Hés·so Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

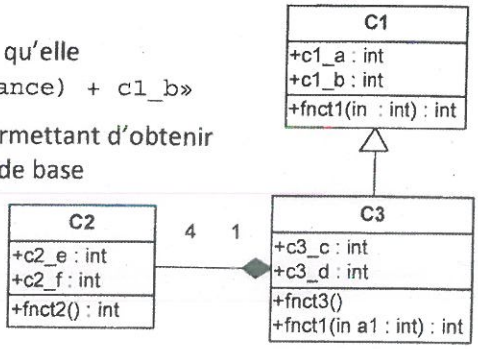
Microprocesseurs 1 & 2: Travail écrit no 4. 5.0

Nom : Carnevale **Prénom :** Antonio **Classe : T/2** **Date : 17.06.2011**

5

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 la fonction «`fnct1`» de la classe C3 de manière à ce qu'elle retourne la somme de «`a1 + c3_d + c2_e(de la 2ème instance) + c1_b`»
3. Implémentez les macros «`offset_of`» et «`container_of`» permettant d'obtenir la référence sur l'objet dérivé à partir de la référence sur la classe de base



c1.h

```
struct class_c1_t {
    int c1_a;
    int c1_b;
    int (* fnct1)(struct class_c1_t* c1_t, int in);
};
extern struct class_c1_t* c1_init(struct class_c1_t* c1_t);
```

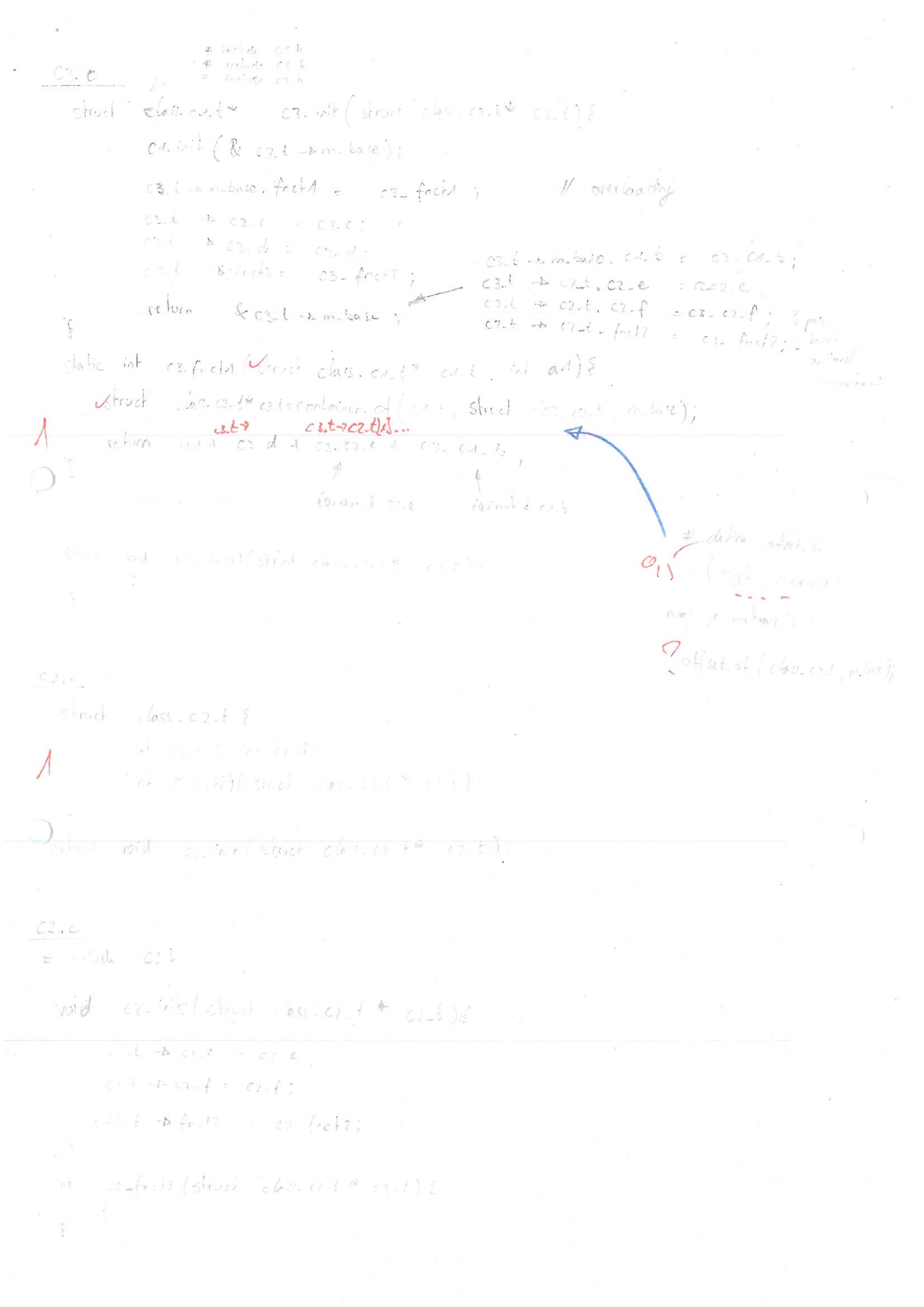
c1.c

```
#include c1.h
struct class_c1_t* c1_init(struct class_c1_t* c1_t){
    c1_t -> c1_a = c1_a;
    c1_t -> c1_b = c1_b;
    c1_t -> fnct1 = c1_fnct1;
    return c1_t
}
static int c1_fnct1(struct class_c1_t*, int in) {
    :
}
```

c2.h

```
#include c1.h
#include c2.h
struct class_c3_t {
    struct class_c1_t m_base;
    int c3_c; int c3_d;
    void (* fnct3)(struct class_c3_t* c3_t);
};
extern struct class_c1_t c3_init(struct class_c3_t * c3_t);
```

1.5 — struct class_c2_t c2_t [4]



Microprocesseurs 1 & 2: Travail écrit no 4.

Problème n° 2 (Toolchain)

1. Concevez un Makefile pour la génération d'une application composée de 3 fichiers (main.c, file_a.c et file_b.c). Le programme exécutable sera appelé «appl». Le compilateur «gcc» sera utilisé pour la génération de l'application avec les flags de compilation «-W -Wall -Wextra -O2 –std=c99».
   Le Makefile devra également permettre d'effacer tous les fichiers générés. Il est impératif d'utiliser des variables pour spécifier les flags de compilation et les fichiers sources.

   Makefile :

```
EXEC = appl
SRCS += main.c file_a.c file_b.c
CC = gcc
CFLAGS = -W -Wall -Wextra -O2 -std=c99
OBJS = $(SRCS:.c=.o)

%.o :
	$(CC) $(CFLAGS) $(OBJS) $< -o $@

all : $(EXEC)
	$(EXEC) : $(OBJS)
	$(CC) $(OBJS) $(LDFLAGS) -o $@

clean :
	rm -rf $(EXEC) $(OBJS)

.PHONY : all clean
```

2. Quel est la fonction des utilitaires suivants :

   a. gdb

   l'utilitaire "gdb" va nous permettre de tracer le débogage. (Autre comme gcc qui indique le compilateur à utiliser)

   b. gprof

   l'utilitaire "gprof" va nous permettre d'analyser le profil de notre application. (profilage des données).

   c. as

   l'utilitaire "as" va nous permettre d'assembler le code en machine

   d. ld

   l'utilitaire "ld" va nous permettre de linker les fichiers objet

Microprocesseurs 1 & 2: Travail écrit no 4.

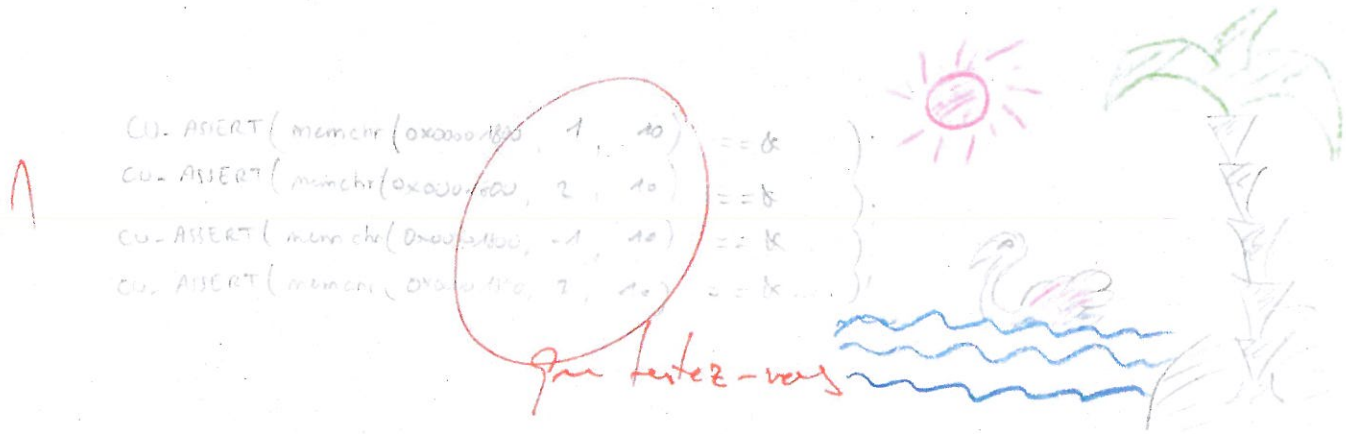
Problème n° 3 (Vérification)

1. Citez 3 techniques/méthodes permettant de valider des applications logicielles dans les différentes phases de leur développement

2. Décrivez une technique/méthode permettant de garantir qu'un composant logiciel a été correctement et si possible complètement vérifier. Citez un utilitaire de la chaîne d'outils GNU permettant de mettre d'utiliser cette méthode/technique ainsi que la façon de le mettre en œuvre.

3. Implémentez, pour la fonction « memchr » de la librairie standard C (selon description ci-dessous) , un test unitaire permettant de valider/vérifier deux résultats positifs et deux résultats négatifs.

```
/**
 * Locate character in block of memory
 * Searches within the first num bytes of the block of memory pointed by ptr for the
 * first occurrence of value (interpreted as an unsigned char), and returns a pointer
 * to it.
 * @param ptr pointer to the block of memory where the search is performed.
 * @param value value to be located. The value is passed as an int, but the function
 *              performs a byte per byte search using the unsigned char conversion of
 *              this value.
 * @param num number of bytes to be analyzed.
 *
 * @return A pointer to the first occurrence of value in the block of memory pointed
 *         by ptr. If the value is not found, the function returns NULL.
 */
void * memchr (const void * ptr, int value, size_t num );
```



Microprocesseurs 1 & 2: Travail écrit no 4.

10

Problème n° 4 (Documentation)

1. Indiquez 4 outils permettant de documenter du logiciel.

2

- description (doxygen, asciidoc, doxia, ...)
- Specifications (contraintes, bugs, ...)
- code (commentaire, source code management, ...)
- Artefact Wiki (manuel de référence, ...)

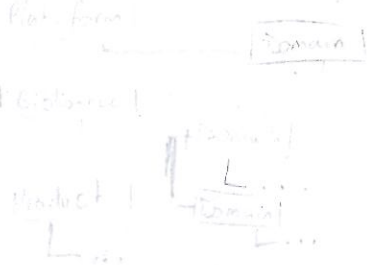
2. Citez 2 outils permettant de simplifier le développement de logiciels et d'améliorer sa qualité

2

- Automate Build Tools (makefile, ...)
- source Code Management (SVN, ...)

3. Indiquez une manière de structurer le logiciel et sa documentation afin de simplifier son développement et sa maintenance

2



4. Décrivez succinctement comment on peut procéder pour révisionner du logiciel

+SCM

On peut utiliser les Tag/Label :

→ Label de module : + grand granularité

1.1

→ Label de composants : + simple à implémenter

5. Citez les 3 niveaux principaux de la documentation du logiciel (*public cible*)

2

- Documentation de l'utilisateur
- Documentation du vérificateur
- Documentation du développeur

Microprocesseurs 1 & 2: Travail écrit no 4.

Problème n° 5 (DMA)

1. Décrivez succinctement le principe d'un DMA.

Le contrôleur DMA doit avoir accès aux signaux du bus (données, adresses et données). Ceci afin de décharger le CPU de ses tâches (c'est le DMA qui va "initier les transactions" sur ces bus).

2. Décrivez à l'aide d'une figure l'architecture DMA

Le fait du contrôleur DMA est de contrôler le transfert de données entre I/O et Memory

3. Citez les 4 phases principales d'un transfert DMA entre une mémoire et un périphérique

① Initialisation du DMA: le CPU initialise le DMA (adresse mémoire, taille des données à transférer, ...)

② Transfert: transfert des données entre la mémoire, ...

④ Notification de status: le CPU s'informe du status à l'aide du registre status du DMA

4. Citez 2 modes d'opérations des DMA du processeur i.MX27

Il y a 3 modes d'opérations, voici 2 de ceux-ci:

Linear memory: transfert de données entre la mémoire linéaire et un périphérique (autre)

FIFO: transfert de données entre la mémoire FIFO (UART,...) et un périphérique (autre)

5. Citez 2 exemples d'application avec DMA

- Echantillonnage de données à intervalle réguliers
- Contrôleurs Ethernet
- Contrôleurs bus PCI
- Contrôleurs disques durs (IDE, SATA, ...)