

Übung 2: Strings

Lernziele

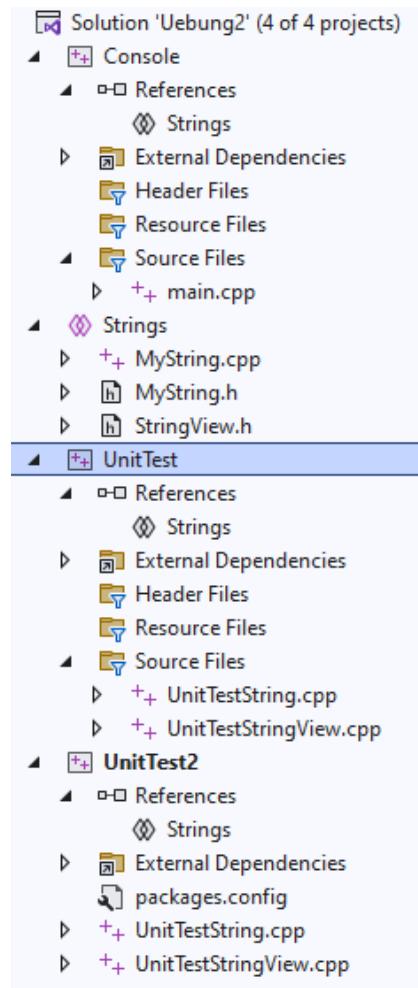
- Sie repetieren den Stoff aus der Vorlesung: Struktur eines C++-Programms, einfache und strukturierte Datentypen, Zeiger, Referenzen, Arrays, Klassen, Konstruktoren/Destruktoren, Verschiebesemantik und Operatoren überladen.
- Sie implementieren eine eigene String-View-Klasse, welche als Wrapper für C-Strings und deren Länge dienen kann.
- Sie implementieren eine eigene String-Klasse, welche kurze Zeichenketten direkt im String-Objekt verwaltet und nur für lange Zeichenketten den Heap verwendet.
- Sie testen Ihre beiden Implementierungen mit zur Verfügung gestellten Unit-Tests.
- Sie verstehen das Potential der Move-Semantik und die Auswirkungen auf die Performance.

1 Projektstruktur

Für diese Übung gehen Sie am besten von nebenstehender Projektstruktur aus:

Die Solution enthält vier Projekte:

1. *Console* ist eine einfache Konsolenanwendung, welche Sie für kleine Tests und den Einsatz der String-Klassen verwenden können. Diese Konsolenanwendung verwendet das *shared project* *Strings*. Durch Einfügen einer solchen Projektreferenz haben Sie direkten Zugriff auf die darin enthaltenen Dateien.
2. *Strings* ist ein *shared project*. Es enthält die Implementierungen der Klassen *StringView* und *String*. Für die Klasse *String* wird der Dateiname *MyString* empfohlen, weil es beim Namen *String* beim Kompilieren zu schwer auflösbaren Fehlern kommen kann.
3. *UnitTest* ist ein Projekt mit nativen Unit-Tests. Es enthält zwei Sets von Unit-Tests: *UnitTestStringView* für die Klasse *StringView* und *UnitTestString* für die Klasse *String*. In VS handelt es sich bei dem *UnitTest*-Projekt um eine DLL. Diese DLL kann nicht direkt ausgeführt werden, sondern wird innerhalb des *Test Explorers* ausgeführt. Dazu nachfolgend mehr.
4. *UnitTest2* ist ein Projekt mit Unit-Tests, welche das Test-Framework von Google verwenden. Es enthält wiederum zwei Sets von Unit-Tests: *UnitTestStringView* für die Klasse *StringView* und *UnitTestString* für die Klasse *String*. Diese Unit-Test sind vor allem für Benutzer von Linux und MacOS gedacht, können aber auch von Windows-Benutzer verwendet werden. Die beiden Sets von Unit-Test werden zu einem einzigen ausführbaren Programm kombiniert.



1.1 CMake und Visual Studio Solution

Die Projektstruktur ist so aufgebaut, dass Sie entweder die Datei «Uebung2.sln» in Visual Studio mit all darin enthaltenen Projekten und Projekteinstellungen öffnen können oder dass Sie den Solution-Ordner als *Folder* in Visual Studio öffnen können. Im letzteren Fall werden darin enthaltene CMake-Dateien «CMakeLists.txt» automatisch gelesen und ausgeführt. Es werden dann zwei Targets (*Console.exe* und *UnitTest2.exe*) erkannt, welche wie üblich gebildet und ausgeführt werden können.

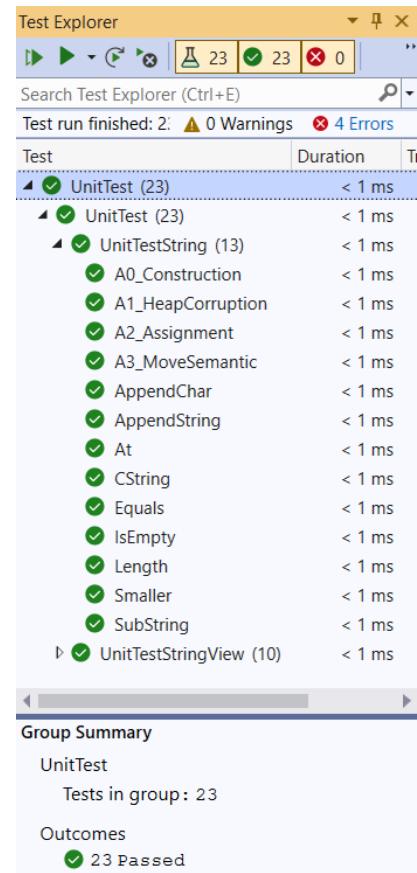
Die Ordnerstruktur mit den CMake-Dateien kann auf allen Plattformen, welche CMake unterstützen, verwendet werden. Die meisten C++-IDEs unterstützen CMake und daher sollten Sie keine grösseren Schwierigkeiten haben, diese Übung in Ihrer bevorzugten IDE zu bilden und auszuführen.

1.2 Unit Tests

In diesem Abschnitt lernen Sie, wie Sie in VS ein natives C++ Unit-Test-Projekt anlegen und ausführen können. Beachten Sie, dass die vorgegebene VS *Solution* schon ein fixfertiges natives Unit-Test-Projekt enthält.

Mit einem Rechtsklick im *Solution Explorer* auf Ihrer *Solution* können Sie *Add/New Project...* auswählen. Wählen Sie dann die Filter-Kategorie *Test* und danach das *Native Unit Test Project* aus. Ändern Sie den Projektnamen zu *UnitTest* und drücken Sie auf OK. Nun wird das Projekt im gleichnamigen Ordner mit der Testdatei «*unittest1.cpp*» automatisch angelegt. Ersetzen Sie die automatisch generierte Datei «*unittest1.cpp*» durch die Ihnen zur Verfügung gestellte. Bevor Sie die Unit-Tests ausführen, müssen Sie wiederum eine Referenz auf das Projekt *Strings* hinzufügen: Mit einem Rechtsklick im *Solution Explorer* auf das Projekt *UnitTest* können Sie *Add/Reference...* auswählen und dann das Projekt *Strings* auswählen.

Sobald Sie die Solution neu kompiliert haben, können Sie die Tests ausführen lassen. Dazu gehen Sie im Menü auf *Test/Run All Tests* oder zum Debuggen der Tests wählen Sie *Test/Debug All Tests*. Selbstverständlich können Sie auch nur die im *Test Explorer* selektierten Tests ausführen. Das Fenster *Test Explorer* sollte beim Ausführen der Tests automatisch geöffnet werden. Falls nicht, so können Sie es über *Test/Test Explorer* öffnen. Die vorangegangene Abbildung zeigt einen Ausschnitt aus dem *Test Explorer*.



2 Klasse **StringView**

String-Literale in C++ sind C-Strings mit einem terminierenden 0-Character, aber ohne Längenattribut. Solche String-Literale sind nach wie vor in fast jedem C/C++-Programm zu finden. Wird das String-Literal lediglich für die Initialisierung einer veränderbaren Zeichenkette benutzt, so wird in C++ eine Instanz der Klasse *string* mit einem Konstruktor erstellt, welcher ein C-String-Literal entgegennimmt. In Situationen, wo die Zeichenkette jedoch nicht verändert werden soll, eignet sich eine *StringView* sehr gut zur Verwaltung des C-String-Literals, weil sie einerseits sicherstellt, dass die Zeichenkette nicht verändert werden kann und weil sie noch ein Längenfeld neben der Speicheradresse der Zeichenkette verwaltet, so dass beides zusammen eine Einheit bildet.

Beachten Sie, dass eine in *StringView* verwaltete Zeichenkette nicht zwingend mit einem 0-Character terminiert sein muss, weil die *StringView* ja die Länge der Zeichenkette kennt. Eine *StringView* ist zudem nicht die Besitzerin der Zeichenkette und wird somit nie versuchen, den Speicherplatz für die Zeichenkette freizugeben. Darüber hinaus wird kein Konstruktor und keine Methode Speicherplatz auf dem Heap allozieren.

Eine *StringView* macht die Verwendung von zwei separaten formalen Parametern bei einer Übergabe eines String-Literals an eine Methode überflüssig. Im nachfolgenden Beispiel wird das String-Literal «hello, world!» an einen Konstruktor von *StringView* übergeben, welcher dann die Länge des übergebenen C-Strings bestimmt und im Längenfeld der *StringView*-Instanz speichert. Die Referenz zur *StringView*-Instanz in der Prozedur *foo(...)* verweist sowohl auf den Zeiger zum ersten Zeichen der Zeichenkette als auch zur Länge der Zeichenkette. In der Prozedur *foo(...)* wird die Zeichenkette Zeichen für Zeichen auf der Konsole ausgegeben, wobei vor der Ausgabe noch eine Konvertierung von Klein- zu Grossbuchstaben erfolgt. Die resultierende Ausgabe ist also: «HELLO, WORLD!».

```
void foo(const StringView& sv) {
    for (auto c : sv) {
        cout << (char)toupper(c);
    }
    cout << endl;
}
int main() {
    foo(StringView("hello, world!"));
}
```

StringView-Objekte sind nicht nur hilfreich bei der Bündelung von Speicheradresse und Länge der Zeichenkette, sondern sie sind auch maximal effizient, da sie keine Rechenzeit zur Laufzeit des Programms benötigen. Die maximale Performanz wird dadurch erreicht, dass alle Konstruktoren und Instanzmethoden *constexpr* sind, d.h. sie können alle bereits zur Kompilationszeit ausgewertet werden.

Das nachfolgende Beispiel macht deutlich, dass die Zeichenkettenlänge bereits zur Kompilationszeit bekannt ist, weil die Länge des C++-Arrays `data` eine `constexpr` sein muss.

```
constexpr StringView s("hello, world!");
array<int, s.length()> data;
cout << s << endl;
```

2.1 Aufgaben

Lösen Sie die nachfolgenden drei Teilaufgaben ineinander verschachtelt.

2.1.1 Implementierung

Gegeben ist die Schnittstellendefinition «StringView.h» der Klasse `StringView`. Da die Konstruktoren und Instanzmethoden alle `constexpr` sind, müssen diese direkt im gegebenen h-File implementiert werden. Die einzelnen Methoden sind im h-File kurz beschrieben.

Die Implementierungen der einzelnen Konstruktoren/Methoden können sehr kurz gehalten werden. Nur der Spaceship-Operator wird mehr als drei Zeilen Code benötigen. Beachten Sie, dass Sie in dieser Übung aus naheliegenden Gründen keine Methoden aus `<cstring>`, `<string>` und `<string_view>` verwenden dürfen.

Bei den meisten Methoden steht das Schlüsselwort `noexcept`. Damit wird ausgedrückt, dass die Methode keine Exception werfen wird. Bei allen anderen Methoden ist es Ihnen gestattet, Exceptions zu werfen. In C++ werfen Sie beispielsweise eine `std::out_of_range` Exception wie folgt:

```
throw std::out_of_range("out of range");
```

2.1.2 Konsolenanwendung

Implementieren Sie eine kleine Konsolenanwendung, in der Sie ein paar `StringView`-Objekte erstellen und neu implementierte Methoden gleich ausprobieren. Ein `StringView`-Objekt können Sie mittels dem Output-Stream-Operator `<<` auf die Konsole ausgeben.

2.1.3 Unit-Tests

Führen Sie die Unit-Tests wiederholt aus, bis alle Tests sowohl in der Debug- als auch der Release-Konfiguration erfolgreich ausgeführt werden. In der Release-Version werden keine zusätzlichen Initialisierungen vorgenommen und daher kann es passieren, dass Tests, welche in der Debug-Version korrekt ausgeführt werden, in der Release-Version nicht korrekt laufen.

3 Klasse String

Objekte unserer eigenen Klasse `String` repräsentieren veränderbare Zeichenketten, welche zwingend mit einem 0-Character terminiert sind, analog zu C-Strings. Eine solche 0-Terminierung wäre grundsätzlich nicht notwendig, aber sie vereinfacht die Konvertierung zu einem C-String wesentlich und daher verlangen wir diese 0-Terminierung.

Die hier vorgeschlagene Implementierung der Klasse `String` orientiert sich am Paradigma einer halbdynamischen Datenstruktur, ähnlich zu einem C++ `vector<char>`. Mit «halbdynamisch» ist gemeint, dass lange Zeichenketten in einem Array auf dem Heap abgespeichert werden und dieses Array automatisch durch ein grösseres ersetzt wird, wenn das bisherige zu klein geworden ist (siehe Methode `ensureCapacity(...)`). Um diese Halbdynamik zu realisieren, werden zwei Längen-Attribute benötigt:

- `m_size`: gibt die Länge der Zeichenkette in Anzahl `char` an; der terminierende 0-Character wird nicht mitgezählt;
- `m_capacity`: gibt die Kapazität bzw. die Länge des Arrays an; die Kapazität muss immer mindestens um eins grösser als die Länge der Zeichenkette sein, weil der terminierende 0-Character auch im Array abgespeichert werden muss.

Da das Anlegen und Verwalten von Arrays auf dem Heap ein Mehrfaches länger dauert als das Anlegen eines Arrays auf dem Stack, sollte dieser zeitliche Mehraufwand nur dann in Kauf genommen werden, wenn die Zeichenkettenlänge über eine vordefinierte `ShortCapacity` hinausgeht. Daher sollten kürzere Zeichenketten direkt auf dem Stack angelegt werden. Die Speicherallokation eines Arrays auf dem Heap (dynamisch) hat jedoch einen wichtigen Vorteil gegenüber derjenigen auf dem Stack (statisch): bei der Allokation auf dem Heap kann die Array-Länge zur Laufzeit gewählt werden, während sie beim Stack zur Kompilationszeit bekannt sein muss.¹ Diese Einschränkung führt dazu, dass die angesprochene

¹ Gewisse C++-Compiler ermöglichen auch eine zur Laufzeit berechnete Array-Länge bei Arrays auf dem Stack. Diese Fähigkeit ist nicht portable und muss vermieden werden.

ShortCapacity nicht zu gross gewählt werden sollte, weil es sonst bei sehr kurzen Zeichenketten zu einer Speicherverschwendug auf dem Stack kommen kann. Wir verwenden eine ShortCapacity von 16.

Die vorher gemachten Überlegungen führen zu den folgenden zwei Attributnen:

- `m_short`: statischer Buffer der Länge ShortCapacity wird nur für kurze Zeichenketten verwendet; ist die Zeichenkette gleich oder länger wie ShortCapacity, dann wird die ganze Zeichenkette in `m_data` gespeichert; damit ist sichergestellt, dass die ganze Zeichenkette in einem zusammenhängenden Speicherbereich liegt;
- `m_data`: dynamisch allozierter Buffer auf dem Heap für Zeichenketten der Länge ShortCapacity und länger; falls `m_data` ein `nullptr` ist, dann wird `m_short` stattdessen verwendet.

Zur Verwaltung von längeren Zeichenketten auf dem Heap benutzen wir einen `unique_ptr<char[]>`. Dieses Zeigerobjekt stellt sicher, dass die String-Instanz der alleinige Besitzer der Zeichenkette auf dem Heap ist, und dass der allozierte Speicherplatz automatisch freigegeben wird, wenn der Destruktor der String-Instanz ausgeführt wird.

3.1 Aufgaben

Lösen Sie die nachfolgenden drei Teilaufgaben ineinander verschachtelt.

3.1.1 Implementierung

Gegeben ist die Schnittstellendefinition «MyString.h» der Klasse `String`. Die einzelnen Konstruktoren, Methoden und Operatoren sind im h-File kurz beschrieben.

Implementieren Sie alle nicht bereits implementierten Methoden in einer neu zu erstellenden C++-Datei «MyString.cpp». Achten Sie darauf, keine Veränderungen in «MyString.h» vorzunehmen, da die zur Verfügung gestellten Unit-Tests auf dieser Schnittstelle basieren. Allenfalls weitere Hilfsmethoden, wie z.B. `stringLen(...)` zur Berechnung der Länge eines C-Strings können Sie als Modul-Methoden (z.B. `static size_t stringLen(...)`) innerhalb von «MyString.cpp» implementieren.

Beachten Sie, dass Sie in dieser Übung aus naheliegenden Gründen keine Methoden aus `<cstring>`, `<string>` und `<string_view>` verwenden dürfen.

Um den Code einfach zu halten, empfehle ich Ihnen, regen Gebrauch vom überladenen Index-Operator sowohl für lesenden als auch schreibenden Zugriff zu machen. Innerhalb des Index-Operators können Sie dann die Fallunterscheidung vornehmen, ob Sie auf `m_short` oder `m_data` zugreifen, um das Zeichen bzw. eine Referenz aufs Zeichen zu bekommen. Wenn Sie den Index-Operator auf das `this`-Objekt an der Position `i` anwenden möchten, dann schreiben Sie `(*this)[i]`.

3.1.2 Konsolenanwendung

Verwenden Sie die Konsolenanwendung, um ein paar `String`-Objekte zu erstellen und neu implementierte Methoden ausprobieren. Ein `String`-Objekt können Sie mittels dem Output-Stream-Operator `<<` auf die Konsole ausgeben.

3.1.3 Unit-Tests

Führen Sie die Unit-Tests aus der Datei «UnitTestString.cpp» wiederholt aus, bis alle Tests sowohl in der Debug- als auch der Release-Konfiguration erfolgreich ausgeführt werden können. In der Release-Version werden keine zusätzlichen Initialisierungen vorgenommen und daher kann es passieren, dass Tests, welche in der Debug-Version korrekt ausgeführt werden, in der Release-Version nicht korrekt laufen.

4 Abgabe

Senden Sie die folgenden drei Dateien **ungezippt** als E-Mail-Anhänge an christoph.stamm@fhnw.ch:

1. «StringView.h» mit den von Ihnen implementierten Methoden der Klasse `StringView`. Schreiben Sie Ihren **vollständigen Namen als Kommentar an den Anfang der Datei**. Dabei bezeugen Sie, dass Sie die Implementierung eigenständig erstellt haben.
2. «MyString.cpp» mit den von Ihnen implementierten Methoden der Klasse `String`. Schreiben Sie Ihren **vollständigen Namen als Kommentar an den Anfang der Datei**. Dabei bezeugen Sie, dass Sie die Implementierung eigenständig erstellt haben.
3. Erstellen Sie einen Screenshot von allen ausgeführten Unit-Tests.