Opdrachten kwartaal 4 week 3 – Templates

Door het gebruik van de programmeertaal C++ in plaats van C kunnen we software beter herbruikbaar, aanpasbaar en uitbreidbaar maken. Zo is het in C++ eenvoudig mogelijk om generieke functies en classes te definiëren met behulp van zogenoemde templates. Een generieke functie of class kan met verschillende types gebruikt worden.

Je leert deze les hoe je:

- de herbruikbaarheid en aanpasbaarheid van een functie kunt verbeteren door een functietemplate te gebruiken;
- de foutmelding bij het verkeerd gebruiken van een template functie kunt verbeteren door gebruik te maken van een **concept**;
- de herbruikbaarheid en aanpasbaarheid van een class kunt verbeteren door een class template te gebruiken;
- de standaard template classes `std::array<T, N>` en `std::vector<T>` kunt gebruiken.

Functietemplate

Het harmonisch gemiddelde h van een rij getallen $x_1, x_2, x_3, \ldots, x_n$ is gedefinieerd als[1]:

$$h = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{1}$$

Dit wordt toegepast voor het berekenen van het gemiddelde van verhoudingsgetallen. Als je bijvoorbeeld op de heenweg naar huis naar school gemiddeld 15 km/h fietst en op de terugweg gemiddeld 20 km/h, dan is je gemiddelde over de gehele afstand (heen en terug) gelijk aan het harmonisch gemiddelde van 15 en 20. Dat is 17,1429 km/h.

Het harmonisch gemiddelde heeft ook een toepassing in de Elektrotechniek. Als je bijvoorbeeld vier weerstanden parallel plaatst, dan kun je die ook vervangen door vier parallelle weerstanden die elk gelijk zijn aan het harmonisch gemiddelde van de oorspronkelijke weerstandswaarden.

Een beginnende C++ programmeur heeft de twee functies geschreven. Één om het harmonisch gemiddelde te bepalen van een `vector` met **int**'s en een ander om het harmonisch gemiddelde te bepalen van een `vector` met **double**'s.

[1] Zie eventueel https://nl.wikipedia.org/wiki/Harmonisch_gemiddelde.

```
double harmonisch_gemiddelde(const vector<double>& v) {
    double totaal {0};
    for (const double& e: v) {
        totaal += 1/e;
    }
    return v.size() / totaal;
}

double harmonisch_gemiddelde(const vector<int>& v) {
    double totaal {0};
    for (const int& e: v) {
        totaal += 1.0/e;
    }
    return v.size() / totaal;
}
```

4.3.1 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 `harmonisch_gemiddelde.zip`, pak het uit en verwijder het weer met de volgende commando's:

```
wget https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/↩
 ↪ Opdrachten/progs/harmonisch_gemiddelde.zip
unzip harmonisch_gemiddelde.zip
rm harmonisch_gemiddelde.zip
```

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

```
code -r harmonisch_gemiddelde
```

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 slagen alle testen. Bestudeer de inhoud van de bestanden: `harmonisch_gemiddelde.h`, `harmonisch_gemiddelde.cpp`, `run_harmonisch_gemiddelde.cpp`, `test_harmonisch_gemiddelde.cpp` en `CMakeLists.txt`. Stel vragen aan je docent als je iets niet begrijpt.

4.3.2 De programmeur heeft als parametertype van de eerste functie gekozen voor `const vector<double>&` in plaats van `vector<double>&` of `vector<double>`. Ben je het met deze keuze eens? Motiveer je antwoord.

4.3.3 Het zou natuurlijk de herbruikbaarheid en de aanpasbaarheid van de code vergroten als er slechts één functie zou zijn, waarmee zowel het harmonisch gemiddelde van een `vector` met `int`'s als van een `vector` met `double`'s berekend kan worden. Vervang beide functies door één functietemplate die dit mogelijk maakt. Bedenk dat deze functietemplate in de module interface file `harmonisch_gemiddelde.cppm` moet komen te staan. Je mag daarbij de code van de testen in `test_harmonisch_gemiddelde.cpp` en de code in de functie `main` in `run_harmonisch_gemiddelde.cpp` niet aanpassen.

4.3.4 Als je probeert om de functietemplate te gebruiken met een type waarvoor de bewerkingen die in de functie worden gebruikt niet gedefinieerd zijn, dan krijg je een foutmelding die moeilijk te begrijpen is. Bijvoorbeeld: de volgende code:

```
    vector<string> woorden {"hallo", "wereld"};
    println("Harmonisch gemiddelde van woorden: {}", ↩
 ↪ harmonisch_gemiddelde(woorden));
```

levert de volgende foutmelding op:

```
error: no match for 'operator/' (operand types are 'double' ↩
 ↪ and 'const std::__cxx11::basic_string<char>')
```

De foutmelding ontstaat in de door de compiler gegenereerde code voor de functietemplate `double harmonisch_gemiddelde(const vector<string>& v)`, maar deze code is niet zichtbaar voor de programmeur. Het zou beter zijn als de compiler een foutmelding zou geven die duidelijk maakt dat de functietemplate `harmonisch_gemiddelde` niet gebruikt kan worden met een `vector<string>`. Dit kun je (sinds C++20) bereiken door gebruik te maken van een `concept`. Definieer een concept dat bepaald of een type `T` een geheel of floating point getal bevat. Je kunt hierbij gebruik maken van de in de standaard bibliotheek gedefinieerde concepten, zie https://en.cppreference.com/w/cpp/concepts.html. Beperk vervolgens de functietemplate `harmonisch_gemiddelde` tot alleen die types waarvoor dit concept geldt. De bovenstaande code met de `vector<string>` zal dan de volgende foutmelding geven:

```
error: no matching function for call to ↩
 ↪ 'harmonisch_gemiddelde( ↩
 ↪ std::vector<std::__cxx11::basic_string<char> >&)'
note: constraints not satisfied
```

Class template

In veel DSP (Digital Signal Processing) applicaties wordt gebruik gemaakt van een delay line[2] waarmee een signaal een aantal (D) sampletijden vertraagd kan worden. In het z-domein geven we zo'n delay line aan met de formule z^{-D}.

Een programmeur heeft de class `Delay_line` gedeclareerd in `delay_line.cppm`. Een object van deze class kan gebruikt worden als een delay line voor samples van het type **`int`** die 8 sampletijden vertraagd worden. De functie `put()` kan aangeroepen worden om het huidige sample in de delay line te plaatsen. Na een sampletijd kan daarna de memberfunctie `get()` aangeroepen worden om een sample uit de delay line te halen. De samples die uit de delay line komen zijn met 8 sampletijden vertraagd. Dus nadat sample 9 in de delay line is geplaatst, komt sample 1 uit de delay line. De functies `put()` en `get()` moeten dus om en om aangeroepen worden. Als de functie `put()` meerdere keren achter elkaar wordt aangeroepen, dan wordt steeds hetzelfde sample in de delay line overschreven. Als de functie `get()` meerdere keren achter elkaar wordt aangeroepen, dan wordt steeds hetzelfde sample uit de delay line gelezen.

De class `Delay_line` is als volgt gedefinieerd:

```
export class Delay_line {
public:
    Delay_line();
//  call put before get and alternate calls
    void put(int sample);
    int get();
private:
    array<int, 9> buffer;
    typename decltype(buffer)::size_type index;
    bool get_called;
};
```

De implementatie van alle memberfuncties en een testprogramma vind je in `delay_line.cpp` en `test_delay_line.cpp`.

[2] Zie eventueel: https://en.wikipedia.org/wiki/Digital_delay_line.

Na verloop van tijd heeft deze programmeur een delay line nodig om samples van het type **float** vier sampletijden te vertragen. Hij besluit om hier geen aparte class voor te maken, maar om jou te vragen om een template class te schrijven die gebruikt kan worden om samples van het type `T`, `D` sampletijden te vertragen. De benodigde testen heeft hij al wel geschreven m.b.v. het GoogleTest framework.

4.3.5 In `delay_line.zip`[3] vind je `delay_line.h`, `delay_line.cpp`, `test_delay_line.cpp` en `CMakeLists.txt`. Voer de `test_delay_line.cpp` uit onder WSL, als het goed is slaagt deze test.

4.3.6 Zet de class `Delay_line` om in een class template `Delay_line<T, D>` die gebruikt kan worden om samples van het type `T`, `D` sampletijden te vertragen. Bedenk dat deze class template in de module interface file `delay_line.cppm` moet komen te staan. Pas de gegeven test `TEST(delay_line, int)` aan zodat daarin de class template gebruikt wordt. Zorg ervoor dat ook de, al gegeven test `TEST(delay_line, float)` slaagt.

4.3.7 Test of de class template `Delay_line<T, D>` ook gebruikt kan worden om samples van het UDT `Tijdsduur`, die je hebt gemaakt bij opdracht 4.2.2, met twee sampletijden te vertragen. Een test `TEST(delay_line, Tijdsduur)` is al gegeven in `test_delay_line.cpp`.

4.3.8 Heeft het zin om de template parameter `T` te beperken (to constrain) met een **concept**? Zo ja, definieer dan dit concept en beperk de template parameter `T` van de class template `Delay_line<T, D>` tot alleen die types waarvoor dit concept geldt.

[3] https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/Opdrachten/progs/delay_line.zip