Opdrachten week 6 les 1 – 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 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.

```
double harmonisch_gemiddelde(const vector<double>& v) {
    double totaal {0};
    for (const double& e: v) {
```

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

```
        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;
}
```

6.1.1 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.

6.1.2 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 in het programma `harmonisch_gemiddelde.cpp` door één functietemplate die dit mogelijk maakt. Je mag daarbij de functie `main` niet aanpassen.

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 het programma `delay_line.cpp` geschreven waarin de class `Delay_line` is geïmplementeerd. Een object van deze class kan gebruikt worden als een delay line voor samples van het type **int** die 8 sampletijden vertraagd worden. Na elke sampletijd kan de memberfunctie `get()` aangeroepen worden om een sample uit de delay line te halen en daarna kan de functie `put()` aangeroepen om het huidige sample in de delay line te plaatsen. De sample die uit de delay line komen zijn met 8 sampletijden vertraagd. Dus als sample 1 uit de delay line komt, dan wordt sample 9 in de delay line geplaatst. De functies

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

`get()` en `put()` moeten dus om en om aangeroepen worden. Als de functie `get()` meerdere keren achter elkaar wordt aangeroepen, dan wordt steeds hetzelfde sample uit de delay line gelezen. Als de functie `put()` meerdere keren achter elkaar wordt aangeroepen, dan wordt steeds hetzelfde sample in de delay line overschreven.

De class `Delay_line` is als volgt gedefinieerd:

```
class Delay_line {
public:
    Delay_line();
//  call get before put and alternate calls
    int get();
    void put(int sample);
private:
    array<int, 8> buffer;
    typename array<int, 8>::size_type index;
    bool put_called;
};
```

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

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. Het testprogramma heeft hij al wel geschreven.

6.1.3 Schrijf een class template `Delay_line<T, D>` die gebruikt kan worden om samples van het type `T`, `D` sampletijden te vertragen. Een testprogramma met de gewenste output is gegeven in `delay_line_float.cpp`.

6.1.4 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 5.2.2, met tien sampletijden te vertragen. Een testprogramma met de gewenste output is gegeven in `delay_line_tijd.cpp`. Pas de class `Tijdsduur` indien nodig aan.