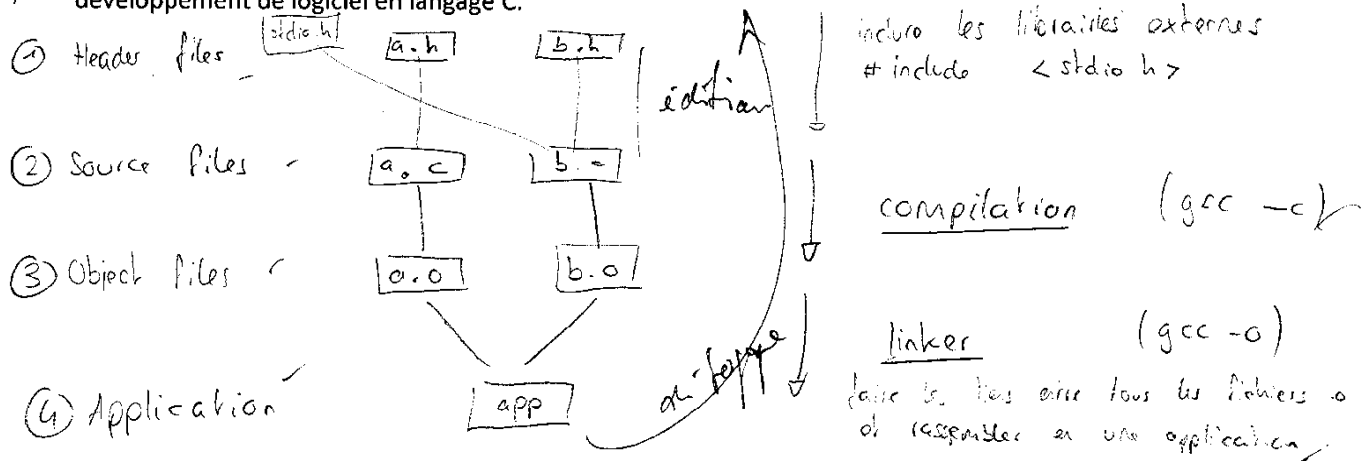


Microprocesseurs 1 & 2: Travail écrit no 2.

10

Problème n° 2 (structure et organisation des fichiers en C, scope des fonctions et variables)

3) Décrire sommairement la structure et l'organisation des fichiers en C, ainsi que le processus de développement de logiciel en langage C.



Indiquer le scope (visibilité) des fonctions suivantes :

2

fichier : file1.c

`extern void fonction1();` module de compilation + autres fichiers (fonction externe au fichier file1.c) ✓
`void fonction2 (int, int);` module de compilation + autres fichiers déclarant extern fonction2 (int, int) ✓
`static fonction3 (float);` fichier file1.c uniquement ✓

Indiquer le scope (visibilité) des variables suivantes (var1 à var6):

fichier : file2.c

`extern int var1;` tous les fichiers (variable globale externe au fichier file2.c) ✓
`int var2;` variable globale : partout file2.c + autres ✓
`static int var3;` uniquement ce fichier file2.c ✓
`static void fonction4 (int var4) {` local à la fonction.
`static int var5;` dans ce bloc (variable remarquable, état sauvegardé entre appels) ✓
`int var6;` dans ce bloc ✓

ce bloc
{ }

Microprocesseurs 1 & 2: Travail écrit no 2.

Problème n° 3 (programmation)

Programmer dans les règles de l'art la fonction de la librairie C « void *memcpy (void*, const void*, int, size_t); » afin de satisfaire à sa spécification, voir ci-dessous.

NAME

memcpy - copy bytes in memory

SYNOPSIS

#include <string.h>

void *memcpy(void * s1, const void * s2, int c, size_t n);

unsigned int

DESCRIPTION

The *memcpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied, whichever comes first. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *memcpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null pointer if *c* was not found in the first *n* bytes of *s2*.

#include <string.h>

#include <stddef.h>

typedef unsigned char;

void *memcpy (void * s1, const void * s2, int c, size_t n) {

uchar stop-char = c; ✓

uchar *ps1 = (uchar *) s1; // typecasting...

uchar *ps2 = (uchar *) s2;

uchar curr = *ps2++; ✓

while (n > 0) { ✓

*ps1++ = curr; ✓

if (curr == stop-char)

return ps1; ✓

curr = *ps2++; ✓

n--;

}

return NULL; ✓

}

Microprocesseurs 1 & 2: Travail écrit no 2.

Problème n° 4 (pointeurs et pointeurs de fonctions)

Définir la structure « struct Opers » et le type « fonction pointer » pour les 3 fonctions ci-dessous et permettant de construire la variable « oper ».

```
int oper1 (int i, int j) {return (i*3) + (j%5); }
int oper2 (int i, int j) {return (i/4) + (j<<2); }
int oper3 (int i, int j) {return (i*j) + 10; }
```

2 typedef int (*Oper_T)(int, int);

```
struct Opers {
    int index;
```

```
    Oper_T oper;
```

```
    struct Opers * next;
```

```
};
```

```
struct Opers oper[] = {
    {0, oper1, &oper[2]},
    {1, oper2, &oper[0]},
    {2, oper3, &oper[5]},
    {3, oper3, &oper[4]},
    {4, oper2, 0},
    {5, oper1, &oper[3]}
};
struct Opers* head = &oper[1];
```

Pour le code ci-dessous et pour chaque itération, indiquer le nom de la fonction appelée avec la valeur des arguments « i » et « j », ainsi que la valeur contenue dans le tableau « result ».

```
int main () {
    int result[6] = {0,1,2,3,4,5};
    struct Opers* op = head; // oper[1]
    int k = 0;
    while (op != 0) {
        int i = op->index;
        int j = result[(i+1)%6]; //
        result[k++] = op->oper (i, j);
        op = op->next;
    }
}
```

Iteration 0: i: 1 j: result[(1+1)%6] = 2	Iteration 1: i: 0 j: result[(0+1)%6] = 1	Iteration 2: i: 2 j: result[(2+1)%6] = 3	Iteration 3: i: 5 j: result[(5+1)%6] = 0	Iteration 4: i: 5 j: result[(5+1)%6] = 0	Iteration 5: i: 4 j: result[(4+1)%6] = 5
result[0]: (1/4) + (2<<2) 0 + 8 = 8	result[1]: (0*3) + (1%5) 0 + 1 = 1	result[2]: (2*3) + 10 16	result[3]: (5*3) + (0%5) 15 + 0 = 15	result[4]: 3*4 + 10 = 22	result[5]: (4/4) + (5<<2) 1 + 20 = 21

Microprocesseurs 1 & 2: Travail écrit no 2.

Problème n° 5 (device driver / spécification de l'interface)

Le processeur i.MX27 de Freescale dispose de 6 compteurs à usages multiples (General Purpose Timer / GPT1 à GPT6). La figure ci-dessous décrit sommairement les registres pour la gestion et l'opération d'un compteur.

Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x1000_3000 (TCTL1)- 0x1000_F000 (TCTL6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1000_3004 (TPRER1)- 0x1000_F004 (TPRER6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1000_3008 (TCMP1)- 0x1000_F008 (TCMP6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1000_300C (TCR1)- 0x1000_F00C (TCR6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1000_3010 (TCN1)- 0x1000_F010 (TCN6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1000_3014 (TSTAT1)- 0x1000_F014 (TSTAT6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Définir l'interface C (structure, constantes, etc.) pour le compteur ci-dessus permettant l'implémentation d'un pilote de périphérique en C. Déclarer la variable permettant d'accéder le 1^{er} compteur (GPT1) situé à l'adresse 0x1000'3000 et initialiser le prescaler à 0.

```
#include <stdint.h>
struct GPT {
    uint32_t tctl;
    uint32_t tprer;
    uint32_t tcmp;
    uint32_t tcr;
    uint32_t tcn;
    uint32_t tstat;
};
```

```
#define TEN 1 << 0
#define CLK_SOURCE 7 << 1
#define COMP_EN 1 << 4
#define CAPT_EN 1 << 5
#define CAP 3 << 6
#define FRR 1 << 8
#define OM 1 << 9
#define CC 1 << 10
#define PRESALER 0x7FF << 0
#define COMP 1 << 0
#define CAPT 1 << 1
#define WAC-1 1 << 0
#define WAC-2 1 << 1
```

volatile struct GPT * gpt1 = (struct GPT *) 0x10003000;

//init prescaler to 0

gpt1->tprer = (gpt1->tprer & ~PRESALER);

Indiquez la technique utilisée ainsi que le mode d'accès pour les 4 arguments de la fonction ci-dessous.

```
void fnct (int a1, const int a2, int* a3, int* const a4);
```

a1 paramètre copié, pouvant être modifié ✓

a2 paramètre copié, ne pouvant pas être modifié ✓

2. a3 paramètre par référence, pouvant être modifié ✓

a4 paramètre par référence, ~~ne~~ pouvant pas être modifié

donnée RA
pointeur RO