Opdrachten kwartaal 4 week 2 – User-defined Data Type in C++

In deze opdracht maak je kennis met de belangrijkste taalconstructies die C++ biedt voor het programmeren van UDT's (User-defined Data Types). Een UDT is een user-defined type dat door de gebruiker, de programmeur die dit type gebruikt, niet te onderscheiden is van ingebouwde types (zoals `int`).

Lees nu de inleiding en hoofdstuk 2 tot en met paragraaf 2.10 van het dictaat Objectgeoriënteerd Programmeren in C++.

In deze opdracht ga je een UDT genaamd `Tijdsduur` implementeren waarin je tijdsduren uitgedrukt in uren, minuten en seconden kunt opslaan en waarmee je bewerkingen op deze tijdsduren kunt uitvoeren.

Gebruik het voorbeeld UDT `Breuk`, dat in het dictaat wordt besproken, als inspiratie.

Het belangrijkste leerdoel van deze opdracht is dat je het nut van UDT's begrijpt en dat je een eenvoudige UDT in C++ kunt implementeren.

Om dit te kunnen doen moet je een aantal taalconstructies uit C++ leren gebruiken die in C niet beschikbaar zijn, zoals `class`, references en de `this`-pointer.

Je leert deze les hoe je:

- een UDT genaamd `Tijdsduur` kunt definiëren in de vorm van een `class`;
- de *implementatie* van de class `Tijdsduur` kunt afschermen van de gebruiker door `private` datavelden en `private` memberfuncties te definiëren;
- de *interface* van de class `Tijdsduur` beschikbaar kunt maken voor de gebruiker door `public` memberfuncties te definiëren;
- een object van de class `Tijdsduur` kunt *initialiseren* (door middel van *constructors*);
- memberfuncties kunt definiëren die ook voor *read-only* objecten van de class `Tijdsduur` gebruikt kunnen worden;
- er met behulp van *operator overloading* voor kunt zorgen dat een `Tijdsduur` op dezelfde manier gebruikt kan worden als een ingebouwd datatype (b.v. `int`);
- de implementatie van het UDT `Tijdsduur` kunt *wijzigen* zonder dat de code die deze UDT gebruikt aangepast hoeft te worden.

In de inleiding van het dictaat is een C-programma gegeven waarin gebruik wordt gemaakt van **struct**-variabelen waarin een tijdsduur kan worden opgeslagen. In dit programma zijn functies gegeven waarmee je deze variabelen kunt bewerken. In paragraaf 2.2 van het dictaat wordt uitgelegd dat deze manier van werken programma's oplevert die niet goed onderhoudbaar, uitbreidbaar en herbruikbaar zijn. Je kunt een tijdsduur in C++ beter dan UDT definiëren.

UDT `Tijdsduur`

We willen nu dus een UDT (User-defined Data Type), ook wel zelfgedefinieerd datatype genoemd, maken waarin een tijdsduur in uren, minuten en seconden kan worden opgeslagen. We noemen dit zelf gedefinieerde datatype `Tijdsduur`.

Het UDT `Tijdsduur` kan in C++ gedeclareerd worden zoals (gedeeltelijk) gegeven in `tijd1.h`:

```
#ifndef _HR_BroJZ_Tijd1_
#define _HR_BroJZ_Tijd1_

#include <iostream>

// De declaratie van de UDT Tijdsduur:
class Tijdsduur {
public:
    // ...
    void print(std::ostream& out) const;
    // ...
private:
    void normaliseer();
    int uur;
    int min;
    int sec;
};

#endif
```

Een deel van de implementatie van het UDT `Tijdsduur` kan in C++ gedefinieerd worden zoals gegeven in `tijd1.cpp`:

```
#include <iostream>
#include <iomanip>
#include "tijd1.h"
```

```
using namespace std;

// De definities van de memberfunctie van de ADT Tijdsduur, ↩
 ↪ oftewel: de implementatie van de ADT Tijdsduur:

// ...

void Tijdsduur::print(ostream& out) const {
    if (uur == 0) {
        if (min == 0) {
            out << sec;
        }
        else {
            out << min << ':' << setw(2) << setfill('0') << sec;
        }
    }
    else {
        out << uur<< ':' << setw(2) << setfill('0') << min << ':' ↩
 ↪ << setw(2) << setfill('0') << sec;
    }
}

// ...

void Tijdsduur::normaliseer() {
    min += sec / 60;
    sec = sec % 60;
    if (sec < 0) {
        sec += 60;
        --min;
    }
    uur += min / 60;
    min = min % 60;
    if (min < 0) {
        min += 60;
        --uur;
    }
}
```

In run_tijd1.cpp is een C++-programma gegeven waarin het UDT Tijdsduur gebruikt wordt:

```
#include <iostream>
#include <iomanip>
#include "tijd1.h"

using namespace std;

int main() {
    Tijdsduur t1 {3, 50, 10}; // t1 is 3 uur, 50 minuten en 10 ↩
 ↪ seconden
    cout << "t1 = "; t1.print(cout); cout << '\n';
    Tijdsduur t2 {3, -122}; // t2 is 3 minuten minus 122 seconden
    cout << "t2 = "; t2.print(cout); cout << '\n';
    Tijdsduur t3 {3, 122}; // t3 is 3 minuten plus 122 seconden
    cout << "t3 = "; t3.print(cout); cout << '\n';
    const Tijdsduur kwartier {15, 0}; // kwartier is 15 minuten
    cout << "kwartier = "; kwartier.print(cout); cout << '\n';
    t1.erbij(kwartier); // Tel kwartier bij t1 op
    cout << "t1 = "; t1.print(cout); cout << '\n';
    Tijdsduur t4 {t1}; // t4 is een kopie van t1
    cout << "t4 = "; t4.print(cout); cout << '\n';
    t4.eraf(kwartier); // Trek kw van t4 af
    cout << "t4 = "; t4.print(cout); cout << '\n';
    t4.maal(7); // Vermenigvuldig t4 met 7;
    cout << "t4 = "; t4.print(cout); cout << '\n';
}
```

De gewenste output van dit programma is al volgt:

```
t1 = 3:50:10
t2 = 58
t3 = 5:02
kwartier = 15:00
t1 = 4:05:10
t4 = 4:05:10
t4 = 3:50:10
t4 = 26:51:10
```

In test_tijd1.cpp zijn een aantal testen gegeven waarmee het UDT Tijdsduur, m.b.v. het testfamework GoogleTest, getest kan worden. De eerste test is hieronder weergegeven:

```
#include <gtest/gtest.h>
#include <string>
#include <sstream>
#include "tijd1.h"

using namespace std;

TEST(Tijdsduur, aanmaken_zonder_argumenten)
{
    Tijdsduur t1; // t1 is 0 uur, 0 minuten en 0 seconden
    ostringstream out1;
    t1.print(out1);
    EXPECT_EQ(out1.str(), "0");
}
```

4.2.1 Start Visual Studio Code en druk op F1 om het commando venster te openen. Type "WSL" en kies voor "WSL: Connect to WSL in New Window". Open een terminal window door op ctrl + ‘ te drukken. Ga naar het directory Opdrachten door in het terminal window in te typen:

```
cd ~/Opdrachten/
```

Download het bestand tijd1.zip, pak het uit en verwijder het weer met de volgende commando's:

```
wget https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Opdrach-↩
 ↪ ten/progs/tijd1.zip
unzip tijd1.zip
rm tijd1.zip
```

Voer vervolgens onderstaand commando uit om dit directory tijd1 te openen in Visual Studio Code en CMake (automatisch) uit te voeren.

```
code -r tijd1
```

Build het project door op de Build-knop te klikken of door op F7 te drukken. Je kunt de testen uitvoeren door gebruik te maken van de TestMate-plugin. Als het goed is

faalt de eerste test[1]. Bestudeer de inhoud van de bestanden: `tijd1.h`, `tijd1.cpp`, `run_tijd1.cpp`, `test_tijd1.cpp` en `CMakeLists.txt`. Om compilatiefouten te voorkomen is nog veel code in `run_tijd1.cpp` en `test_tijd1.cpp` uitgecommentarieerd. Stel vragen aan je docent als je iets niet begrijpt.

Breid de gegeven code in `tijd1.h` en `tijd1.cpp` uit zodat alle testen in `test_tijd1.cpp` slagen en het programma `run_tijd1.cpp` de gewenste uitvoer geeft. Je mag daarbij de code van de testen in `test_tijd1.cpp` en de code in de functie `main` in `run_tijd1.cpp` niet aanpassen.

Tips:

- Maak de testen in `test_tijd1.cpp` één voor één werkend door code toe te voegen in `tijd1.h` en `tijd1.cpp`.
- Als alle testen slagen, dan kun je de commentaartekens in `run_tijd1.cpp` verwijderen zodat het programma de gewenste uitvoer geeft.

UDT `Tijdsduur` met operator overloading

Objecten van het UDT `Tijdsduur`, dat je bij opdracht 4.2.1 hebt geïmplementeerd, kunnen door de gebruiker van het UDT gebruikt worden door middel van de **`public`** memberfuncties. Zo kan de waarde van de `Tijdsduur t1` afgedrukt worden op het scherm door middel van de aanroep `t1.print(cout);`. De waarde van de `Tijdsduur t2` kan bij de `Tijdsduur t1` worden opgeteld door middel van de aanroep `t1.erbij(t2);`.

Het is voor de gebruiker van het UDT `Tijdsduur` eenvoudiger als een object van de class `Tijdsduur` op vergelijkbare wijze gebruikt kan worden als een variabele van het type **`int`**. De waarde van de `Tijdsduur t1` kan dan afgedrukt worden op het scherm door middel van `cout << t1;`. De waarde van de `Tijdsduur t2` kan dan bij de `Tijdsduur t1` worden opgeteld door middel van `t1 += t2;`. Het UDT `Tijdsduur` is hierdoor eenvoudiger te gebruiken. Je kunt dit realiseren in C++ door gebruik te maken van *operator overloading*.

Lees nu paragrafen 2.11 tot en met 2.20 van het dictaat.

Het UDT `Tijdsduur` kan in C++ met operator overloading gedeclareerd worden zoals (gedeeltelijk) gegeven in `tijd2.h`:

```
#ifndef _HR_BroJZ_Tijd2_
```

[1] Omdat het object `t1` van de class `Tijdsduur` niet geïnitialiseerd is, kan het ook zo zijn dat de test toevallig slaagt als het betreffende geheugen toevallig de waarde nul bevat.

```
#define _HR_BroJZ_Tijd2_

#include <iostream>

// De declaratie van de UDT Tijdsduur:
class Tijdsduur {
public:
    // ...
    friend std::ostream& operator<<(std::ostream& o, const ↩
 ↪ Tijdsduur& t);
private:
    void normaliseer();
    int uur;
    int min;
    int sec;
};

#endif
```

Een deel van de implementatie van het UDT `Tijdsduur` met operator overloading kan in C++ gedefinieerd worden zoals gegeven in `tijd2.cpp`:

```
#include <iostream>
#include <iomanip>
#include "tijd2.h"

using namespace std;

// De definities van de memberfunctie van de ADT Tijdsduur, ↩
 ↪ oftewel: de implementatie van de ADT Tijdsduur:

// ...

ostream& operator<<(ostream& o, const Tijdsduur& t) {
    if (t.uur == 0) {
        if (t.min == 0) {
            o << t.sec;
        }
        else {
            o << t.min << ':' << setw(2) << setfill('0') << t.sec;
        }
    }
```

```
    else {
        o << t.uur<< ':' << setw(2) << setfill('0') << t.min << ':' << setw(2) << setfill('0') << t.sec;
    }
    return o;
}

// ...

void Tijdsduur::normaliseer() {
    min += sec / 60;
    sec = sec % 60;
    if (sec < 0) {
        sec += 60;
        --min;
    }
    uur += min / 60;
    min = min % 60;
    if (min < 0) {
        min += 60;
        --uur;
    }
}
```

In run_tijd2.cpp is een C++-programma gegeven waarin het UDT `Tijdsduur` met operator overloading gebruikt wordt:

```
#include <iostream>
#include <iomanip>
#include "tijd2.h"

using namespace std;

int main() {
    Tijdsduur t1 {3, 50, 10}; // t1 is 3 uur, 50 minuten en 10 seconden
    cout << "t1 = " << t1 << '\n';
    Tijdsduur t2 {3, -122}; // t2 is 3 minuten minus 122 seconden
    cout << "t2 = " << t2 << '\n';
    Tijdsduur t3 {3, 122}; // t3 is 3 minuten plus 122 seconden
    cout << "t3 = " << t3 << '\n';
    const Tijdsduur kwartier {15, 0}; // kwartier is 15 minuten
```

```
    cout << "kwartier = " << kwartier << '\n';
    t1 += kwartier; // Tel kwartier bij t1 op
    cout << "t1 = " << t1 << '\n';
    Tijdsduur t4 {t1}; // t4 is een kopie van t1
    cout << "t4 = " << t4 << '\n';
    t4 -= kwartier; // Trek kw van t4 af
    cout << "t4 = " << t4 << '\n';
    t4 *= 7; // Vermenigvuldig t4 met 7;
    cout << "t4 = " << t4 << '\n';
    t1 += t2 += t3; // meerdere operatoren in één expressie
    cout << "t1 = " << t1 << '\n'; // tel t2 en t3 op bij t1
    cout << "t2 = " << t2 << '\n'; // tel t3 op bij t2
    cout << "t3 = " << t3 << '\n'; // t3 is ongewijzigd
    (t1 += t2) += t3; // meerdere operatoren in één expressie
    cout << "t1 = " << t1 << '\n'; // tel t2 en t3 op bij t1
    cout << "t2 = " << t2 << '\n'; // t2 is ongewijzigd
    cout << "t3 = " << t3 << '\n'; // t3 is ongewijzigd
}
```

De gewenste output van dit programma is al volgt:

```
t1 = 3:50:10
t2 = 58
t3 = 5:02
kwartier = 15:00
t1 = 4:05:10
t4 = 4:05:10
t4 = 3:50:10
t4 = 26:51:10
t1 = 4:11:10
t2 = 6:00
t3 = 5:02
t1 = 4:22:12
t2 = 6:00
t3 = 5:02
```

In `test_tijd2.cpp` zijn een aantal testen gegeven waarmee het UDT `Tijdsduur` met operator overloading, m.b.v. het testfamework GoogleTest, getest kan worden. De eerste test is hieronder weergegeven:

```
#include <gtest/gtest.h>
#include <string>
#include <sstream>
#include "tijd2.h"

using namespace std;

TEST(Tijdsduur, aanmaken_zonder_argumenten)
{
    Tijdsduur t1; // t1 is 0 uur, 0 minuten en 0 seconden
    ostringstream out1;
    out1 << t1;
    EXPECT_EQ(out1.str(), "0");
}
```

De definitie van de overloaded operator<< voor Tijdsduur is al gegeven en die kun je dus meteen gebruiken[2].

4.2.2 Ga naar het directory Opdrachten door in het terminal window in te typen:

```
cd ~/Opdrachten/
```

Download het bestand tijd2.zip, pak het uit en verwijder het weer met de volgende commando's:

```
wget https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Opdrach-↩
 ↪ ten/progs/tijd2.zip
unzip tijd2.zip
rm tijd2.zip
```

Voer vervolgens onderstaand commando uit om dit directory tijd2 te openen in Visual Studio Code en CMake (automatisch) uit te voeren.

```
code -r tijd2
```

Breid de gegeven code in tijd2.h en tijd2.cpp uit zodat alle testen in test_tijd2.cpp slagen en het programma run_tijd2.cpp de gewenste uitvoer geeft. Je mag daarbij de code van de testen in test_tijd2.cpp en de code in de functie main in run_tijd2.cpp niet aanpassen.

Tips:

[2] Als je wilt begrijpen hoe dit werkt, dan kun je paragraaf 16.7 en 16.8 uit het dictaat bestuderen.

- Maak de testen in `test_tijd2.cpp` één voor één werkend door code toe te voegen in `tijd2.h` en `tijd2.cpp`.
- Als alle testen slagen, dan kun je de commentaartekens in `run_tijd2.cpp` verwijderen zodat het programma de gewenste uitvoer geeft.

Als je het UDT `Tijdsduur` in de praktijk zou willen gebruiken, moeten er natuurlijk nog veel meer operators voor dit UDT gedefinieerd worden zodat je, bijvoorbeeld, `Tijdsduur t1` en `Tijdsduur t2` als volgt kunnen gebruiken in een **`if`**-statement: **`if`** `(t1 == t2)`.[3]

UDT `Tijdsduur` met alternatieve implementatie

In deze opdracht verlaten we het onderwerp *operator overloading* en ga je, tot slot, een andere implementatie van het UDT `Tijdsduur` implementeren.

In de implementatie die je voor opdracht 4.2.1 hebt gemaakt, werden de uren, minuten en seconden apart bijgehouden en telkens als er een bewerking op een `Tijdsduur` werd uitgevoerd, werden deze datavelden bijgewerkt en genormaliseerd. Het afdrukken van een `Tijdsduur` op het scherm was daardoor relatief eenvoudig te implementeren.

Het is echter ook mogelijk om een tijdsduur altijd om te rekenen naar seconden en alleen die seconden op te slaan. Het rekenen met tijdsduren wordt dan ook eenvoudiger omdat je niet steeds meer hoeft te normaliseren. Het afdrukken van een `Tijdsduur` wordt echter complexer omdat de opgeslagen seconden op dat moment moeten worden omgerekend naar uren, minuten en (resterende) seconden.

Het UDT `Tijdsduur` kan in C++ op bovenstaande alternatieve manier gedeclareerd, worden zoals (gedeeltelijk) gegeven in `tijd3.h`:

```
#ifndef _HR_BroJZ_Tijd3_
#define _HR_BroJZ_Tijd3_

#include <iostream>

// De declaratie van de UDT Tijdsduur:
class Tijdsduur {
public:
    // ...
    void print(std::ostream& out) const;
```

[3] Als je dit interessant vindt, dan kun je hoofdstuk 16 van het dictaat bestuderen.

```
    // ...
private:
    int sec;
};

#endif
```

4.2.3 In `tijd3.zip`[4] vind je `tijd3.h`, `tijd3.cpp`, `run_tijd3.cpp`, `test_tijd3.cpp` en `CMakeLists.txt`. Breid de gegeven code in `tijd3.h` en `tijd3.cpp` uit zodat alle testen in `test_tijd3.cpp` slagen en het programma `run_tijd3.cpp` de gewenste uitvoer geeft. Je mag daarbij de code van de testen in `test_tijd3.cpp` en de code in de functie `main` in `run_tijd3.cpp` niet aanpassen.

Je ziet dat de code waarin het UDT `Tijdsduur` wordt gebruikt (de functie `main`) in `run_tijd1.cpp` en `run_tijd3.cpp` exact gelijk is. Ook de testen in `test_tijd1.cpp` en `test_tijd3.cpp` zijn exact gelijk.

Het maakt voor de gebruiker van het UDT `Tijdsduur` dus niet uit welke implementatie gebruikt wordt. Je kunt de implementatie van het UDT wijzigen zonder dat de code die het UDT gebruikt, gewijzigd hoeft te worden.

Beide implementaties hebben hun eigen voor- en nadelen. Een applicatie waarin veel tijdsduren moeten worden opgeslagen of waarin veel met tijdsduren wordt gerekend, kan het beste de implementatie uit `tijd3.h` en `tijd3.cpp` gebruiken. Een applicatie waarin niet zoveel tijdsduren worden opgeslagen en waarin tijdsduren vaak moeten worden afgedrukt, kan het beste de implementatie uit `tijd1.h` en `tijd1.cpp` gebruiken.

[4] https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Opdrachten/progs/tijd3.zip