**Systèmes Embarqués 1 : Travail écrit no 2.**

**Nom :** **Prénom :** **Classe : T-2/I-2** **Date : 15.01.2015**

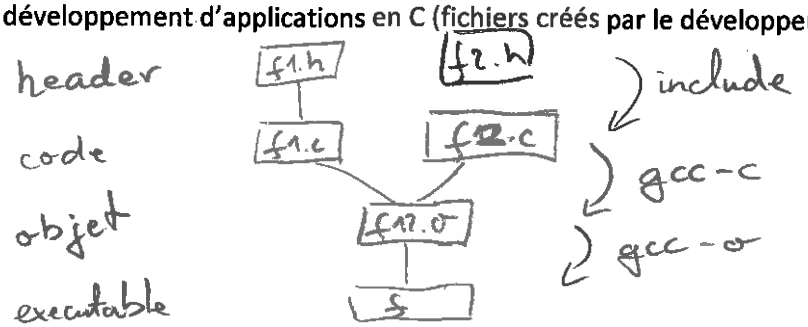
4,5

9

**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).

2




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

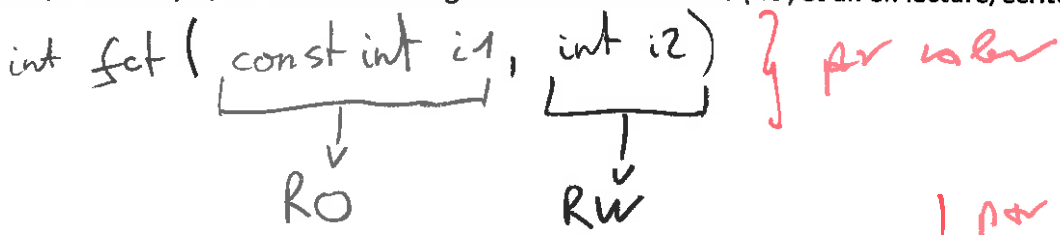
2

`fichier : file1.c`

```
extern void foo1();          // déclaration d'une fonction externe au fichier « file1.c »
                             // (globale à l'ensemble du programme)
(rien) void foo2(int){…}     // définition d'une fonction globale à l'ensemble du programme
static 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).

2




⇒ on peut faire pareil avec les pointeurs | par référence
↳ RO + pointeur non modif = const int* const

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.

2,5

`fichier : file2.c`

```
static int v1 = 0;           // variable globale pour toutes les fonctions de ce fichier,
                             // mais pas accessible à l'extérieur
(rien) int v2 = 0;           // variable globale accessible par tous les fichiers du
                             // programme et définie dans ce fichier
extern int v3;               // variable globale au programme, mais définie hors du fichier
void foo5 () {
    static int v4 = 0;       // variable rémanente et locale à la fonction
    (rien) 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***

```
extern void init();
extern int create();
extern void destroy(int);
extern void post(int, struct msg_queue_message);
extern struct msg_queue_message recv(int);
extern const int symbole = 1024;
extern int msg_sz;

struct msg_queue_message {
    char priority;
    uint32_t timestamp;
    char[msg_sz] msg;
}
char[][] priority = {"low", "medium", "high"};
```

**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)
{
    char *res = dest;
    char *crtSource = src;
    char crt = *dest++;
    while (crt != '0') {
        crt = *dest++;
    }
    crt = *crtSource;
    while( crt != 0 && n!=0){
        *dest = crt;
        *dest++;
        crt= ++*crtSource;
        n--;
    }
    *dest = '0';
    return *res;
}
```

dest src ?

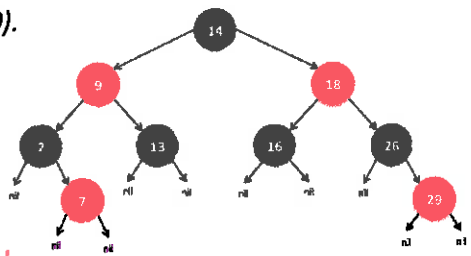
incrémenter le pointeur dest
récupérer le char de src suivant.
finir le string concat avec '0'
retourner le pointeur initial

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)
{
    int count=1;
    if (root->right!='0') {
        count+=get_nb_eles(root->right);
    }if (root->left !=0){
        count+=get_nb_eles(root->left);
    }
    return count;
}
```

if (root == 0) return 0;

## 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 7 | KCO 6 | KCO 5 | KCO 4 | KCO 3 | KCO 2 | KCO 1 | KCO 0 | KRE 7 | KRE 6 | KRE 5 | KRE 4 | KRE 3 | KRE 2 | KRE 1 | KRE 0 |
| | W | | | | | | | | | | | | | | | | |
| 0x1000_8002 (KPSR) | R | 0 | 0 | 0 | 0 | 0 | KPP_EN | KRIE | KDIE | 0 | 0 | 0 | 0 | 0 | 0 | KPKR | KPKD |
| | W | | | | | | | | | | | | | KRSS | KDSC | w1c | w1c |
| 0x1000_8004 (KDDR) | R | KCD D7 | KCD D6 | KCD D5 | KCD D4 | KCD D3 | KCD D2 | KCD D1 | KCD D0 | KRD D7 | KRD D6 | KRD D5 | KRD D4 | KRD D3 | KRD D2 | KRD D1 | KRD D0 |
| | W | | | | | | | | | | | | | | | | |
| 0x1000_8006 (KPDR) | R | KCD 7 | KCD 6 | KCD 5 | KCD 4 | KCD 3 | KCD 2 | KCD 1 | KCD 0 | KRD 7 | KRD 6 | KRD 5 | KRD 4 | KRD 3 | KRD 2 | KRD 1 | KRD 0 |
| | W | | | | | | | | | | | | | | | | |

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

```
#define KPKD 1<<0
   "    KPKR 1<<1
   "    KDSC 1<<2
   "    KRSS 1<<3
   "    KOIE 1<<8
   "    KRIE 1<<9
   "    KPP_EN 1<<10

struct KPP {
    uint16_t KPCR;
    uint16_t KPSR;
    uint16_t KDDR;
    uint16_t KPDR;
};

static volatile struct KPP * var = (struct KPP *) 0x10008000;

static uint16_t set() {
    var->KPSR &= ~KPP_EN;
    var->KPSR &= ~KRIE;
    var->KPSR &= ~KOIE;
    var->KPSR &= ~KRSS;
    var->KPSR &= ~KDSC;
    var->KPSR |= KPKR;
    var->KPSR |= KPKD;
    var->KPSR |= KPP_EN;
    while((var->KPSR & KPKD)==0);
    return var->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° | | | | |