A3P(AL)

Cours C1

Sommaire

1. Le cycle de développement
2. Le pré-processeur (quizz)
3. Les types pré-définis
4. Les types définis par le programmeur
5. Les affichages (quizz)
6. Les différences avec Java
7. `const`
8. La Standard Library du C
9. La compilation séparée (quizz)

1.1 Cycle de développement

3 phases dans la « compilation » :

1. phase de pré-processing : **#**
(substitution de texte)

2. phase de compilation : « habituelle »
`sin` inconnu => **`math.h`**

3. phase d'édition de liens :
avec les bibliothèques standard
ou d'autres morceaux de programme
`sin` inconnu ==> **`-lm`** *(pas de pb de syntaxe)*

1.2 Édition de liens

- Même si programme en un seul morceau !
  (édition de liens avec les fonctions système)
- `double y = fabs( x );`
- Compilation =>
  implicit declaration of function ‘fabs’
- `#include <math.h>`
- Édition de liens =>
  undefined reference to ‘fabs’
- `-lm` *(présent dans* ***mycc****)*

2. Le pré-processeur

- `#include <fichier.h>`

- `#define MOT par quoi remplacer`
  attention à `;` et `//`

- Il existe d'autre utilisations de `#define` (macros) et d'autres directives de compilation (`#ifdef`)

QUIZZ

- ```
  #define carre(X) X*X
  int n=3;    int v = carre(n+1);
  ```

Valeur de v ?

A **7**	**D** **autre**
B **9**	**C** **16**
  ```

3. Types prédéfinis

- Portage d'une machine/d'un système à l'autre : un programme peut ne plus fonctionner à cause du nombre d'octets !

- Existence de `signed` et `unsigned`

- `int, long, double,`
  `short, byte, float, char`

4. Types définis par le programmeur

- **`typedef`** définition de type Nouveau_nom`;`

- `typedef enum { FALSE, TRUE } Bool;`
  équivalent aux 3 lignes (mais meilleur !) :

```
#define FALSE 0
#define TRUE  1
typedef  int  Bool;
```

- **`typedef`** type_éléments Type_tableau **`[]`** `;`

5.1 Affichages

- ```
  System.out.print( "une phrase" );
  printf( "une phrase" );
  ```

- ```
  System.out.println( "une phrase" );
  printf( "une phrase\n" );
  ```

- ```
  System.out.println( "une " + unEntier
  + " phrase " + unReel );
  printf( "une %d phrase %lf\n",
          unEntier, unReel );
  ```

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

QUIZZ

- `printf( ? , unEnt, unRéel, unEnt);`

Quel format est correct ?

A **"Voici les 3 nbs :"**	**D** **Aucun des trois**
B **"%d, %lf, %d"**	**C** **"i=%d, x=%d, j=%d"**

5.2 Autres formats pour `printf`

- `%4d` ou `%04d` ou `%4.1f`
- `%x` : en hexadécimal
- `%s` : chaîne de caractères
  (voir TP suivant)
- `%%` : le caractère `%`

6. Différences avec Java

- Ni classes, ni objets, ni protection, ni exception, …
- fonctions/procédures AVANT le `main`
  (et avant d'être utilisées)
- absence de paramètre ==> `void`
  (ne rien mettre ==> ne plus vérifier)
- pas de surcharge :
  `int abs(int pE)` **et** `double abs(double pR)`
  ==> erreur fonction déjà définie, d'où `fabs`
- Autres différences : *voir le document en ligne*

7. `const`

- équivalent de `final` en Java

- on doit le mettre pour tous les paramètres
  (sauf les cas qui seront vus à la séquence n°2)

- mais avec certains vieux cc, l'utilisation de
  `const int TAILLE=10;`
  peut être refusée pour créer les tableaux ==>
  `#define TAILLE 10`

8. La *Standard Library* du C

- `ctype.h` ➔ `Character`
- `math.h` ➔ `Math`
- `stdlib.h` ➔ `java.lang.*`
- `string.h` ➔ `String`
- `time.h` ➔ `Calendar`

9. Compilation séparée

- Objectif : pouvoir compiler SÉPARÉMENT différents morceaux du programme complet.
- Répartir les instructions dans des fichiers **`.c`** et utiliser des **`#include "fichier1.c"`** ne servirait à rien !
- Mettre uniquement des déclarations dans les .h (dont prototypes de fonctions = *signatures*)
- Compiler sans édition de liens chaque morceau : **`mycc -c fichier1.c`** → **`fichier1.o`**
- Faire l'édition de liens entre tous les morceaux : **`mycc fichier1.o fichier2.o -o prog.exe`**

QUIZZ

- Un programme est divisé en 3 paquets de fonctions, en plus du **`main`**.

Combien de fichiers à créer ?

A **4**	**D** **Autre**
B **7**	**C** **8**

QUIZZ

- Un programme est divisé en 3 paquets de fonctions, en plus du **`main`**.

Combien de commandes **`mycc`** ?

A **3**	**D** **Autre**
B **4**	**C** **5**