printf("Hello\n");
}

void say_hello_n_times(int n) {
    for (int _ = 0; _ < 10; _++)
        printf("Hello\n");
}

int max(int a, int b) {
    if (a < b)
        return b;
    return a;
}
```

Langage C : fonction printf

- Elle utilise des indicateurs pour formatter le texte.
- Elle accepte un nombre variable de paramètres.

Langage C : la fonction main

- Retourne un entier qui indique l'état de fin du programme.
- Accepte des arguments via la ligne de commande.

Les pointeurs

- `<TYPE> *` : indique qu'il s'agit d'un pointeur vers une valeur de type 'TYPE'.
- `*<pointeur>` = valeur.
- `&<variable>` = pointeur.
- `<TYPE> **` : Un pointeur vers un tableau d'éléments du type 'TYPE'.

Les structures en C

- Sans alias de type

```
struct person_s {
    char[] name;
    unsigned int age;
};
```

Les structures en C

- Avec alias de type

```
typedef struct person_s {
    char[] name;
    unsigned int age;
} person_t;
```

Les structures en C : Utilisation

```c
typedef struct person_s {
    char[] name;
    unsigned int age;
} person_t;

person_t person = { "Tom", 25 };

printf("Name : %s, age : %d", person.name, person.age);
```

Les structures en C : Utilisation

```
typedef struct person_s {
    char[] name;
    unsigned int age;
} person_t;

person_t person = { "Tom", 25 };
person_t *pointer = &person;

printf("Name : %s, age : %d\n", (*person).name, (*person).age);

printf("Name : %s, age : %d\n", person->name, person->age);
```

Les énumérations en C

- Sans alias de type

```
enum direction_e {
    NORTH,
    EAST,
    SOUTH,
    WEST
};
```

Les énumérations en C

- Avec alias de type

```
typedef enum direction_e {
    NORTH,
    EAST,
    SOUTH,
    WEST
} direction_t;
```

Les énumérations en C : Utilisation

```c
typedef enum direction_e {
    NORTH,
    EAST,
    SOUTH,
    WEST
} direction_t;

void move(direction_t direction) {
    switch (direction) {
    case NORTH: printf("North\n"); break;
    case EAST: printf("East\n"); break;
    case SOUTH: printf("South\n"); break;
    case WEST: printf("West\n"); break;
    }
}
```

Les unions en C

- Sans alias de type

```
union integer_s {
    int an_integer;
    short a_short;
    long a_long;
};
```

Les unions en C

- Avec alias de type

```
typedef union integer_s {
    int an_integer;
    short a_short;
    long a_long;
} interger_t;
```

Les unions en C : Utilisation

Allocation et libération de mémoire

- `malloc(taille)` : pour allouer de la mémoire. Retourne un pointeur.
- `free(pointeur)` : pour libérer l'espace mémoire.
- `sizeof(type)` : pour connaître la taille (nombre de bits) d'un type.

Allocation et libération de mémoire : Exemple

```
typedef struct person_s {
    char[] name;
    unsigned int age;
} person_t;

person_t *person_pointer = malloc(sizeof(person_t));
person_pointer->name = "Tom";
person_pointer->age = 10;
free(person_pointer);
```

Les prototypes

```
void foo() {
    // ...
    bar();
    // ...
}

void bar() {
    // ...
    foo();
    // ...
}
```

Les prototypes

- Pour les fonctions :
  `<TYPE> <nom_de_la_fonction>([params]);`
- Pour les types :
  `typedef struct <nom structure> <nom alias type>;`

Compilation de plusieurs fichiers

Compilation de plusieurs fichiers

```
#IFNDEF <NOM>
    // Contenu du fichier
#ENDIF
```

```
#include <chemin_vers_le_fichier.h>
```

Manipuler les fichiers

- `fopen` pour ouvrir
- `fprintf` pour écrire
- `fgets` pour lire
- `fclose` pour fermer

Manipuler les fichiers : ouvir

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

int main(int argc, char** argv) {

    // Suppression du contenu + écriture
    FILE* write_file = fopen("file.txt", "w");

    // Conserve le contenu + écrit
    FILE* append_file = fopen("file.txt", "a");

    // Lecture
    FILE* read_file = fopen("file.txt", "r");

    return EXIT_SUCCESS;
}
```

Manipuler les fichiers : fermer

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

int main(int argc, char** argv) {

    FILE* myfile = fopen("file.txt", "w");
    // ...
    fclose(my_file);

    return EXIT_SUCCESS;
}
```

Manipuler les fichiers : écrire

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

int main(int argc, char** argv) {
    FILE *my_file = fopen("file.txt", "w");

    if (fprintf(my_file, "Some text to write\n") < 0)
        printf("Something went wrong");

    fclose(my_file);
    return EXIT_SUCCESS;
}
```

Manipuler les fichiers : écrire

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

#define BUFFER_SIZE 256

int main(int argc, char** argv) {
    FILE *my_file = fopen("file.txt", "r");

    char buffer[BUFFER_SIZE];
    int n = fgets(my_file, buffer);

    if (n < 0)
        printf("Echec de la lecture");
    else
        printf("Lu: %s", buffer);

    fclose(my_file);
    return EXIT_SUCCESS;
}
```

Les erreurs en C

Les bonnes pratiques

- Fonctions de création
- Fonctions de destruction
- Fonctions pour cloner
- Fonctions 'getter'
- Fonctions 'setter'
- Fonctions 'print'

Javadoc

- https://www.baeldung.com/javadoc

Optimisation

- Masques de bits
- L'arena allocator
- Fast inverse square root