Microprocesseurs 1 & 2: Travail écrit no 4.

5.4

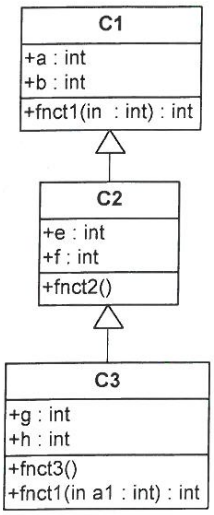
Nom : [redacted] **Prénom :** [redacted]

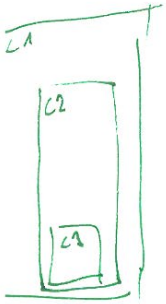
Classe : I/2
Date : 14.06.2011

6

Problème n° 1 (programmation orienté-objet)

1. Déclarez complètement les structures C pour le diagramme de classes ci-dessous. La classe C3 surcharge la fonction «`fnct1`» de la classe C1.
2. Implémentez complétement la méthode d'initialisation de la classe C3 «`extern struct C1* C3_init (struct C3 *obj);`».
3. Implémentez la fonction «`fnct1`» de la classe C3 de manière à ce qu'elle retourne la somme de « `a1 + C3::g + C2::f + C1::a` »






```
typedef int (* fnct1_t) (c1_class_t * oref, int);
typedef (* fnct2_t) void (c2_class_t * oref)
typedef (* fnct3_t) void (c3_class_t * oref);

struct c1_class_t {
    fnct1_t * fnct1;
    int a;
    int b;
};

struct c2_class_t {
    c1_class_t * base_m;
    fnct2_t * fnct2;
    int e;
    int f;
};

struct c3_class_t {
    c2_class_t * base_m;
    fnct3_t * fnct3;
    int g;
    int h;
};

int function1_c3 (struct c1_class_t oref, int a1) {
    return a1 + (oref -> g) +
           (oref -> base_m -> f) +
           (oref -> base_m -> base_m -> a);
}

struct c3_oref = container_of(c1_oref, struct c3, base_c2.base_c1)
```

```
struct c1_class_t * c3_init (struct c3_class_t *obj) {
    c2_init (obj -> base_m);
    obj -> h = 0; obj -> g = 0;
    obj -> fnct3 = function3;
    obj -> base_m.base_m.fnct1 = function1_c3
    return &(obj -> base_m -> base_m);
}
```

obj -> base_m.base_m.fnct1 = fnct1_3;

Microprocesseurs 1 & 2: Travail écrit no 4.

8

Problème n° 2 (Toolchain)

Concevez un Makefile pour la génération d'une application composée de 3 fichiers (file1.c, file2.c et file3.c). Cette application devra pouvoir être générée aussi bien pour la machine hôte que pour une cible équipée d'un processeur ARM et fonctionnant sous Linux. Le choix de la cible se fera à l'aide de la variable «`TARGET=host`» ou «`TARGET=apf`». Sur la machine hôte le compilateur «`gcc`» sera utilisé, tandis que pour la cible le compilateur «`arm-linux-gcc`» devra être employé. Le programme exécutable sera appelé «`app_host`» pour la machine hôte et «`app_apf`» pour la cible. Les flags de compilation sont «`-g -W -Wall -Wextra -O2 -MD`» (pour rappel -MD génère les dépendances). Les objets ainsi que les fichiers de dépendance devront être placés dans les répertoires «`./.obj/host`» respectivement «`./.obj/apf`». Remarquez que la fonction «`addprefix`» permet d'ajouter un préfix aux noms d'une liste. Le Makefile devra également permettre d'effacer les fichiers générés pour une cible donnée. Il est impératif d'utiliser des variables pour spécifier les flags de compilation et les fichiers sources.

Makefile :

```
ifeq ($(TARGET), "host")
    EXEC = app-host
    CC   = gcc
    DEPF = ./.obj/host
else
    EXEC = app_apf
    DEPF = ./.obj/apf
    CC   = arm-linux-gcc
endif

CFLAGS = "-g -W -Wall -Wextra -O2 -MD" -c
SRCS = file1.c file2.c file3.c
OBJS = (addprefix ($(SRCS):.c=.o), $(DEPF)/)
DEP  = (addprefix ($(SRCS):.c=.d), $(DEPF)/)

all: $(EXEC)

$(DEPF)/%.o: %.c
    $(CC) $(CFLAGS) $< -o $(DEPF)/$@

$(EXEC): $(OBJS)
    $(CC) $(CFLAGS) -d $(DEP) -o $(EXEC)

clean:
    rm -rf $(DEPF)
    rm -f $(EXEC)
.PHONY: clean all
```

- include ...
$(DEPF)/: mkdir -p —

— dossier des dépendances

Microprocesseurs 1 & 2: Travail écrit no 4.

7.1

Problème n° 3 (Vérification)

Implémentez, pour la fonction ci-dessous, un test unitaire permettant de valider/vérifier son bon fonctionnement. Veuillez noter que l'implémentation de la fonction « create_obj » n'est pas forcément complète. Le test unitaire devrait permettre de découvrir les lacunes de son implémentation.
Remarque : n'implémentez que le 1er test par cas et indiquez simplement la valeur des arguments de test et le résultat pour les autres.

```
/**
 * @param n number of objects (struct S) to be created (valid range: 1..512)
 * @param s size of the object to be created in word (32-bit)
 * @param i initial value
 * @param o address to store the created objects
 * @return int execution status, >0:object size, -1:out-of-range, -2:invalid pointer
 */
int create_obj (int n, int s, int i, void* *o);
{
    if (o == 0) return -2;
    *o = 0;
    int size = s * n * 4;
    if ((n <= 0) || (n > 512) || (s <= 0)) return -1;
    int* p = malloc (size);
    if ((p == 0) return -2;
    for (int j=(s*n)-1; j>=0; j--) p[j] = i;
    *o = p;
    return size;
}
```

n et s grand

2^31 = MAX_INTEGER_VALUE

```
static int MAX_SIZE = 1000;
test_create_obj () {
    void *o;
    // param n not valid
    CU_ASSERT ( create_obj (0, 10, 20; &o), -1); // valeur 0
    // à tester : n = -10 -> -1 valeur négative
                  n = 513 -> -1 valeur trop grande

    // param s not valid
    CU_ASSERT (create_obj ( 10, -10, 20, &o), -1); // valeur négative
    CU_ASSERT (create_obj ( 10, 0, 20, &o), -1; // s = 0
    // valid params, "brute force"
    for ( int n = 1 ; n <= 512, n++){
        for ( int s = 1 ; s <= MAX_SIZE; s++) {
            CU_ASSERT ( create_obj ( n, s, 100, &o), >0);
            // object created ok

    // test mettant en évidence une lacune de codage
    // quand s * n * 4 > INTEGER_MAX_VALUE -> overflow -> size = 0
    CU_ASSERT ( create_obj ( 512, pow (2,30) (2^30), 100, &o), -1);
    // dépend de la réaction de malloc (negative_value);
```

// test mettant en évidence une lacune de codage -> surcharger malloc ?

Microprocesseurs 1 & 2: Travail écrit no 4.

9

Problème n° 4 (Documentation)

1. Indiquez 4 outils permettant de documenter du logiciel.

documentation de versions : SVN, CVS, --
documentation de Bugs : ~~tes~~ tracker
documentation de code : doxygen
documentation "autre" : fichiers textes ASCII à côté des sources

2

2. Citez 2 aspects importants permettant de maintenir et documenter correctement du logiciel

1 → définir des règles de documentation au départ du projet
→ respecter tout au long du développement ces règles
→ documenter au fur et à mesure

3. Citez les 3 niveaux principaux de la documentation du logiciel et indiquez les documents importants qui en découlent.

- Documentation utilisateurs :
  à destination de l'utilisateur final du produit
  peut comprendre : - manuel d'utilisation
  - limitations du programme

6

- Documentation de développement
  à destination des développeurs, en interne dans l'entreprise
  peut comprendre : - documentation architecture
  - documentation design
  - documentation code

- Documentation des ~~re~~ révisions / versions
  à destination interne / externe
  peut comprendre : - liste des améliorations par version
  - bugs corrigés / bugs restant
  - changement suit aux versions successives

Microprocesseurs 1 & 2: Travail écrit no 4.

10

Problème n° 5 (DMA)

1. Décrivez succinctement le principe d'un DMA

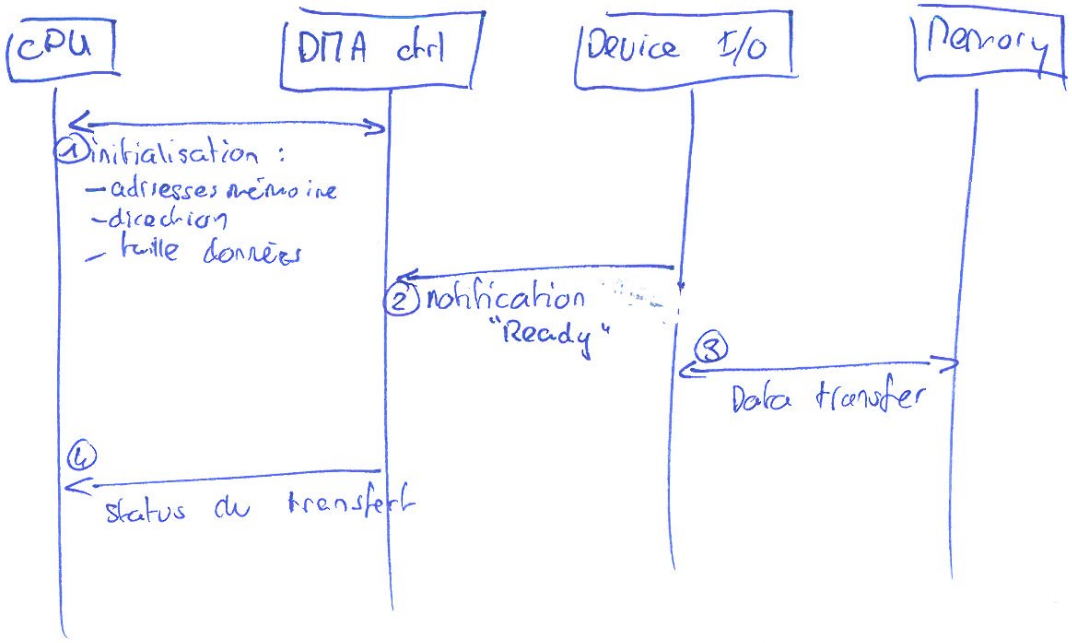
Direct Memory Access

→ décharger le processeur en passant par un contrôleur dédié (DMA) pour les transferts mémoires importants entre la mémoire et les périphériques.

→ indispensable pour les périphériques rapides

2

2. Indiquez à l'aide d'un graphique les 4 phases principales d'un transfert DMA



5

3. Citez 3 exemples d'applications avec DMA

→ interfaces réseau (ex Ethernet, Giga

→ I/O disque dur (SATA, IDE, ...)

→ UART (série)

3

→ échantillonnage / récupération de données à intervalle régulier