

Systèmes Embarqués 1 : Travail écrit no 2.

Nom :

Prénom :

Classe : T-2/I-2

Date : 15.01.2015

Problème n° 1 (10 points) (Processus de développement en C, passage d'arguments et scope (portée) des variables et méthodes)

- a. Citez et décrivez succinctement le rôle/la fonction des 4 types de fichiers principaux faisant partie d'un développement d'applications en C (fichiers créés par le développeur ou générés par la chaîne d'outils).

- b. Ajoutez le mot clef (keyword) permettant de modifier la portée des 3 fonctions ci-dessous comme indiqué.

fichier : file1.c

```
void foo1();           // déclaration d'une fonction externe au fichier « file1.c »
                        // (globale à l'ensemble du programme)

void foo2(int) {...}    // définition d'une fonction globale à l'ensemble du programme

void foo3(float) {...} // définition d'une fonction locale
                        // (seulement visible dans le fichier « file1.c »)
```

- c. A l'aide d'un exemple, citez les deux techniques utilisées pour le passage d'arguments à des fonctions.
Pour chaque technique, on montrera un argument en lecture seule (RO) et un en lecture/écriture (RW).

- d. Adaptez le code ci-dessous afin d'adapter la portée des variables v1 à v5 au commentaire associé et indiquez la valeur initiale de la variable v5 si celle-ci n'était pas initialisée.

fichier : file2.c

```
int v1 = 0;             // variable globale pour toutes les fonctions de ce fichier,
                        // mais pas accessible à l'extérieur

int v2 = 0;             // variable globale accessible par tous les fichiers du
                        // programme et définie dans ce fichier

int v3;                 // variable globale au programme, mais définie hors du fichier

void foo5 () {
    int v4 = 0;          // variable rémanente et locale à la fonction

    int v5 = 0;          // variable locale à la fonction
}
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 2 (10 points) (Interface C)

Implémentez en C l'interface de la bibliothèque « message queue » (header file « msg_queue.h » contenant les déclarations des structures de données, des constantes et des fonctions) permettant d'émettre et de recevoir des messages. L'interface comprendra 5 méthodes : une pour initialiser la bibliothèque (**init**), une pour créer une queue de messages (**create**), une pour détruire une queue (**destroy**), une pour émettre un nouveau message (**post**) et une pour recevoir (**recv**) un message.

La méthode « **create** » retournera l'identifiant de la « message queue » avec un entier positif. La méthode « **post** » permettra de spécifier la « message queue », la priorité du message (**low**, **medium**, **high**) ainsi qu'un message quelconque (une chaîne de caractères et une taille). La méthode « **recv** » permettra de recevoir le message le plus prioritaire d'une « message queue ». Elle retournera le message au moyen de la structure « **struct msg_queue_message** » composée des champs suivants :

- Priorité du message: énumération.
- Timestamp : valeur entière non-signée.
- Message: tampon de taille fixe permettant de stocker des données quelconques.
La taille maximale du tampon sera définie par une constante (symbole), valeur 1024.
Le nombre de données stockées dans le tampon sera indiqué par l'attribut « **msg_sz** »
- Attribut permettant de chaîner dynamiquement les messages dans une liste.

Fichier : **msg_queue.h**

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 3 (10 points) (Programmation C)

- a. Implémentez dans les règles de l'art la fonction de la bibliothèque standard C afin de satisfaire aux spécifications ci-dessous.

/ Description*

The C library function `char *strncat(char *dest, const char *src, int n)` appends the string pointed to by `src` to the end of the string pointed to by `dest` up to `n` characters long.

Parameters

`dest` -- This is pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string which includes the additional null-character.

`src` -- This is the string to be appended.

`n` -- This is the maximum number of characters to be appended.

Return Value

This function returns a pointer to the resulting string `dest`.

**/*

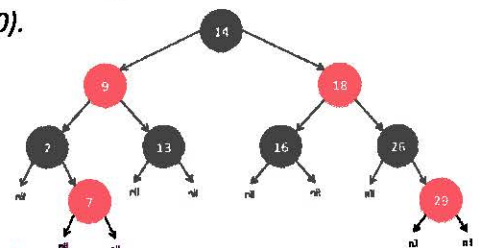
`char *strncat(char *dest, const char *src, int n)`

Implémentez la méthode « `int get_nb_eles (const struct red_black_element* root);` » permettant de calculer le nombre d'éléments contenus dans d'un arbre de type « red-black ».

Pour rappel, les feuilles vides sont indiquées par un pointeur NULL (0).

```
struct red_black_element {
    struct red_black_element* right;
    struct red_black_element* left;
    // further attributes...
};
```

```
int get_nb_eles (const struct red_black_element* root)
{
```



Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 4 (10 points) (Pilote de périphérique)

Le processeur i.MX27 de Freescale dispose d'un contrôleur permettant de gérer un interface « keypad matrix 8x8 ». La figure ci-contre décrit sommairement les registres du contrôleur. Pour rappel, l'i.MX27 travaille en « little endian » et autorise des accès 8, 16 et 32 bits.

Table 25-4. KPP Register Summary

Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1000_8000 (KPCR)	R	KCO	KCO	KCO	KCO	KCO	KCO	KCO	KRE	KRE	KRE	KRE	KRE	KRE	KRE	KRE
	W	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1
0x1000_8002 (KPSR)	R	0	0	0	0	0	KPP	KRI	KDI	0	0	0	0	0	KPK	KPK
	W						.EN	E	E						R	D
0x1000_8004 (KDDR)	R	KCD	KCD	KCD	KCD	KCD	KCD	KCD	KRD	KRD	KRD	KRD	KRD	KRD	KRD	KRD
	W	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1
0x1000_8006 (KPDR)	R	KCD	KCD	KCD	KCD	KCD	KCD	KCD	KRD	KRD	KRD	KRD	KRD	KRD	KRD	KRD
	W	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1

- a. Définissez l'interface C (structure, constantes, ...) pour le contrôleur ci-dessus permettant l'implémentation d'un pilote de périphérique en C.

Remarque : seuls les bits du registre KPSR doivent être déclarés.

- b. Déclarez la variable permettant d'accéder au contrôleur situé à l'adresse 0x1000'8000
- c. Ecrivez le code permettant de:

- i. Initialiser le registre KPSR du contrôleur KPP.

Pour rappel: w1c signifie qu'il faut écrire un 1 pour effacer le bit (c.à.d. le remettre à 0)

- ii. Enclencher le contrôleur (mettre le bit KPP_EN à 1)

- iii. Attendre qu'une touche (key) aille été pressée (bit KPKD à 1) et lire le contenu du registre KPDR

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 5 (10 points) (Pointeurs et pointeurs de fonctions)

Afin de solutionner la partie d) de ce problème 5, vous devez tout d'abord terminer l'implémentation du programme en définissant : a) le pointeur de fonction « **oper_t** », b) la structure « **struct operation** » et finalement c) la variable « **opers** ».

a. Définissez le type « **oper_t** » (pointeur de fonction) permettant de pointer sur les 3 fonctions ci-dessous.

```
static int oper1 (int i, int j) { return i + j; }
static int oper2 (int i, int j) { return i * j; }
static int oper3 (int i, int j) { return i / j; }
```

b. Définissez la structure « **struct operation** » permettant de construire le tableau « **opers** » ci-dessous.

opers:

	name	oper	attr	chain
[0]	« oper1 »	oper1	10	&opers[2]
[1]	« oper2 »	oper2	20	&opers[0]
[2]	« oper3 »	oper3	30	&opers[4]
[3]	« oper2 »	oper2	40	0
[4]	« oper1 »	oper1	50	&opers[3]

c. Déclarez et initialisez la variable « **opers** » afin de construire le tableau ci-dessus.

d. Pour le code ci-dessous et pour chaque itération, indiquez la fonction appelée, la valeur des arguments « **i** » et « **j** », ainsi que la valeur retournée et stockée dans le tableau « **output** ».

```
static struct operation* oper_list = &opers[1];
static int output[5];
void foo () {
    struct operation* op = oper_list;
    int j = 0;
    while (op != 0) {
        output[j] = op->oper (op->attr, j);
        op = op->chain; j++;
    }
}
```

itér	oper	i	j	output
1°				
2°				
3°				

itér	oper	i	j	output
4°				
5°				