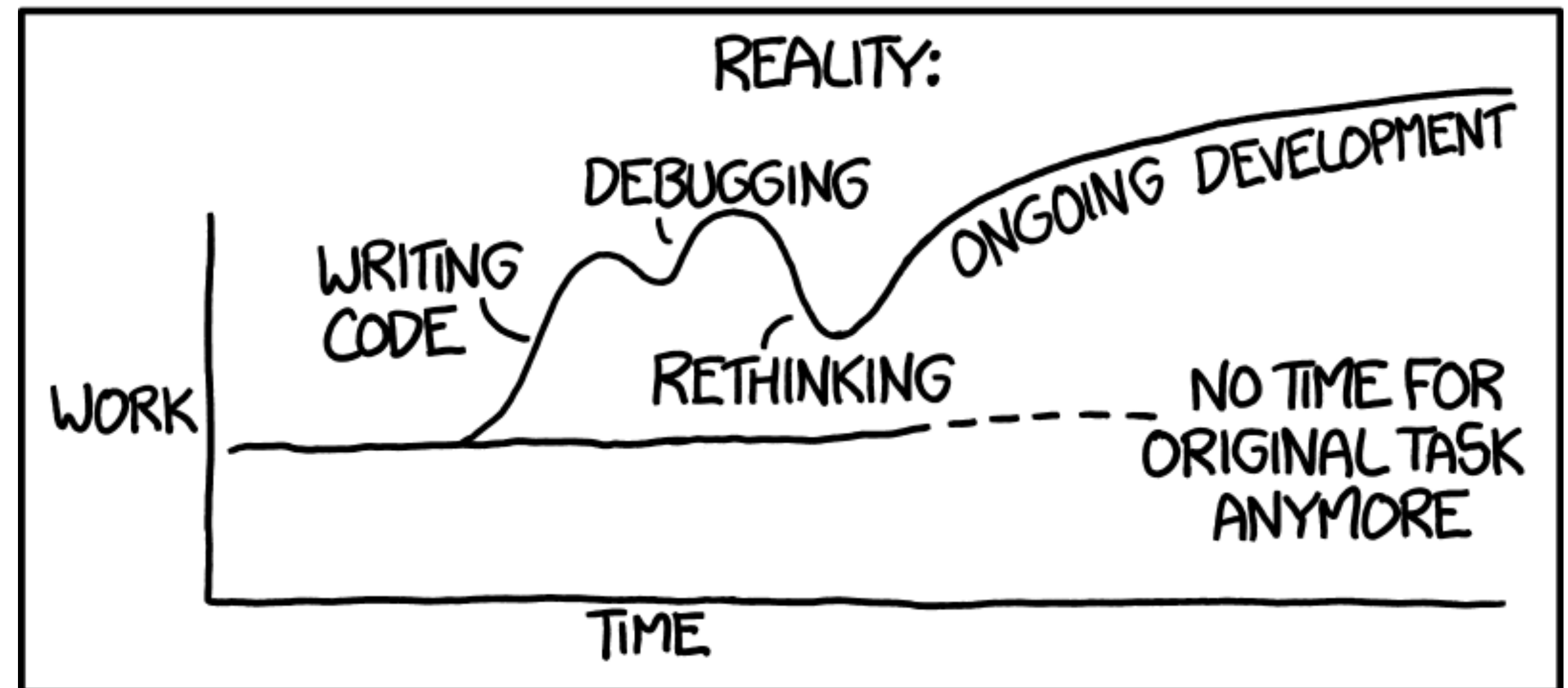
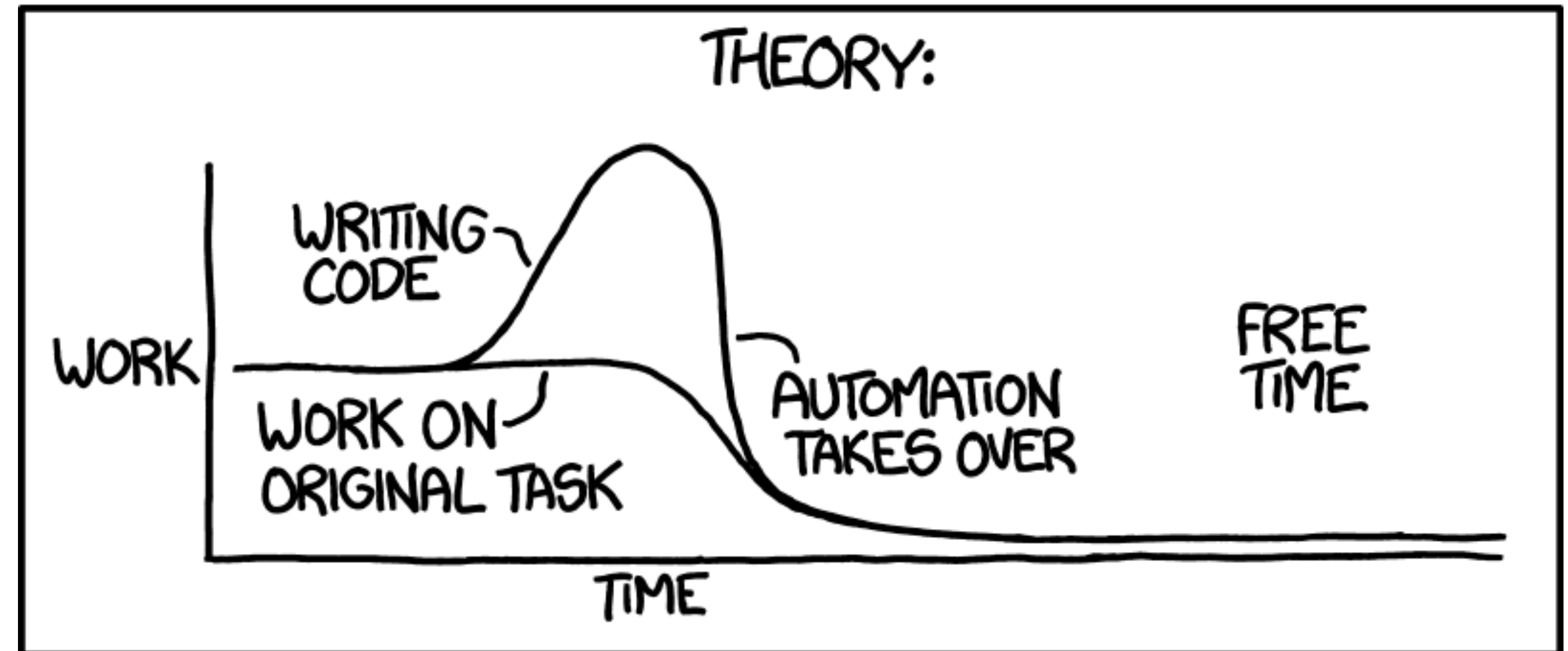


I/O

Entrée/sortie

ICC-C Cours 11

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



Enumérations

- Parfois nous voulons définir des constantes avec des valeurs consécutives
- Exemples:
 - Les jours de la semaine,
 - Les sept nains,
 - Des couleurs, etc.
- Il existe un `type` pour ça!

Enumérations

- Syntaxe:

```
enum nom
{
    constante_1,
    constante_2,
    ...
    constante_n
}
```

```
enum jour {
    LUNDI,      // 0
    MARDI,      // 1
    MERCREDI,   // 2
    JEUDI,      // ...
    VENDREDI,
    SAMEDI,
    DIMANCHE,
};
```

```
enum nain {
    Atchoum = 5,
    Dormeur,   // 6
    Grincheux, // 7
    Joyeux,    // ...
    Prof,
    Simplet,
    Timide
};
```

- Les constantes (entières) auront des valeurs consécutives qui commencent à 0
- On peut spécifier une valeur de départ, les valeurs suivantes seront aussi consécutives

Enumérations

Variables

- On peut définir une variable de type enum

```
enum jour j = MARDI;
```

- Ce sont en fait des entiers

```
printf("jour = %d (taille=%ld) \n",  
      j, sizeof(j));  
// Affiche: Enum jour = 1 (taille=4)
```

- On peut leur affecter une toute autre valeur

```
j = 20;
```

- Mais surtout on peut affecter ces constantes à d'autres variables:

```
int someday = JEUDI;
```

```
enum jour {  
    LUNDI,  
    MARDI,  
    MERCREDI,  
    JEUDI,  
    VENDREDI,  
    SAMEDI,  
    DIMANCHE,  
};
```

L'instruction `switch`

- Parfois nous voulons tester un nombre **discret** de valeurs
- Pour chaque valeur — exécuter un code différent

```
switch (expression)
{
case valeur_1:
    instruction_1
    break;
...
case valeur_k:
    instruction_k
    break;
default:
    instruction_par_défaut
}
```

Ne pas oublier `break` ici

Ni ici

L'instruction `switch`

Pourquoi `break`?

```
void afficher_contact(int tel)
{
    const char *nom;

    switch (tel)
    {
        case 791112233:
            nom = "Alice";

        case 791112244:
            nom = "Bob";

        default:
            nom = "Inconnu";
    }

    printf("Le numéro %d est %s\n", tel, nom);
}
```

`afficher_contact(790000000)`

Le numéro 12345 est Inconnu

`afficher_contact(791112244)`

Le numéro 791112244 est Inconnu

`afficher_contact(791112233)`

Le numéro 791112233 est Inconnu



Debugging

“Déboggage” 

- Il y a un problème dans mon code
- Que fais-je?
 - **Option 1**: Je demande à un assistant
 - **Option 2**: Je demande à ChatGPT
 - **Option 3**: Je mets des `printf` partout

Debugging

- Les `printf` peuvent nous aider à comprendre ce qui ne va pas!
- On veut suivre le cheminement dans le code qui a pu mener au comportement inattendu
- On affiche aux endroits clé

L'instruction `switch`

Pourquoi `break`?

```
void afficher_contact(int tel)
{
    const char *nom;

    switch (tel)
    {
        case 791112233:
            nom = "Alice";
            printf("C'est Alice!\n");

        case 791112244:
            nom = "Bob";
            printf("C'est Bob!\n");

        default:
            nom = "Inconnu";
            printf("C'est Inconnu!\n");
    }

    printf("Le numéro %d est %s\n", tel, nom);
}
```

```
afficher_contact(790000000)
```

C'est Inconnu!

Le numéro 12345 est Inconnu

```
afficher_contact(791112244)
```

C'est Bob!

C'est Inconnu!

Le numéro 791112244 est Inconnu

```
afficher_contact(791112233)
```

C'est Alice!

C'est Bob!

C'est Inconnu!

Le numéro 791112233 est Inconnu

L'instruction `switch`

Pourquoi `break`?

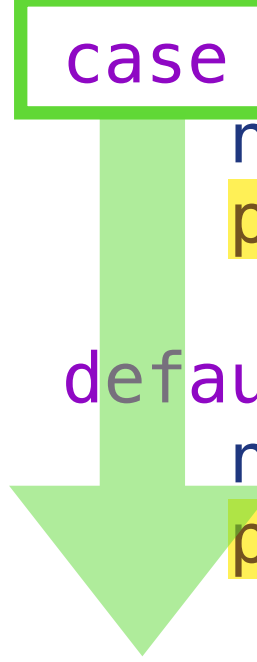
```
void afficher_contact(int tel)
{
    const char *nom;

    switch (tel)
    {
        case 791112233:
            nom = "Alice";
            printf("C'est Alice!\n");

        case 791112244:
            nom = "Bob";
            printf("C'est Bob!\n");

        default:
            nom = "Inconnu";
            printf("C'est Inconnu!\n");
    }

    printf("Le numéro %d est %s\n", tel, nom);
}
```



`afficher_contact(791112244)`

C'est Bob!

C'est Inconnu!

Le numéro 791112244 est Inconnu

- Saute directement au bon endroit
- Mais continue et exécute aussi `default`
- On a oublié les `break`

L'instruction `switch`

Pourquoi `break`?

```
void afficher_contact(int tel)
{
    const char *nom;

    switch (tel)
    {
        case 791112233:
            nom = "Alice";
            break;

        case 791112244:
            nom = "Bob";
            break;

        default:
            nom = "Inconnu";
            // On peut mettre break, mais sans effet
    }

    printf("Le numéro %d est %s\n", tel, nom);
}
```

- D'habitude on met des `break` après chaque cas
- Alors c'est comme un enchaînement de `if - else`

```
if (tel == 791112233)
{
    nom = "Alice";
}
else
{
    if (tel == 791112244)
    {
        nom = "Bob";
    }
    else
    {
        nom = "Inconnu";
    }
}
```


L'instruction `switch`

Pourquoi `break`?

```
int weekend(enum jour j)
{
    switch (j)
    {
        case LUNDI:
        case MARDI:
        case MERCREDI:
        case JEUDI:
        case VENDREDI:
            return 0;

        case SAMEDI:
        case DIMANCHE:
            return 1;

        default:
            return -1;
    }
}
```

- Si on choisit de ne pas mettre `break` après un cas = *fallthrough* (tomber-à-travers)

Debugging

- Parfois on peut utiliser un outil qui s'appelle **le Debugger**
- Le debugger original = gdb - pas d'interface graphique 🤨
- Concepts importants:
 - **Breakpoint** = point où l'exécution s'arrêtera pour inspection de l'état 
 -  **Step-in** = rentrer dans l'exécution d'une fonction
 -  **Step-out** = exécuter jusqu'à sortir d'une fonction
 -  **Step-over** = évaluer l'instruction suivante, sans suivre les appels de fonction

Alias de type

typedef

- Syntaxe:
`typedef type alias;`

- Exemples:

```
typedef int *pointeur_int;  
typedef char string99[100]; // Pour les tableaux le [] vient après l'alias  
typedef float petit_réel;  
typedef double gros_réel;  
typedef char **double_pointeur_char;
```

```
pointeur_int ptr;  
string99 coucou = "coucou";
```


Alias pour les struct

```
typedef struct personal_info personal_info_t;
```

Le vrai type

Alias

- Le nouveau type `personal_info_t` est synonyme de la structure `struct personal_info`
- Pour définir une variable on peut maintenant écrire:

```
personal_info_t bob;
```

```
struct personal_info
{
    char nom[30];
    char prénom[30];
    int année;
    int taille_cm;
    float poids_kg;
};
```

Fichiers

- Un fichier réside dans le stockage “long-terme” de l’ordinateur
- SSD, Disque-dur, etc.
- Il contient une suite d’octets
- Il peut être trop gros pour être “chargé” dans la mémoire
- On peut lire, écrire, ou modifier des fichiers

Fichiers binaires vs. texte

- Tous les fichiers peuvent être lus en **mode “binaire”** = une suite d’octets
- Ils ont un **format** spécifique à l’application qui les écrit/lit
- **Exemples:** Un document Word, une présentation Keynote, la sauvegarde d’un jeu vidéo, un fichier exécutable
- Certains fichiers sont censés contenir uniquement des caractères
= **fichiers texte**
- **Exemples:** Un fichier source .c, une page web .html, etc.

message.txt

Dear Mr Potter,
We are pleased to inform you that you have been accepted at Hogwarts School of
Witchcraft and Wizardry.

	'D'	(68)		'e'	(101)		'a'	(97)		'r'	(114)		' '	(32)		'M'	(77)		'r'	(114)	
	' '	(32)		'P'	(80)		'o'	(111)		't'	(116)		't'	(116)		'e'	(101)		'r'	(114)	
	' ,'	(44)		'\n'	(10)		'W'	(87)		'e'	(101)		' '	(32)		'a'	(97)		'r'	(114)	

...

Ouvrir un fichier

- On veut lire le contenu d'un fichier
- ???
- Comme pour `malloc`, on demande au système d'exploitation...
- On appelle la fonction `fopen` qui retourne un pointeur vers un objet `FILE`

`FILE` est un alias
pour une struct
de `stdio.h`

```
FILE* file = fopen("message.txt", "r");
```

le chemin (*path*) vers le
fichier qu'on veut lire

en quel "mode" on veut
ouvrir le fichier

Modes

Read (Lecture): “r”

- On peut lire depuis le **début** du fichier

Write (Écriture): “w”

- On **crée** un nouveau fichier
-   Si un fichier du même nom existe, il sera **écrasé** 

Append (Ajouter): “a”

- On écrit **à la fin** d'un fichier existant

—suivi par... (optionnel, utile sous Windows)

Binary: “b”

- On lit des octets

Text: “t”

- On lit des caractères

Le curseur

- Quand on ouvre un fichier, il existe un **curseur** qui indique où on se trouve



```
FILE* file_read = fopen("message.txt", "r"); // Read mode
```



```
FILE* file_append = fopen("message.txt", "a"); // Append mode
```

Lecture en mode texte

fscanf

Texte

- On lit avec fscanf, c'est comme scanf mais depuis un fichier

```
char buffer[100];  
int items_read;
```

```
FILE* file_read = fopen("message.txt", "r"); // Read mode
```

```
items_read = fscanf(file_read, "%s", buffer);
```

+ strlen(buffer)



buffer: "Dear"
items_read: 1

Mais où est passé mon curseur?

`ftell`

- On aimerait retrouver la position du curseur par rapport au début du fichier
- Il faut utiliser la fonction `ftell` ~ “f-raconte”

```
FILE* file_read = fopen("message.txt", "r"); // Read mode
```

```
items_read = fscanf(file_read, "%s", buffer);
```

```
printf("Offset = %d\n", ftell(file_read));
```

```
// Affiche: Offset = 4
```

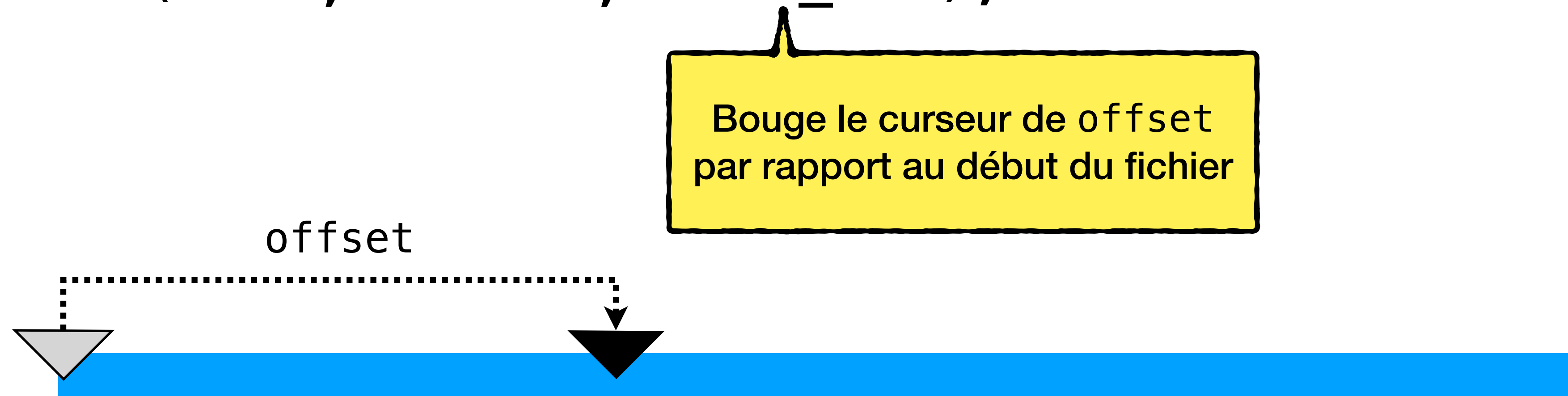


Bouger le curseur

fseek

- Quand on lit/écrit le curseur bouge à l'endroit où on a fini de lire/écrire
- Parfois on aimerait aller à un endroit précis dans le fichier

`fseek(file, offset, SEEK_SET);`

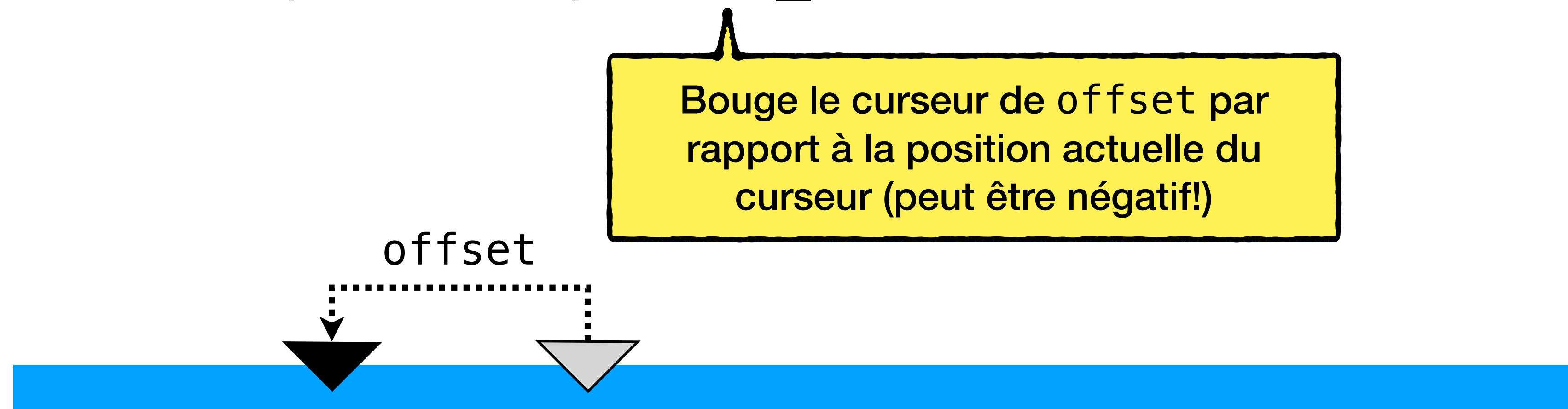


Bouger le curseur

fseek

- Quand on lit/écrit le curseur bouge à l'endroit où on a fini de lire/écrire
- Parfois on aimerait aller à un endroit précis dans le fichier

`fseek(file, offset, SEEK_CUR);`



Bouger le curseur

fseek

- Quand on lit/écrit le curseur bouge à l'endroit où on a fini de lire/écrire
- Parfois on aimerait aller à un endroit précis dans le fichier

`fseek(file, offset, SEEK_END);`

Bouge le curseur de `offset` par rapport à la fin du fichier (doit être négatif!)



Rembobinez!

- On aimerait relire depuis le début du fichier

```
FILE* file_read = fopen("message.txt", "r"); // Read mode
```

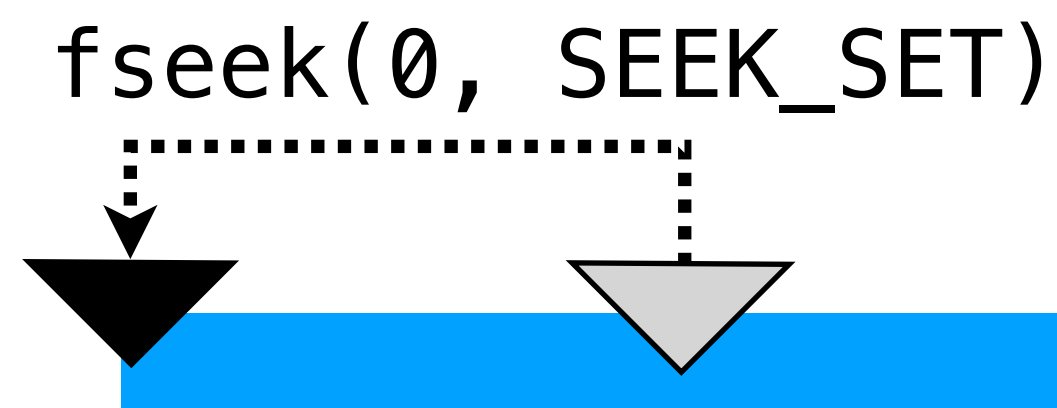
```
items_read = fscanf(file_read, "%s", buffer);
```

```
fseek(file_read, 0, SEEK_SET); // Revenir au début du fichier
```

```
char new_buffer[100];  
fscanf(file_read, "%s", new_buffer);
```

On vient de relire le premier
mot du fichier dans
new_buffer

new_buffer: "Dear"
buffer: "Dear"
items_read: 1



Écriture en mode texte

fprintf

Texte

- On peut aussi écrire dans des fichiers en **mode texte** avec fprintf:

```
int items_written;
```

```
FILE* file_write = fopen("message.txt", "w"); // Write mode
```

```
items_written = fprintf(file_write, "ICC %d\n", 2025);
```

```
+ strlen("ICC 2025\n")
```



Fichier écrasé par
l'ouverture en mode "w"

```
items_written: 1
```

Erreurs...

- Souvent quand on interagit avec l'I/O il y a des **erreurs** - et il faut vivre avec!
- **“Le fichier à lire n'existe pas”, “Pas assez de droits pour lire/écrire”, ...**
fopen retournera NULL — il faut tester si l'objet FILE* retourné est **valide**
- **“On a atteint la fin du fichier, il n'y a plus rien à lire!”**
fscanf / fread retournent moins d'éléments lus que prévu

On peut tester si on a atteint **la fin du fichier** avec
feof(file)

EOF = *End Of File*

cette fonction retourne 1 si le fichier est terminé, ou 0 sinon