Opdrachten kwartaal 4 week 1 – Van C naar C++

C++ is een uitbreiding van de programmeertaal die je al kent: C. C++ is een veel uitgebreidere taal dan C en in C++ kun je allerlei dingen doen die in C niet (of alleen heel onhandig) gedaan kunnen worden, zoals objectgeoriënteerd en generiek programmeren. Alle dingen die je in C geleerd hebt, kun je ook toepassen in C++, maar sommige dingen die je in C hebt geleerd, kun je in C++ beter (handiger en efficiënter) op een andere manier doen. Deze opdracht is bedoeld als een eerste kennismaking met C++.

Lees nu de inleiding en hoofdstuk 1 van het dictaat Objectgeoriënteerd Programmeren in C++.

Je leert in deze opdracht werken met de standaard class `string` en de standaard objecten `cin` en `cout`. In C heb je leren werken met character arrays voor het opslaan van strings. In de standaard C++ library is het type `string` gedefinieerd dat veel eenvoudiger te gebruiken is dan character arrays. In C gebruik je de library `stdio` voor input en output. In de C++ standaard is de library `iostream` gedefinieerd die eenvoudiger te gebruiken is dan de library `stdio`. Deze nieuwe library is bovendien uitbreidbaar (zoals je later in opdracht 4.2.2 zult zien).

Je leert deze les hoe je:

- in C++ iets naar een outputdevice (bijvoorbeeld je beeldscherm) kunt sturen en iets van een inputdevice (bijvoorbeeld je toetsenbord) kunt inlezen;
- in C++ standaard classes (bijvoorbeeld `string`) kunt gebruiken.

Een van de eerste opvallende dingen die je tegenkomt als je van C overstapt naar C++ is dat de vertrouwde `printf` en `scanf` functies in C++ niet meer gebruikt worden maar vervangen zijn door (op het eerste gezicht) onduidelijke expressies zoals `cout << "i = " << i << '\n';` en `cin >> i`.

1 Input en output met << en >>

Je bent gewend om in C programma's de functies uit de bibliotheek `stdio` te gebruiken voor input en output. De meest gebruikte functies zijn `printf` en `scanf`. Deze functies zijn echter niet 'type veilig' omdat de inhoud van de als eerste argument meegegeven format string pas tijdens het uitvoeren van het programma verwerkt wordt. De compiler merkt het dus niet als de 'type aanduidingen' zoals `%d` die in de format string gebruikt zijn niet overeenkomen met de typen van de volgende argumenten. Bovendien is het niet mogelijk om een eigen format

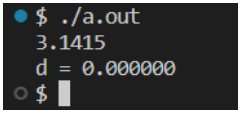
type-aanduiding aan de bestaande aanduidingen toe te voegen. Om deze redenen is in de ISO/ANSI C++-standaard naast de oude `stdio`-bibliotheek (om compatibel te blijven) ook een nieuwe I/O library `iostream` opgenomen. Bij het ontwerpen van nieuwe software kun je het best van deze nieuwe library gebruik maken. De belangrijkste output faciliteiten van deze library zijn de standaard output stream `cout` (vergelijkbaar met `stdout`) en de bijbehorende << operator. De belangrijkste input faciliteiten van deze library zijn de standaard input stream `cin` (vergelijkbaar met `stdin`) en de bijbehorende >> operator.

1.1 De problemen met `printf` en `scanf` in C

In een C programma kunnen de functies `scanf` en `printf` als volgt gebruikt worden om een variabele van het type **`double`** in te lezen en meteen weer af te drukken:

```c
#include <stdio.h>
//  ...
    double d = 0;
    scanf("%d", d); // deze regel bevat twee fouten!
    printf("d = %lf\n", d);
```

Zie jij de fouten? Op zich is het geen probleem als je de fouten niet ziet, maar het is wel een probleem dat de *C compiler* deze fouten niet ziet!s Als je de bovenstaande code compileert dan geeft de compiler geen errors[1]. Als je het programma start lijkt alles, in eerste instantie, in orde. Je kunt gewoon een getal invoeren. Er verschijnt echter geen correcte uitvoer, zie figuur 1.



Figuur 1: De uitvoer van het bovenstaande C-programma.

Bij het gebruik van een andere compiler of platform kan het ook gebeuren dat het programma afgebroken wordt met een foutmelding. Blijkbaar wordt de variabele `d` niet correct ingelezen in de functie `scanf`. Als je nog eens goed naar de aanroep van `scanf` kijkt, dan zie je hopelijk dat je als tweede argument het adres van de variabele `d` moet meegeven (in plaats van waarde

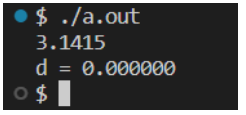
[1] De gcc C-compiler geeft, als de optie `-Wall` gebruikt wordt, wel twee waarschuwingen:

- format '%d' expects argument of type '**`int`** *', but argument 2 has type '**`double`**';
- 'd' is used uninitialized in this function.

van de variabele d). Als je de aanroep van `scanf` als volgt wijzigt, dan geeft de compiler nog steeds geen error[2]:

```
scanf("%d", &d); // deze regel bevat één fout!
```

Als je deze verbetering toepast en het programma opnieuw compileert en uitvoert, dan verschijnt nog steeds de foutieve uitvoer, zie figuur 2.



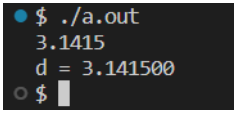
Figuur 2: De uitvoer van het bovenstaande programma na het verwijderen van de eerste fout.

Het resultaat is echter niet correct. Als we met `scanf` een variabele van het type **`double`** willen inlezen moeten we de format specifier `%lf` gebruiken in plaats van `%d` (`%d` is bedoeld voor het inlezen van een ariabele van het type **`int`**).

De aanroep van `scanf` moet dus als volgt gewijzigd worden:

```
scanf("%lf", &d); // deze regel is correct!
```

Als je deze verbetering toepast en het programma opnieuw compileert en uitvoert, dan verschijnt wel de verwachte uitvoer, zie figuur 3.



Figuur 3: De uitvoer van het bovenstaande programma na het verwijderen van beide fouten.

De functie `printf` heeft soortgelijke problemen. Deze functie geeft verkeerde uitvoer als de format specifier niet klopt. Ook hier geeft de C compiler geen error.

1.2 De oplossing in C++: het gebruik van `cin` met >> en `cout` met <<

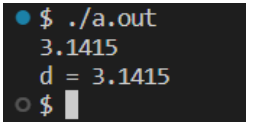
In een C++ programma kun je een variabele van het type **`double`** als volgt inlezen en meteen weer afdrukken:

```
#include <iostream>
```

[2] De gcc C-compiler geeft, als de optie `-Wall` gebruikt wordt, nog wel een waarschuwing: format '%d' expects argument of type '**`int`** *', but argument 2 has type '**`double`** *'.

```
// ...
    double d;
    std::cin >> d; // lees d in vanaf het toetsenbord
    std::cout << "d = " << d << '\n'; // druk d af op het scherm ↩
 ↪ en ga naar het begin van de volgende regel
```

Zoals je ziet in figuur 4 werkt deze code zoals in het commentaar staat toegelicht.



Figuur 4: De uitvoer van het bovenstaande C++-programma.

In de include file `iostream` is de variabele (eigenlijk het object, maar dat wordt later pas duidelijk) `std::cin` gedefinieerd. Deze variabele stelt het toetsenbordgeheugen voor. Je kunt data uit dit geheugen 'trekken' met behulp van de operator >>. De expressie `std::cin >> d` zorgt er dus voor dat de waarde die op het toetsenbord wordt ingevoerd wordt ingelezen in de variabele `d`. Merk op dat je *niet* op hoeft te geven welk type moet worden ingelezen (in dit geval een **`double`**). De C++-compiler 'zoekt' dit zelf voor je uit (later zullen we zien dat hier gebruik gemaakt wordt van *operator overloading*). Het is ook niet nodig om het adres van `d` op te geven. De C++-compiler 'snapt' zelf dat de ingelezen waarde naar het adres van de variabele `d` moet worden weggeschreven (later zullen we zien dat hier gebruik gemaakt wordt van een *reference*). Omdat de operator >> gebruikt wordt om data in te lezen vanuit `std::cin` wordt deze operator in dit geval de *get* operator genoemd.

Kijk nog eens naar de fouten in de aanroep van `scanf` in het bovenstaande C-programma. Deze fouten kun je dus in C++ niet maken als je `std::cin` gebruiken omdat de C++-compiler zelf *uitzoekt* of de variabele zelf, of het adres van de variabele moet worden gebruikt en wat het type is dat moet worden ingelezen. Handig!

Natuurlijk kun je bij het gebruik van `std::cin` ook fouten maken. Je kunt bijvoorbeeld per ongeluk toch het adres van `d` opgeven (omdat je dat bij `scanf` zo gewend was). In dit geval krijg je netjes een foutmelding.

In de include file `iostream` is ook de variabele (eigenlijk het object, maar dat wordt later pas duidelijk) `std::cout` gedefinieerd. Deze variabele stelt het beeldschermgeheugen voor. Je kunt data naar dit geheugen 'duwen' met behulp van de operator <<. De expressie `std::cout << "d = " << d << '\n'` zorgt er dus voor dat eerst de tekst `d =` op het scherm verschijnt. Vervolgens wordt de waarde van de variabele `d` op het scherm gezet en tot slot

gaat de cursor naar het begin van de volgende regel. Merk op dat je niet op hoeft te geven welk type moet worden afgedrukt (in dit geval een **double**). De compiler 'zoekt' dit zelf voor je uit (later zullen we zien dat hier gebruik gemaakt wordt van *operator overloading*). Omdat de operator << gebruikt wordt om data weg te schrijven naar `std::cout` wordt deze operator in dit geval de *put* operator genoemd.

Zoals je ziet moet je bij het gebruik van namen die in de ISO/ANSI C++ standaard include files (zoals `iostream`) gedeclareerd zijn (zoals `cin` en `cout`) steeds aangeven dat de naam uit de standaard wil gebruiken door er `std::` voor te zetten. Dit is gedaan om deze include files ook te kunnen gebruiken in (oude) bestaande programma's waarin door de programmeur al (toevallig) dezelfde namen gebruikt zijn. In het bovenstaande programma kun je dus in plaats van de variabele naam `d` ook de naam `cin` gebruiken. Omdat de `cin` die geïnclude wordt uit `iostream` altijd vooraf gegaan wordt door `std::` levert dit (voor de compiler) geen verwarring op.

```
#include <iostream>
// ...
    double cin;
    std::cin >> cin; // lees cin in vanaf het toetsenbord
    std::cout << "cin = " << cin << '\n'; // druk cin af op het ↩
 ↪ scherm en ga naar het begin van de volgende regel
```

Bij het schrijven van nieuwe programma's kunnen we ook besluiten om de in de standaard include files gebruikte namen niet ook zelf te gaan gebruiken. In dit geval kunnen we aangeven dat onbekende namen altijd afkomstig zijn uit de geïnclude files, dit doe je als volgt:

```
#include <iostream>
using namespace std;
// ...
    double d;
    cin >> d; // lees d in vanaf het toetsenbord
    cout << "d = " << d << '\n'; // druk d af op het scherm en ga ↩
 ↪ naar het begin van de volgende regel
```

De `iostream` library is zeer uitgebreid. Verderop in deze practicumopgave komen we hier nog even op terug.

2 Het type `string`

We zullen eerst de problemen met strings in C op een rijtje zetten en daarna het nieuwe type `string` uit C++ bespreken.

2.1 De problemen met strings in C

De programmeertaal C heeft geen 'echt' string type. Als je in C een string (rij karakters) wilt opslaan dan doe je dat in een character array. In C geldt de afspraak dat elke string wordt afgesloten door een zogenaamd nul-karakter: `'\0'`. Voorbeeld:

```
char naam[] = "Harry"; // deze array bevat 6 karakters!
```

Een string in C heeft de volgende problemen:

- Een character array is statisch (de lengte wordt tijdens het compileren bepaald).
- Het nul-karakter als afsluiting is onhandig (het maximale aantal 'nuttige' karakters dat in de string kan worden opgeslagen is één minder dan de lengte van de array).
- De toekenningsoperator en de vergelijkingsoperatoren zijn niet bruikbaar (in plaats daarvan moeten de `strxxx`-functies gebruikt worden).

Als voorbeeld bekijken we het volgende C-programma:

```
#include <string.h>
#include <stdio.h>

int main(void) {
    char naam[10];
    naam = "Harry"; // Fout! Zie opmerking 1.
    strcpy(naam, "Harry"); // OK
    printf("%s\n", naam);
    strcpy(naam, "Willem-Alexander"); // Fout! Zie opmerking 2.
    strcpy(naam, "Alex"); // OK
    printf("%s\n", naam);
    if (naam == "Alex") { // Fout! Zie opmerking 3.
        // ...
    }
    if (strcmp(naam, "Alex") == 0) { // OK
        // ...
    }
}
```

Opmerkingen:

1. Bij deze regel geeft de gcc C-compiler de volgende foutmelding: “assignment to expression with array type”. Aan een array variabele (`naam`) kun je namelijk niets toekennen. Als je de array wilt vullen dan moet je dat met de functie `strcpy` (gedefinieerd in `<string.h>`) doen.
2. Deze regel geeft tijdens het compileren geen foutmelding of warning. Tijdens het uitvoeren van het programma kan het programma echter vastlopen of zich vreemd gaan gedragen! Dit komt doordat er 17 karakters (even natellen, en het nul-karakter niet vergeten) naar de array `naam` worden gekopieerd. Terwijl er maar 10 karakters gereserveerd zijn. In C wordt hier echter helemaal niet op gecontroleerd en de 7 extra karakters worden gewoon naar het geheugen geschreven (achter de 10 gereserveerde karakters). Dit kan tot gevolg hebben dat een ander deel van je programma plotseling niet meer correct werkt omdat een variabele uit dit deel van het programma overschreven wordt. De ‘wet van bedrog’[3] zegt: Het programma zal pas vastlopen door deze fout als je het aan je belangrijkste klant demonstreert!
3. Ook deze regel geeft tijdens het compileren geen foutmelding[4]. De vergelijking `naam == "Alex"` levert altijd **`false`** op! Als je twee array variabelen met elkaar vergelijkt, dan worden hun *adressen* met elkaar vergeleken. Als je de *inhoud* van de arrays met elkaar wilt vergelijken moet je de functie `strcmp` (gedefinieerd in `<string.h>`) gebruiken.

2.2 De oplossing in C++: het type `string`

De programmeertaal C++ heeft wel een ‘echt’ string type. Dit type is gedefinieerd in de standaard library. Later zul je leren dat string geen ‘ingebouwd’ type is maar een zogenoemde class. Voor het gebruik maakt dat echter niet uit.

Voorbeeld:

```
#include <string>
using namespace std;
// ...
    string naam {"Harry"}; // deze string bevat 5 karakters.
```

[3] Zie https://nl.wikipedia.org/wiki/Wet_van_bedrog.

[4] De gcc C-compiler geeft, als de optie `-Wall` gebruikt wordt, wel een waarschuwing: “comparison with string literal results in unspecified behavior”.

Een `string` in C++ heeft de volgende voordelen ten opzichte van een string uit C:

- Een `string` is *dynamisch* (de lengte kan tijdens het uitvoeren van het programma, indien nodig, worden aangepast, er is dus ook geen maximale lengte!).
- De toekenningsoperator en de vergelijkingsoperatoren zijn gewoon bruikbaar.
- Het type `string` bevat veel extra functionaliteit.

Als voorbeeld bekijken we het volgende C++-programma:

```
#include <string> // Zie opmerking 1.
#include <iostream>
using namespace std;

int main() {
    string naam;
    naam = "Harry"; // Zie opmerking 2.
    cout << naam << endl;
    naam = "Willem-Alexander"; // Zie opmerking 3.
    cout << naam << '\n';
    if (naam == "Willem-Alexander") { // Zie opmerking 4.
        cout << "Hoi Alex!\n";
    }
}
```

Opmerkingen:

1. De C++ include file <string> is dus heel wat anders dan de C include file <string.h>. Als je in een C++-programma toch de oude C strings wilt gebruiken (om oude C code te hergebruiken) dan kun je de oude `strxxx` functies includen met de include file <cstring>.
2. Je kunt gewoon een waarde toekennen aan een variabele van het type `string` met behulp van de operator =.
3. Het type `string` is dynamisch en 'groeit' als dat nodig is!
4. Je kunt variabelen van het type `string` gewoon vergelijken met behulp van de operator ==.

2.3 Je eerste stap op weg naar objectoriëntatie.

Nu komt de verrassing: een variabele van het type (eigenlijk de class) `string` is geen gewone variabele maar een *object*! Wat dat precies betekent, wordt later nog uitgebreid behandeld.

Op dit moment zullen we alleen bekijken wat dit betekent voor het gebruik van objecten (variabelen) van de class (het type) `string`.

Objecten zijn vergelijkbaar met gewone variabelen: ze hebben een naam en je kunt er 'iets' in opslaan. Maar met objecten kun je iets wat met gewone variabelen niet kan. Je kunt objecten boodschappen (*messages*) sturen.

Je kunt een message naar een object sturen om het object een *vraag* te stellen. Als je wilt weten hoeveel karakters een object van de class `string` bevat dan kun je dat object de message `size` sturen. Het antwoord op deze message is dan een integer die het aantal karakters weergeeft.

Bijvoorbeeld:

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

int main() {
    string naam {"Willem-Alexander"};
    cout << "De naam " << naam << " bevat " << naam.size() << " ↩
 ↪ karakters.\n";
}
```

De uitvoer van dit programma is gegeven in figuur 5.

```
$ ./a.out
De naam Willem-Alexander bevat 16 karakters.
$
```

Figuur 5: De uitvoer van het bovenstaande C++-programma.

De syntax voor het versturen van een message is:

```
naam_van_object.naam_van_message(parameters)
```

Dus eerst de naam van het object waar de message naartoe verstuurd moet worden (dat object wordt de *receiver* van de message genoemd) dan een punt gevolgd door de naam van de message en tot slot een haakje openen en een haakje sluiten met daartussen eventuele *parameters*. De message `size` heeft geen parameters. Het versturen van een message lijkt een beetje op het aanroepen van een functie. In C++ wordt het versturen van een message meestal het aanroepen van een *memberfunctie* genoemd. Toch is er een duidelijk verschil

tussen een memberfunctie (message) en een functie: een memberfunctie heeft een receiver en een gewone functie niet!

Je kunt ook een memberfunctie van een object aanroepen om het object iets te laten *doen*. Na afloop van de memberfunctie is het object dan veranderd. Als je bijvoorbeeld iets aan een object van de class `string` wilt toevoegen dan kun je dat doen door de memberfunctie append van dit object aan te roepen. De `string` die moet worden toegevoegd moet je als argument van de memberfunctie meegeven.

Bijvoorbeeld:

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

int main() {
    string naam {"Willem-Alexander"};
    naam.append(" en Maxima");
    cout << naam << '\n';
}
```

De uitvoer van dit programma is gegeven in figuur 6.

```
$ ./a.out
Willem-Alexander en Maxima
$
```

Figuur 6: De uitvoer van het bovenstaande C++-programma.

3 De mogelijkheden van het type `string`

In deze paragraaf worden enkele mogelijkheden van het type `string` gegeven.

- **Vergelijken** met operatoren >, <, >=, <=, == en != (werking spreekt voor zich).
- **Losse karakters opvragen** met de operator `[]`.
  `char c {s[2]};`
  Als s gelijk is aan `"Maxima"`, dan wordt het karakter c gelijk aan `'x'`.
- **Losse karakters vervangen** met de operator `[]`.
  `s[3] = 'a';`
  Als s gelijk is aan `"Maxima"`, dan wordt het s gelijk aan `"Maxama"`.

- **Toekennen** met operator = of met `assign`. Met `assign` kun je ook een deel van de ene `string` aan de andere `string` toekennen.
  `s1.assign(s2, 7, 4);`
  `s1` wordt gelijk aan het deel van `s2` dat begint op index 7 en 4 karakters lang is. Als `s2` gelijk is aan `"Willem-Alexander"`, dan zal `s1` gelijk worden aan `"Alex"`.
- **Toevoegen** met operatoren + en += of met append. Met append kun je ook een deel van de ene `string` achter de andere `string` plakken. De tweede parameter geeft aan bij welke index het betreffende deel begint en de derde parameter geeft aan hoe lang het deel is.
  ```
  string s1 {"Willem"};
  s1 += "-Alexander";
  string s2 {"Maxima"};
  s2 += " en ";
  s2.append(s1, 7, 4);
  ```
  `s2` wordt gelijk aan `"Maxima en Alex"`.
- **Invoegen** met `insert`.
  `s1.insert(4, s2);`
  Voegt `s2` op index 4 in `s1` in. Als `s1` gelijk is aan `"Maxima"` en `s2` gelijk is aan `"Willem-Alexander"`, dan wordt `s1` gelijk aan `"MaxiWillem-Alexanderma"`.
  Met `insert` kun je ook een deel van de ene string in de andere toevoegen. De derde parameter geeft aan bij welke index het betreffende deel begint en de vierde parameter geeft aan hoe lang het deel is.
  `s1.insert(4, s2, 7, 4);`
  Als `s1` gelijk is aan `"Maxima"` en `s2` gelijk is aan `"Willem-Alexander"` dan wordt `s1` gelijk aan `"MaxiAlexma"`.
- **Verwijderen** met `erase`.
  `s1.erase(3, 10);`
  Als `s1` gelijk is aan `"Willem-Alexander"`, dan wordt `s1` gelijk aan `"Wilder"`.
- **Vervangen** met `replace`.
  `s1.replace(0, 4, "Jongs");`
  Als `s1` gelijk is aan `"Maxima"`, dan wordt `s1` gelijk aan `"Jongsma"`.
- **Gedeelte opvragen** met `substr`.
  `s1 = s2.substr(7);`
  `s1` wordt gelijk aan het deel van `s2` dat begint op index 7. Als `s2` gelijk is aan `"Willem-Alexander"` dan zal `s1` gelijk worden aan `"Alexander"`. Je kunt ook als

tweede parameter opgeven hoeveel karakters er gekopieerd moeten worden.
`s1 = s2.substr(7, 4);`
`s1` wordt gelijk aan het deel van `s2` dat begint op index 7 en 4 karakters lang is. Als `s2` gelijk is aan `"Willem-Alexander"`, dan zal `s1` gelijk worden aan `"Alex"`.

- **Zoeken** met `find`.
  `int i {s1.find("Alex")}; // niet helemaal goed! Zie hieronder`
  Als `s1` gelijk is aan `"Willem-Alexander"`, dan wordt `i` gelijk aan 7. Als de string `"Alex"` niet in `s1` voorkomt, krijgt `i` de waarde `string::npos` (een in de class `string` gedefinieerde constante). Het gebruik van het type `int` is niet correct, je moet in plaats van het type `int` het type `string::size_type` gebruiken:
  `string::size_type i {s1.find("Alex")};`
  In de class `string` wordt namelijk het type `size_type` gedefinieerd dat gebruikt moet worden om indexwaarden van een `string` in op te slaan. Het zou namelijk zo kunnen zijn dat een indexwaarde van een string niet in een `int` past (bijvoorbeeld op een 64-bit machine). Je kunt ook de compiler zelf laten uitzoeken welk type `find` teruggeeft:
  `auto i {s1.find("Alex")};`
  Als tweede parameter kun je een index meegeven, het zoeken begint dan bij die index.
  `auto i {s1.find("le")};`
  `auto j {s1.find("le", 7)};`
  Als `s1` gelijk is aan `"Willem-Alexander"`, dan wordt `i` gelijk aan 3 en `j` gelijk aan 8. Zoals je hebt gezien doorzoekt `find` de string van voor naar achter. Je kunt de memberfunctie `rfind` gebruiken om van achter naar voor te zoeken.
  `auto i {s1.rfind("le")};`
  Als `s1` gelijk is aan `"Willem-Alexander"`, dan wordt `i` gelijk aan 8.
- **Zoeken** met `find_first_of`.
  `auto i {s1.find_first_of("aeiou")};`
  Zoek de eerste letter uit `s1` die voorkomt in de als parameter meegegeven `string`. Als `s1` gelijk is aan `"Maxima"`, dan wordt `i` gelijk aan 1. Als de string `s1` geen van de karakters `'a'`, `'e'`, `'i'`, `'o'` of `'u'` bevat, krijgt `i` de waarde `string::npos`. Als tweede parameter kun je een index meegeven, het zoeken begint dan bij die index.
  `auto i {s1.find_first_of("aeiou", 2)};`
  Als `s1` gelijk is aan `"Maxima"` dan wordt `i` gelijk aan 3. Het gebruik van de functies: `find_last_of`, `find_first_not_of` en `find_last_not_of` spreekt, denk ik, voor zich.

Voor het complete overzicht verwijs ik je naar de online C++ reference guide[5]. De typenaam `string` blijkt een andere naam te zijn voor het template type `basic_string<char>`. Het begrip template wordt pas later behandeld.

4 Nogmaals: De `iostream` library

Zoals je hierboven hebt gelezen gebruiken we in C++ de input/output library `iostream` in plaats van de C library `stdio`. Het zal je niet verbazen dat de variabelen `cin` en `cout` die je in het begin van deze opdracht hebt leren kennen geen variabelen, maar objecten zijn. Je kunt bij deze objecten (net als objecten van de class `string`) dus ook memberfuncties aanroepen. Elke class definieert echter zijn eigen memberfuncties. Het object `cout` is een object van de class `ostream` en het object `cin` is een object van de class `istream`. Later zal blijken dat dit niet helemaal klopt, maar dat maakt voor dit verhaal niets uit.

Voor een object van de class `string` kun je bijvoorbeeld de memberfunctie `size` aanroepen om te vragen hoeveel karakters het object bevat. Voor `cout` kun je deze memberfunctie echter niet aanroepen (deze functie is namelijk niet gedefinieerd in de class `ostream`). De aanroep `cout.size()` geeft tijdens het compileren de volgende foutmelding: 'std::ostream' has no member named 'size'. Welke memberfuncties je voor `cout` en `cin` kunt aanroepen kun je opzoeken in de online C++ reference[6]. Zo kun je bijvoorbeeld de memberfuncties `fill` en `width` gebruiken:

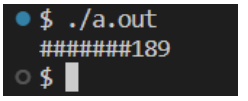
```
#include <iostream>
using namespace std;

int main() {
    cout.fill('#');
    cout.width(10);
    int i {189};
    cout << i << '\n';
}
```

De uitvoer van dit programma is gegeven in figuur 7.

[5] Zie: http://www.cppreference.com/cppstring/index.html.

[6] Zie: http://www.cppreference.com/cppio/index.html.



Figuur 7: De uitvoer van het bovenstaande C++-programma.

5 Voorbeeldprogramma

In listing 1 vind je de header file `splits_emailadres.h` waarin de functie `splits_emailadres` is gedeclareerd waarmee een emailadres gesplitst kan worden in de gebruikers- en de domeinnaam. De implementatie van deze functie is gegeven in het bestand `splits_emailadres.cpp`, zie listing 2. Hierin wordt gebruik gemaakt van de hierboven geïntroduceerde class `string`. Zoals je ziet is de implementatie van deze functie eenvoudig, er wordt niet gecontroleerd of het opgegeven e-mailadres wel voldoet aan de regels die gegeven zijn in RFC 5322[7].

```
#ifndef _HR_BroJZ_splits_Emailadres_
#define _HR_BroJZ_splits_Emailadres_

#include <string>

struct gesplitst_emailadres
{
    std::string gebruiker;
    std::string domein;
    bool geldig;
};

gesplitst_emailadres splits_emailadres(std::string emailadres);

#endif
```

Listing 1: De include file `splits_emailadres.h`.

In het bestand `run_splits_emailadres.cpp`, zie listing 3 wordt deze functie gebruikt om een e-mailadres in te lezen vanaf het toetsenbord, te splitsen in de gebruikers- en de domeinnaam en deze namen af te drukken op het beeldscherm. Hierin wordt gebruik gemaakt van de hierboven geïntroduceerde objecten `cin` en `cout`.

[7] Zie eventueel: https://datatracker.ietf.org/doc/html/rfc5322#section-3.4.1.

```
#include <string>
#include "splits_emailadres.h"

using namespace std;

gesplitst_emailadres splits_emailadres(string emailadres) {
    string::size_type index_aapje {emailadres.find("@")};
    gesplitst_emailadres resultaat;
    if (index_aapje != string::npos) {
        resultaat.geldig = true;
        resultaat.gebruiker = emailadres.substr(0, index_aapje);
        resultaat.domein = emailadres.substr(index_aapje + 1);
    }
    else {
        resultaat.geldig = false;
    }
    return resultaat;
}
```

Listing 2: De implementatie van de functie splits_emailadres, zie splits_emailadres.cpp.

```
#include <iostream>
#include <string>
#include "splits_emailadres.h"

using namespace std;

int main() {
    cout << "Geef je e-mailadres: ";
    string emailadres;
    cin >> emailadres;
    gesplitst_emailadres res {splits_emailadres(emailadres)};
    if (res.geldig) {
        cout << "Gebruiker: " << res.gebruiker << '\n';
        cout << "Domein:    " << res.domein << '\n';
    }
    else {
        cout << emailadres << " is geen geldig e-mailadres!\n";
    }
}
```

Listing 3: Het gebruik van de functie splits_emailadres, zie run_splits_emailadres.cpp.

In het bestand test_splits_emailadres.cpp, zie listing 4 wordt deze functie getest met behulp van het testframework GoogleTest.

```
#include <gtest/gtest.h>
#include "splits_emailadres.h"

TEST(splits_emailadres, geldig)
{
    gesplitst_emailadres res ↩
 ↪ {splits_emailadres("j.z.m.broeders@hr.nl")};
    EXPECT_EQ(res.geldig, true);
    EXPECT_EQ(res.gebruiker, "j.z.m.broeders");
    EXPECT_EQ(res.domein, "hr.nl");
    res = splits_emailadres("Willem-Alexander@koninklijkhuis.org");
    EXPECT_EQ(res.geldig, true);
    EXPECT_EQ(res.gebruiker, "Willem-Alexander");
    EXPECT_EQ(res.domein, "koninklijkhuis.org");
}

TEST(splits_emailadres, ongeldig)
{
    gesplitst_emailadres res ↩
 ↪ {splits_emailadres("www.hogeschoolrotterdam.nl")};
    EXPECT_EQ(res.geldig, false);
    res = splits_emailadres("Maxima.koninklijkhuis.org");
    EXPECT_EQ(res.geldig, false);
}
```

Listing 4: Testcode om de functie splits_emailadres te testen met behulp van het GoogleTest framework, zie test_splits_emailadres.cpp.

6 Opdrachten (eindelijk...)

4.1.1 In dit kwartaal gaan we weer gebruik maken van de tools die je in kwartaal 3 hebt leren gebruiken zoals: Visual Studio Code, WSL, ArchLinux, CMake en het GoogleTest framework. Het begin van het kwartaal is een goed moment om alle packages die je gebruikt in ArchLinux onder WSL te updaten. 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. Update alle packages met het commando:

```
sudo pacman -Syu
```

4.1.2 In deze opdracht ga je het hierboven gegeven voorbeeldprogramma uitvoeren en testen. Ga naar het directory `Opdrachten` door in het terminal window in te typen:

```
cd ~/Opdrachten/
```

Download het bestand `splits_emailadres.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/splits_emailadres.zip
unzip splits_emailadres.zip
rm splits_emailadres.zip
```

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

```
code -r splits_emailadres
```

Build het project door op de Build-knop te klikken of door op F7 te drukken. Je kunt nu het programma `run_splitsemailadres` uitvoeren door in de statusbalk op de Play-knop te klikken. Je kunt de functie testen door gebruik te maken van de TestMate-plugin. Als het goed is slagen alle testen. Bestudeer de inhoud van de bestanden: `splits_emailadres.h`, `splits_emailadres.cpp`, `run_splits_emailadres.cpp`, `test_splits_emailadres.cpp` en `CMakeLists.txt`. Stel vragen aan je docent als je iets niet begrijpt.

4.1.3 Het adres van een webpagina (een URL) kan opgesplitst worden in 4 delen: het protocol, het domein, de directory en de file.

Bijvoorbeeld:
`https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Home.md`
kan worden gesplitst in:
`protocol: https`
`domein: bitbucket.org`
`directory: HR_ELEKTRO/ems31/wiki`
`file: Home.md`

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

```
cd ~/Opdrachten/
```

Download het bestand `splits_URL.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/splits_URL.zip
unzip splits_URL.zip
rm splits_URL.zip
```

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

```
code -r splits_URL
```

In het directory `splits_URL` bevinden zich de volgende bestanden:

- `CMakeLists.txt` de CMake configuratie file.
- `splits_URL.h` de include file waarin de functie `splits_URL` is gedeclareerd.
- `splits_URL.cpp` de definitie van de functie `splits_URL`. De implementatie moet je nog zelf schrijven.
- `run_splits_URL.cpp` het programma dat de functie `splits_URL` gebruikt.
- `test_splits_URL.cpp` de testcode om de functie `splits_URL` te testen met behulp van het GoogleTest framework.

Schrijf de benodigde C++-code in het bestand `splits_URL.cpp` om de functie `splits_URL` te implementeren. Je hoeft hierbij niet te controlleren of de URL voldoet aan de regels die gegeven zijn in RFC 3986[8]. Alles wat je moet doen is de URL splitsen in

[8] Zie eventueel: https://datatracker.ietf.org/doc/html/rfc3986.

de 4 delen: protocol, domein, directory en file. De tekst voor `://` is het protocol, de tekst tussen de `://` en de eerstvolgende `/` is het domein, de tekst tussen de `/` waarmee het domein wordt afgesloten en de laatstvolgende `/` is de directory en de tekst na de laatste `/` na het domein is de file. Als de meegegeven string geen `://` bevat, dan is de URL niet geldig.

Hou er rekening mee dat er niet altijd een directory en/of file aanwezig is.