Architecture & Programmation Système

Langage C:

Structure basique d'un programme C

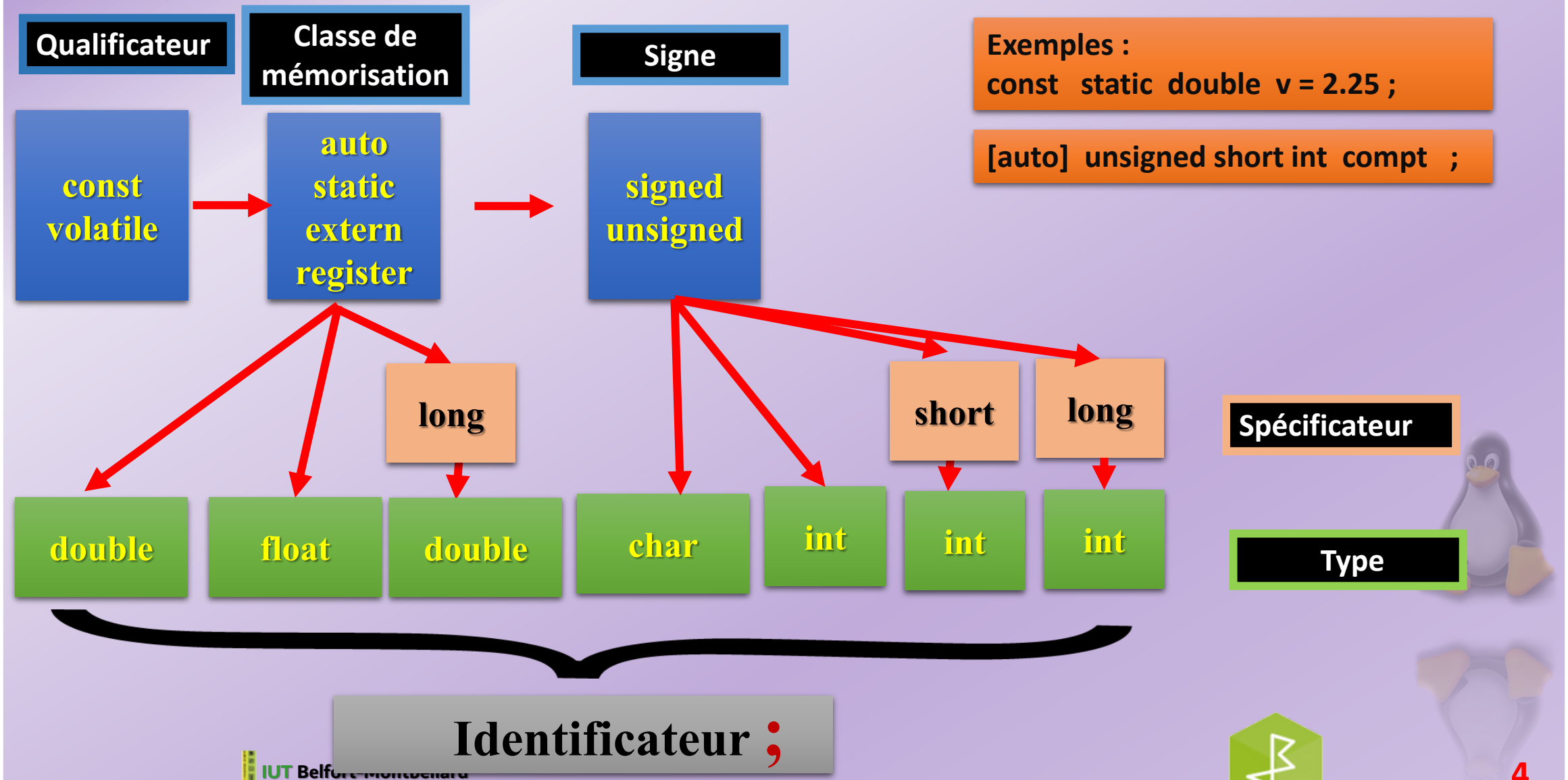
Caractéristiques du langage C

- Langage CPL (combined Programming Langauage) en 1960
- Langage BCPL (Basic CPL) en 1966
- Langage B Ken Thompson en 1970
- Création du langage C par Denis Ritch en 1972

Caractéristiques du langage C

- Qualifié de langage de bas niveau
- Est une suite finie non vide d'instructions organisée sous forme de fonctions
- Possède des types simples (char, int , double, …)
- Possède aussi des structures et des unions
- Ne gère pas les objets
- Un projet C peut être réalisé avec plusieurs fichiers
- Un fichier particulier contient la fonction mère (main)

Déclaration de variable/constante/volatile



Exemples :

```
const   static  double  v = 2.25 ;
[auto]  unsigned short int  compt   ;
```

Prototype du fichier hello.h

```
void printMessage(char*);
void usage();
```

Définition du fichier hello.c

```
#include <stdio.h>

void printMessage(char* msg) {
printf("%s\n",msg);
}
void usage() {
fprintf(stderr,"usage : hello
msg_to_print\n"); }
```

fichier essai.c pour le test

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

int main(int argc, char** argv) {
  if (argc!= 2) {
    usage();
    exit(1);
  }
  printMessage(argv[1]);
  return 0;
}
```

Complitation de hello.c :

gcc -Wall -c hello.c → hello.o

Compilation de essai.c :

gcc -Wall -c essai.c → essai.o

Edition de liens :

gcc hello.o essai.o -o essai → essai

Exécution:

./essai ou essai

Remarque : la compilation de hello.c indique Erreurs sur printf et fprintf

- Utiliser : whereis fonction pour trouver
- le n° de page du manuel de la fonction

Whereis printf ← **Commande**

printf: /usr/bin/printf /usr/include/printf.h
/usr/share/man/man1/printf.1.gz
/usr/share/man/man3/printf.3.gz

- Utiliser : man n° fonction

man 3 printf ← **Commande**

NOM

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf - Formatage des sorties

SYNOPSIS

#include <stdio.h> ← Fichier à inclure

int printf(const char *format, ...); ← Spécification

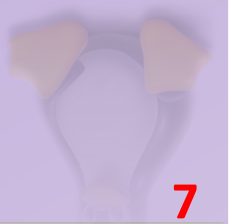
DESCRIPTION

Les fonctions de la famille printf() produisent des sortie en accord avec le format décrit plus bas …

- La compilation de hello.c indique deux erreurs sur : printf et fprintf
- Dans le fichier hello.c, il faut ajouter : #include <stdio.h>
- stdio : Standard input/output

- La compilation de essai.c indique deux erreurs sur : usage et `printMessage`
- Dans le fichier essai.c, il faut ajouter : #include "hello.h"

- La compilation de essai.c indique encore une erreur sur : exit
- Dans le fichier essai.c, il faut ajouter : #include <stdlib.h>
- stdlib : Standard libairié C



Règle 1 :

- Déclarer les fonctions (sauf main) dans un ou plusieurs fichiers d'inclusion (.h)
- Écrire les prototypes des fonctions (méthodes) sans leur code.
- Exemple : `int fonc(char , double) ;`

Règle 2 :

- Pour chaque fichier.h, il faut créer un fichier.c dont le contenu est la définition de chaque fonction (méthode)
- Inclure les fichiers d'inclusion (.h) contenant les déclarations des fonctions utilisées par ce fichier (.c).
- Exemple : `int fonc(char c , double d ) { … ; }`

Règle 3 : Un programme (.c) pour tester les règles 1 & 2.

- Les inclusions de fichiers (.h) sont de la forme :

`#include <nom_fichier.h>` Pour les fichiers standards.

`#include "[chemin/]nom_fichier.h"` Pour les fichiers créés par le développeur.

Application à l'exemple :

Règle 1 : Création de hello.h avec deux prototypes

```
void  usage(void) ;
void  printMessage(char*) ;
```

Règle 2 : Création de hello.c avec deux définitions de fonctions

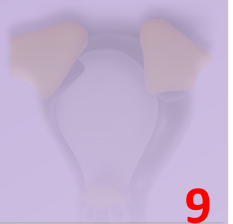
- Utilise : `#include <stdio.h>`
- Définitions de deux fonctions :

```
void  usage(void) { …  ; }
void  printMessage(char*  msg) { …  ; }
```

Règle 3 : Un programme essai.c pour tester les règles 1 & 2.

- Utilise les inclusions suivantes :

`#include <stdlib.h>`	Pour un fichier standard.
`#include "hello.h"`	Pour un fichiers créé par le développeur.



Règles sur le nombre (argc) et la liste (argv) de paramètres:

Indications : int main(int argc , char argv)**

- **argc** : indique le nombre de paramètres (y compris le nom du programme)
- **argv** : est un tableau de chaînes de caractères
  - **argv[0]** contient le nom du programme
  - **argv[1]** contient le 1er paramètre
  - **argv[2]** contient le 2er paramètre
  - …
  - **argv[n]** contient le ner paramètre

Dans la fonction main, il faut réaliser le test suivant :

Si argc est non valide et la taille et/ou le contenu des **paramètres** sont non valides
Alors
- **Afficher un message d'erreur approprié**
- **Quitter le programme (exit)**

Fsi

Pourquoi faire des tests sur **argc** et **argv** ?

- Un pirate peut créer un paramètre contenant en fait du code assembleur

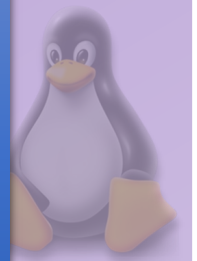
- **Si** le programme contient certaines failles de programmation **Alors**
  - L'exécution peut « sauter » vers le code assembleur
  - Le programme exécute une tache non prévue

- **Si** de plus, le programme a les droits root **Alors**
  - Possibilité de corrompre le système

- Arrêter l'exécution du programme en renvoyant une valeur au système d'exploitation
- utiliser **exit(n);** ou **return n;** dans **main()**

  n = 0 si la terminaison du programme est normale

  n ≥ 1 si la terminaison du programme n'est pas normale



Pour un projet C :

- Créer un ou plusieurs **fichier .h** contenant les **prototypes** des fonctions créées.
- Définir chaque fonction dans les **fichiers.c** correspondant à chaque fonction des **fichier.h**
- **Inclure** les **fichiers.h** adéquats dans les **fichier.c**
- Dans **main**, tester le **nombre** et les **paramètres** (taille, contenu)
- Dans **main**, finir proprement l'exécution en utilisant :
  exit(0); ou return 0; pour une fin normale
  ou
  exit(n); ou return n; (avec **n ≥ 1**) en cas d'erreur

Architecture & Programmation Système

Structure basique d'un programme C

Fin cours