# Examen SoftwareOntwikkeling I
# 2e Bachelor Informatica
# Faculteit Wetenschappen
# Academiejaar 2009-2010
# 22 januari, 2010

**BELANGRIJK**

1. Schrijf je naam onderaan op elk blad.
2. Vragen 1 ,2 en 3 gaan enkel over C++ (GEEN C)
3. Vraag 4 gaat over deel III van de cursus (gesloten boek)
4. Kladpapier vind je achteraan.
5. Gebruik geen potlood noch rode balpen!
6. Veel succes!

## Deel I. Open Boek gedeelte op papier (C++) + Gesloten Boek (cursus deel III) (8u30-9u45u)

### Programmeren in C++

[op 8 punten van de 20]

**Vraag 1: HashTable** [op 2 punten van de 20]
Gegeven de header file HashTable.h van een hashtabel die **pointers naar strings** bijhoudt (in bijlage).

Bij initialisatie wordt in deze hashtabel reeds een hoeveelheid geheugen gealloceerd voor het opslaan van `capacity` elementen van het type `HT_Entry*`. Er wordt bij de default constructor gebruik gemaakt van de aanwezige constante `DEFAULT_CAPACITY`. Alle verdere attributen worden op 0 geïnitialiseerd.

a) Vorm deze beide klassen om tot een template klassen, wetende dat:
   - Enkel de waarde geassocieerd met een sleutel generiek moet zijn, maw. de sleutel zal steeds een string zijn.
   - De generieke implementatie nog steeds **pointers naar objecten** moet bijhouden.

   Duid de nodige aanpassingen **duidelijk** aan in de header file in bijlage. [1 punt]

b) Geef tevens de implementatie van de default constructor van de **template** klasse `HashTable` zoals die in de implementatie zou moeten komen. [0.5 punt]

c) Men wenst nu een operator te definiëren voor het optellen van de template hashtabellen, m.a.w. het resultaat van deze operatie is een template hashtabel met alle elementen van de opgetelde template hashtabellen. Voeg de signatuur van deze operator toe aan de bijgevoegde header file. [0.5 punt]

**Vraag 2: MediaItem** [op 4 punten van de 20]

a) MediaItem [1 punt]

Een bibliotheek wenst zijn informaticasysteem te herorganiseren en vertrekt uit volgende basisklasse, die de gemeenschappelijke kenmerken bijhoudt van de boeken, magazines, cd's en dvd's van de bibliotheek.

```
#ifndef MEDIAITEM_H
#define MEDIAITEM_H
#include <string>

class MediaItem {
private:
      long id;
      std::string title;
      std::string author;
public:
      MediaItem();
      MediaItem(long id, std::string title, std::string author);
      ~MediaItem();
      long getId() const;
      std::string getTitle() const;
      std::string getAuthor() const;
      void setId(long id);
      void setTitle(const std::string title);
      void setAuthor(const std::string author);
      std::string toString() const;
};

#endif
```

Geef de aanpassingen nodig aan deze header file, wetende dat de klasse `MediaItem` een **abstracte** klasse moet zijn en de `toString` methode slechts een zinnige implementatie kan krijgen in een afgeleide klasse.

Duid de nodige aanpassingen **duidelijk** aan in bovenstaande header file.

b) Book [2 punten]

Een boek is een voorbeeld van een afgeleide klasse van `MediaItem`. Deze klasse houdt de uitgever, het aantal pagina's en een array met de namen van de hoofdstukken bij als extra attributen.

```
#ifndef H_BOOK
#define H_BOOK
#include "MediaItem.h"

class Book: public MediaItem{
private:
	std::string* chapters;
	std::string editor;
	int nrOfPages;
public:
	Book(long id, std::string title, std::string author, int
nrOfPages, std::string editor, std::list<std::string> chapters);
	Book();
	~Book();
	void setNrOfPages(int nrOfPages);
	void setEditor(const std::string editor);
	std::string getEditor() const;
	int getNrOfPages() const;
	std::string* getChapters() const;
	std::string toString() const;
};

#endif
```

Geef de implementatie van de **constructor met argumenten** van `Book` zoals die in de cpp file zou moeten komen wetende dat **diepe kopieen** genomen worden van de `string`s in de chapters `list`. [op 1.5 punten van de 20]

Geef de implementatie van de destructor van `Book` zoals die in de cpp file zou moeten komen. [op 0.5 punt van de 20]

c) Operatoren [1 punt van de 20]

We wensen een nu outputstream operator te definiëren om objecten van het type `MediaItem` uit onze overervingsstructuur te kunnen uitschrijven. Deze operator schrijft het resultaat van de `toString()` methode uit. Geef de signatuur en implementatie van deze operator (**`operator<<`**).

Signatuur zoals die in MediaItem.h zou moeten komen: [op 0.5 punt van de 20]

Implementatie zoals die in MediaItem.cpp zou moeten komen: [op 0.5 punt van de 20]

**Vraag 3: BibliotheekDatabank** [op 1 punt van de 20]

Naast `Book` zijn ook de klassen `CD`, `DVD` en `Magazine` afgeleide klassen van `MediaItem`. Men wenst nu instanties van deze klassen bij te houden in de template hashtabel uit vraag 1, je mag ervan uit gaan dat de code hiervoor reeds beschikbaar (gecompileerd) is. Schrijf de code om in de main-functie (main.cpp) van een programma:

a) een `HashTable` van polymorfe objecten (`Book`, `DVD`, `CD` en `Magazine`) aan te maken [op 0.25 punt van de 20]

b) 1 object aan te maken van de klasse `Book` en 1 object van de klasse `CD` en deze in de tabel op te slaan (gebruik default constructoren, en de titel als sleutel). [op 0.5 punt van de 20]

c) het ingenomen geheugen terug vrij te geven [op 0.25 punt van de 20]

**Vraag 4: Begrippen** [op 1 punt van de 20]

a) Waarom worden Makefiles dikwijls in combinatie met CVS systemen gebruikt? [op 0.5 punt van de 20]

b) Hoe wordt de throughput van een applicatie bepaald? [op 0.5 punt van de 20]

KLADPAPIER:

## Bijlage: HashTable.h

```
#ifndef H_HASHTABLE
#define H_HASHTABLE
#include <string>
#define DEFAULT_CAPACITY 32
#define MAX_LOAD_FACTOR 0.75

class HT_Entry{
private:
	std::string key;
	std::string* value;
public:
	HT_Entry(std::string& k, std::string* v): key(k), value(v) {};
	~HT_Entry(){};
	std::string getKey(){return key;};
	std::string* getValue(){return value;};
};

class HashTable{
public:
	void put(std::string& key, std::string* value);
	std::string* get(const std::string& key);
	std::string* remove(const std::string& key);
	void printTable();
	HashTable();
	~HashTable();


private:
	unsigned int capacity;
	unsigned int numberOfEntries;
	float loadFactor;
	HT_Entry** table;
	unsigned int hash(const std::string& key);
	void rehash();


};



#endif
```

**Examen SoftwareOntwikkeling I**
**2e Bachelor Informatica**
**Faculteit Wetenschappen**
**Academiejaar 2009-2010**
**22 januari, 2010**

**BELANGRIJK**
1. Schrijf je naam onderaan op elk blad.
2. Deze vraag gaat enkel over C (GEEN C++)
   Stel dus je compiler hierop in!
3. Je kan de achterzijde van deze opgave als kladpapier gebruiken.
4. Veel succes!

# Deel II. Open Boek gedeelte op pc (10u – 12u30)

**Programmeren in C: Beheer van GSM contacten en oproepen**
[op 12 punten van de 20]

Men wenst code te ontwikkelen voor het beheer van contactpersonen op een GSM en het bijhouden van recente uitgaande oproepen.

Hiervoor gebruikt men volgende datastructuur:

```
enum number_type {MOBILE = 0, PRIVATE = 1, WORK = 2};

typedef struct list_element{
    enum number_type type;
    char* nr;
    struct list_element* next;
    struct list_element* prev;
} list_element;

typedef struct contact_record{
    char* first_name;
    char* last_name;
    list_element* nr_list;
} contact_record;

typedef struct lookup_entry{
    char* number;
    contact_record* contact;
} lookup_entry;

typedef struct gsm_db{
  contact_record*  contact_table;
  int              contact_table_size;
  lookup_entry*    lookup_table;
  int              lookup_table_size;
  contact_record** recent_outgoing_calls;
  int              max_number_recent_calls;
} gsm_db;
```

De hoofddatastructuur is `gsm_db`. Deze houdt een array van `contact_record`s (`contact_table`), een tabel van `lookup_entry`'s (`lookup_table`) en een array van pointers naar `contact_record`s (`recent_outgoing_calls`) bij. Bij elke tabel hoort tevens een integer die aangeeft hoe groot de corresponderende arrays zijn.

Structs van het type `contact_record` houden op hun beurt de volgende gegevens bij: de voornaam van het contact, de familienaam van het contact en een dubbelgelinkte lijst (bestaande uit `list_element`s) die de (eventueel meerdere) nummers van dat contact en hun type (aangegeven door de `enum numbertype`) opslaat.
De `contact_table` array houdt simpelweg alle contactpersonen bij. De grootte van deze array dient **mee te groeien** met het aantal aanwezige elementen. Dit betekent dat er nooit meer plaats is gealloceerd dan nodig voor de aanwezige `contact_record`s.
De contacten en hun nummers zijn **niet** gesorteerd.

De `lookup_table` laat toe om snel voor een bepaald nummer het bijhorende `contact_record` op te zoeken. Deze array is (alfanumeriek) **gesorteerd** op nummer.
De grootte van deze array dient **mee te groeien** met het aantal aanwezige elementen. Dit betekent dat er nooit meer plaats is gealloceerd dan nodig voor de aanwezige `lookup_entry`s.

De tabel `recent_outgoing_calls` houdt een array van pointers naar de contacten waarnaar recent is gebeld bij, **gesorteerd** met de **recenste eerst**.
Indien meerdere nummers van het zelfde contact zijn gebeld, is er hiervoor slechts **één** element in de array.
Het maximale aantal elementen van deze array wordt bij creatie bepaald (bvb. 10) en wordt opgeslagen in `max_number_recent_calls`.

Gegeven het header bestand gsm.h, het aan te vullen implementatieskelet gsm.c en het testprogramma main.c. In het skelet zijn al een aantal functies aanwezig.

Gevraagd is volgende functies te implementeren in gsm.c:

- `gsm_db* create_gsm_db(int max_number_recent_calls)` [1.5 punten]
  Initialiseert de datastructuur en geeft een gebruiksklare, lege `gsm_db` terug. Indien niet genoeg geheugen beschikbaar is wordt `NULL` teruggegeven.

- `void destroy_gsm_db(gsm_db** gsmdb)` [2.5 punten]
  Geeft al het geheugen ingenomen door de opgegeven `gsm_db` terug vrij.

- `int add_contact(gsm_db* gsmdb, char* first_name, char* last_name)` [2 punten]
  Voegt een contact met opgegeven voor- en familienaam toe aan de opgegeven `gsm_db`. Geeft 1 van de volgende statuscodes terug:
  - 0 : nieuw contact succesvol toegevoegd
  - -1 : contact bestaat al
  - -2 : er is onvoldoende geheugen beschikbaar

- `int add_number(gsm_db* gsmdb, char* first_name, char* last_name, char* nr, enum number_type type)` [3 punten]
  Voegt een nummer met opgegeven type toe voor het contact met opgegeven voor- en familienaam aan de opgegeven `gsm_db`. Geeft 1 van de volgende statuscodes terug:
  - 0 : nieuw nummer succesvol toegevoegd
  - -1 : contact bestaat niet
  - -2 : nummer bestaat reeds bij dit contact
  - -3 : er is onvoldoende geheugen beschikbaar

- `int new_outgoing_call(gsm_db* gsmdb, char* nr)` [2 punten]
  Registeert een nieuwe uitgaande oproep naar het opgegeven nummer in de opgegeven `gsm_db`. Geeft van volgende statuscodes terug:
  - 0 : alles succesvol verlopen
  - -1: opgegeven `nr` is ongekend in de gegeven `gsm_db`

- `void print_contacts(gsm_db* gsmdb, int choice)` [1 punt]
  Print de contacten uit afhankelijk van de gegeven keuze:
  - 0 : alle contacten opgeslagen in de `contact_table`
  - 1 : recent gebelde contacten, gesorteerd van meest naar minst recent

  Gebruik volgend formaat:

```
first_name1 last_name1
nr1 nr2 nr3

first_name2 last_name2
nr1

…
```

Enkele hints/opmerkingen:

- Schrijf je naam, stamnummer en richting bovenaan het gsm.c bestand.
- Je mag het gegeven header bestand NIET aanpassen! Doe je dit toch, dan verlies je punten.
- Respecteer de return waarden bij `add_contact`, `add_number` en `new_outgoing_call`.
- Probeer waar mogelijk bestaande methoden te hergebruiken.
- De methoden `find_contact` en `print_lookup_table` zijn gegeven.
- Er worden van `string`s enkel diepe kopiëen genomen.
- De gegeven main methode bevat reeds code om alle te implementeren functies te testen. Je mag deze methode uiteraard ook aanpassen indien gewenst, maar dit is strikt gezien niet nodig.
- Compiler warnings over het gebruik van deprecated functies uit de library `string` mag je negeren.

Veel succes!

Appendix (voor de volledigheid):

Project aanmaken in MS Visual C++ EE:

- Kies **File – New – Project** en selecteer als Project Type **Win32**.
- Als Template selecteer je **Win32 Console Application**.
- Onderaan vul je dan een projectnaam, een opslaglocatie en de naam van de (nieuwe) Solution in.
- Na bevestigen wordt een Win32 Application wizard opgestart. Druk op Next, check of Console Application als Application type wordt weergegeven en vink **Empty Project** aan. Vervolgens kan je deze wizard beëindigen. Er is nu een nieuw project en een nieuwe Solution aangemaakt.
- De header files toevoegen kan je door rechts te klikken op de '**Header files**' folder en **Add – Existing Item** aan te klikken

Compiler instellen op C code:

**Properties – Configuration Properties – C/C++ - Advanced – Compile As…** => compile as C