Opdrachten week 5 les 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.

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, gedefinieerd en gebruikt worden, zoals (gedeeltelijk) gegeven in `tijd1.cpp`[1]:

```
#include <iostream>
#include <iomanip>
using namespace std;

// De declaratie van de UDT Tijdsduur:

class Tijdsduur {
public:
    // ...
    void print() const;
    // ...
private:
    void normaliseer();
    int uur;
    int min;
    int sec;
};

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

// ...

void Tijdsduur::print() const {
    // ...
}
```

[1] In de praktijk zouden we `tijd1.cpp` opsplitsen in `Tijdsduur.h`, `Tijdsduur.cpp` en `Applicatie.cpp` om het hergebruik van het UDT te vereenvoudigen. Maar voor het gemak hebben we de declaratie van het UDT, de definitie van het UDT en de code die gebruik maakt van het UDT allemaal in één bestand gezet.

```
// ...

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;
    }
}

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

5.2.1 Breid de gegeven code in `tijd1.cpp` uit zodat het programma de gewenste uitvoer geeft. Je mag daarbij de functie `main` niet aanpassen.

Tips:

- Je kunt dit het beste stap voor stap aanpakken.
  - Maak alle regels behalve de eerste commentaar en breid het programma uit zodat het foutvrij compileert.
  - Activeer de tweede regel en breid het programma uit zodat het foutvrij compileert en de eerste regel van de gewenste uitvoer produceert.
  - Enzovoort.
- Als je een integer variabele in C++ wilt afdrukken met twee digits dan kun je dat als volgt doen:

```
int s {7};
cout << setw(2) << setfill('0') << s;
```

De uitvoer van dit codefragment is: `07`.

Objecten van het UDT `Tijdsduur`, dat je bij opdracht 5.2.1 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();`. 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, gedefinieerd en gebruikt worden, zoals (gedeeltelijk) gegeven in `tijd2.cpp`:

```
#include <iostream>
#include <iomanip>
using namespace std;

// De declaratie van de UDT Tijdsduur:

class Tijdsduur {
public:
    // ...
    friend ostream& operator<<(ostream& o, const Tijdsduur& t);
private:
    void normaliseer();
    int uur;
    int min;
    int sec;
};

// De definities van de memberfunctie van de UDT Tijdsduur, ↩
 ↪ oftewel: de implementatie van de UDT 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;
    }
}

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
```

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

5.2.2 Breid de gegeven code in `tijd2.cpp` uit zodat het programma de gewenste uitvoer geeft. Je mag daarbij de functie `main` niet aanpassen.

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]

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 5.2.1 hebt gemaakt, werden de uren, minuten en seconden apart bijgehouden en telkens als er een bewerking op een `Tijdsduur` werd

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

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

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, gedefinieerd en gebruikt worden, zoals (gedeeltelijk) gegeven in `tijd3.cpp`:

```
#include <iostream>
#include <iomanip>
using namespace std;

// De declaratie van de UDT Tijdsduur:

class Tijdsduur {
public:
    // ...
    void print() const;
    // ...
private:
    int sec;
};

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

// ...

void Tijdsduur::print() const {
    // ...
}

// ...

int main() {
    Tijdsduur t1 {3, 50, 10}; // t1 is 3 uur, 50 minuten en 10 ↩
 ↪ seconden
```

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

5.2.3 Breid de gegeven code in `tijd3.cpp` uit zodat het programma exact dezelfde uitvoer geeft als het programma dat je bij opdracht 5.2.1 hebt geïmplementeerd. Je mag daarbij de functie `main` niet aanpassen.

Je ziet dat de code waarin het UDT `Tijdsduur` wordt gebruikt (de functie `main`) in `tijd1.cpp` en `tijd3.cpp` exact gelijk is. 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.cpp` gebruiken. Een applicatie waarin niet zoveel tijdsduren worden opgeslagen en waarin tijdsduren vaak moeten worden afgedrukt, kan het beste de implementatie uit `tijd1.cpp` gebruiken.