SOFTWARE REQUIREMENTS SPECIFICATION

für Codegenerator GenTxtSrcCode

Programmentwurf C/C++ 2023 TIK/TIM/TIS/TIT22

Version 1.0

Version 1.0approved

Erstellt durch Th. Staudacher und A. Maus

Inhaltsverzeichnis

Revision History

- Version 1.0: Diese Version ist die erste Finale. Sie ist noch nicht mit den Kursen abgeglichen und einzelne Requirements können sich noch ändern, bzw hinzugefügt werden.

1 Einführung

1.1 Gültigkeit

Diese Version ist Version 1.0. Sie ist die erste unverbindliche Version.

1.2 Gruppengröße

Die empfohlene Gruppengröße für die Bearbeitung dieses Projekts sind 4 Teilnehmer. Die maximale Gruppengröße liegt bei 5 Teilnehmern (der Dozent kann Ausnahmen festlegen). Der Prüfungsumfang ändert sich nicht durch kleinere Gruppengrößen.

1.3 Abgabe der Prüfungsleistung

Jeder Student und Studentin gibt eigenständig seinen Programmentwurf bei Moodle ab, spätestens bis zum vorgesehenen Abgabezeitpunkt der unter Moodle vermerkt ist.

Anmerkung: Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg (DHBW) (Studien- und Prüfungsordnung DHBW Technik – StuPrO DHBW Technik) § 11 Versäumnis, Rücktritt, Täuschung, Ordnungsverstoß / 1

1.4 Aufgabenstellung

Erstellen Sie einen Codegenerator (im folgenden: Ihr Programm) der Text aus einer oder mehreren Dateien (im folgenden: Inputdateien) in Variablen speichert und diese als Header- und Sourcedateien (im folgenden: Outputdateien) ablegt. Die Inputdateien sollen per Kommandozeile an Ihr Programm übergeben werden können. Sonstige Optionen (Ablageort der Outputdateien, Format, etc siehe Anforderungen) sollen hierbei entweder ebenfalls per Kommandozeile oder durch spezielle Tags innerhalb der Inputdatei an Ihr Programm übergeben werden. Die Formatierung der erzeugten Header- und Source Datei ist nicht Umfang des Programmentwurfs, diese kann unter Zuhilfenahme des Kommandozeilenprogramms astyle erfolgen. Der Aufruf von astyle findet ebenfalls nicht im Programmentwurf statt.

Nähere Information zur Umsetzung sind in den Requirements der folgenden Kapitel detailliert aufgeführt.

1.5 Projekt Umfang und Eigenschaften des Produkts

1.6 Referenz

2 Allgemeine Beschreibung

2.1 Funktionale Anforderungen

Funktionale Anforderungen beschreiben was das Programm ausführen soll. Im Gegensatz zu nichtfunktionalen Anforderungen sind sie direkt in der Software zu finden, d.h. es wird kein Tooling beschrieben sondern was im Programmablauf zu beachten ist.

2.1.1 Genereller Ablauf

Ihrem Programm soll per Kommandozeilenparameter eine oder mehrere Inputdateien übergeben werden. Ihr Programm soll diese Inputdateien einlesen und basierend auf dem darin enthaltenen Text und Einstellungen Outputdateien (*.h und *.c/*.cpp) generieren. Die auf diese Weise generierten Dateien sollen die Textbausteine für eine Applikation liefern können.

2.1.2 Kommandozeilenoptionen

Option help

Ein Hilfetext soll dem Benutzer Ihres Programms erklären wie Ihr Programm aufgerufen werden kann und welche Parameter angegeben werden können.

Option inputfilename

Die Angabe der Inputdateien ist der wichtigste Parameter Ihres Programms. Es soll möglich sein sowohl einzelne als auch mehrere zu verarbeitende Dateien und den Pfad dorthin anzugeben (z.B. gentxtsrccode [options] INPUTFILE.TXT oder gentxtsrccode [options] ../Text/*.txt).

Option outputfilename

Über die Option Outputfilename kann der Benutzer den Namen der generierten Files bestimmen.

Option outputtype

Über die Option Outputtype kann der Benutzer zwischen den Optionen C und CPP wählen und so bestimmen welches Format der generierte Code haben soll.

Option headerdir

Der Pfad an welchem die generierte Headerdatei abgelegt wird.

Option sourcedir

Der Pfad an welchem die generierte Sourcedatei abgelegt wird.

Option namespace

Namespaces für C++ Dateien.

Option signperline

Anzahl der Zeichen pro Zeile in der generierten Variablen (Buchstaben/Steuerzeichen/Ersatzzeichen zwischen den Ausführungszeichen wobei bei Ersatzzeichen die einzelnen Buchstaben gezählt werden).

Option sortbyvarname

Aktivierung der alphabetisch aufsteigenden Sortierung der Variablen im Output.

2.1.3 Requirements Kommandozeilenoptionen

Um dem Nutzer die Steuerung Ihres Programms zu erleichtern soll das Programm ein mit getopt/getoptlong erstelltes Optionsparsing enthalten. Es gibt Optionen die nur als Schalter wirken (ein/aus), Optionen die zwingend weitere Argumente benötigen oder Optionen bei denen die weiteren Argumente optional sind. Können Optionen nicht geparst werden so handelt es sich hierbei um übergebene Dateien.

{ReqFunc1} Ihr Programm soll einen Hilfetext besitzen, welcher per Kommandozeilenparameter `-h` oder `--help` aufrufbar ist und die Benutzung Ihres Programms erklärt.

{ReqFunc2} Der Hilfetext soll sämtliche Kommandozeilenparameter Ihres Progamms und deren Benutzung erklären.

{ReqFunc3} Der Hilfetext soll Angaben zum Autorenteam enthalten: Alle Namen der Teammitglieder und zumindest eine Emailadresse unter der man das Autorenteam erreichen kann.

{ReqFunc4} Ihr Programm soll die Inputdatei als Kommandozeilenparameter ohne separate Option (kein `--input` etc.) verarbeiten können.

{ReqFunc5} Ihr Programm soll bei der Angabe mehrerer Inputdateien im gleichen Programmaufruf diese nacheinander verarbeiten können.

{ReqFunc6} Ist der Name der Outputfiles mit der Option outputfilename per Kommandozeile übergeben worden, dann sollen eine Header-Datei und eine Source-Datei mit diesem Namen erstellt werden.

{ReqFunc7} Ist der Dateityp der Outputfiles mit der Option outputtype per Kommandozeile übergeben worden, dann sollen der generierte Code im jeweiligen Dateityp erzeugt werden. Gültige Optionen sind C und CPP.

{ReqFunc8} Ist der Name des Headerverzeichnisses mit der Option headerdir per Kommandozeile übergeben worden, dann soll die generierte Header-Datei im Verzeichnis headerdir abgelegt werden. Falls das Verzeichnis nicht existiert, dann ist es anzulegen.

{ReqFunc9} Ist der Name des Sourceverzeichnisses mit der Option sourcedir per Kommandozeile übergeben worden, dann soll die generierte Source-Datei im Verzeichnis sourcedir abgelegt werden. Falls das Verzeichnis nicht existiert, dann ist es anzulegen.

{ReqFunc10} Ist der Namespace mit der Option namespace per Kommandozeile übergeben worden, dann soll in der Header-Datei und der Source-Datei dieser Namespace eingefügt werden.

{ReqFunc11} Ist die Anzahl der Zeichen pro Zeile mit der Option signperline X per Kommandozeile übergeben worden, dann soll im Text der generierten Variablen nach X Zeichen ein Zeilenumbruch eingefügt werden.

{ReqFunc12} Wenn sich eine Kommandozeilenoption und eine Option aus dem @global Tag widersprechen, dann soll die Kommandozeilenoption Prioriät gegenüber der @global-Option erhalten, um dem Benutzer die Möglichkeit des manuellen Overrides zu geben.

{ReqOptFunc1} Zu ihrem Basisprogramm soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc2} Zu ihrem Basisprogramm soll ein Unittest erstellt werden.

2.1.4 Input-Format

Das Format einer Inputdatei ist wie folgt vorgegeben (siehe Example.txt). Wenn die Datei nur reinen Text enthält soll der gesamte Text als eine einzige String-Variable abgelegt werden. Der Variablenname soll der Dateiname der Input-Datei ohne Extension sein. Durch verschiedene Parameter/Tags innerhalb der Inputdatei sollen unterschiedliche Optionen definiert werden können (siehe unten). Steuerinformation die in einer Textdatei enthalten ist wird zeilenweise ausgeführt, d.h. sie erstreckt sich nicht über mehrere Zeilen. Ist bei einem Tag ein oder mehrere Parameter enthalten so sind diese in einer Zeile zu halten. Die Parameter sind mit einer geschweiften Klammer umschlossen und im JSON Format ausgeführt. Diese Parameter können sowohl mit der bekannten Funktion srtcmp, wie auch mit xscanx Varianten wie auch Libraries JSONCPP und Boost-JSON ausgewertet werden.

{ReqFunc13} Wenn in der Input-Datei keine Tags vorhanden sind, dann soll der gesamte Text als eine einzige String-Variable abgelegt werden. Der Variablenname soll der Dateiname der Input-Datei ohne Extension sein.

{ReqFunc14} Ist der Name der Outputfiles NICHT mit der Option outputfilename per Kommandozeile übergeben worden und der outputfilename auch nicht in der Input-Datei über das Tag @global definiert, dann sollen eine Header-Datei und eine Source-Datei mit einem Default-Namen erstellt werden (z.B. inputdateiname.h und inputdateiname.cpp).

Format @start / @end

Die Tags @start und @end markieren einen zu verarbeitenden Abschnitt. Sie stehen immer alleine in ihrer jeweiligen Zeile. Text der sich außerhalb der beiden Tags befindet kann komplett ignoriert werden.

Format @global

Das Tag @global beinhaltet eine Reihe von Parametern mit deren Hilfe Einstellungen verändert werden können. Es handelt sich um die folgenden Parameter:

{ReqFunc15} Der Parameter signperline (int) aus dem Tag @global definiert die Anzahl der Zeichen pro Zeilen in der generierten Variablen. Der Text der Variablen soll nach der gegebenen Anzahl Zeichen umgebrochen werden. Der Defaultwert dieser Einstellung ist 60 Zeichen.

{ReqFunc16} Der Parameter sortbyvarname (true, false) aus dem Tag @global definiert die Sortierung der Variablen in den Outputfiles. Ist der Parameter auf true gesetzt dann sollen die Variablen alphabetisch aufsteigend sortiert werden. Der Defaultwert dieser Einstellung ist false.

{ReqFunc17} Der Parameter namespace (text) aus dem Tag @global kann einen Namespace definieren. Wenn der Namespace definiert ist und das Outputformat C++ ist, dann sollen die Variablen im definierten Namespace angelegt werden. Der Defaultwert dieser Einstellung ist kein Namespace.

{ReqFunc18} Der Parameter outputfilename (text) aus dem Tag @global definiert den Namen der generierten Dateien. Wenn der Name der generierten Dateien nicht über die Kommandozeile vorgegeben wurde und im @global Tag definiert wurde, dann sollen die Outputdateien mit diesem Namen erzeugt werden.

{ReqFunc19} Der Parameter outputtype (C, CPP) aus dem Tag @global definiert den Typ der generierten Dateien. Die Dateien sollen in der passenden Syntax generiert werden. Etwaige Einschränkungen der Sprachen (z.B. keine Namespaces in C Dateien) sind zu beachten.

{ReqFunc20} Der Parameter headerdir (text) aus dem Tag @global definiert das anzulegende Verzeichnis für die generierten Headerdateien. Die generierten Headerdateien sollen in diesem Verzeichnis abgelegt werden. Falls das Verzeichnis nicht existiert, dann ist es anzulegen. Der Defaultwert dieser Einstellung ist das Verzeichnis in dem die Applikation ausgeführt wird.

{ReqFunc21} Der Parameter sourcedir (text) aus dem Tag @global definiert das anzulegende Verzeichnis für die generierten Sourcedateien. Die generierten Sourcedateien sollen in diesem Verzeichnis abgelegt werden. Falls das Verzeichnis nicht existiert, dann ist es anzulegen. Der Defaultwert dieser Einstellung ist das Verzeichnis in dem die Applikation ausgeführt wird.

Format @variable / @endvariable

Die Tags @variable und @endvariable markieren Start und Ende des Texts einer Variablen. Eine Datei kann beliebig viele solcher Abschnitte enthalten. Sie dürfen sich untereinander nicht überschneiden oder verschachtelt sein. Text der sich nicht zwischen den Tags @variable und @endvariable befindet und auch sonst keinem Tag zugeordnet ist, soll als lokaler Kommentar interpretiert werden, der nicht in die Outputfiles überführt wird.

{ReqFunc22} Sollte eine Input-Datei fehlerhafte Variablenabschnitte enthalten, dann soll die Verarbeitung mit einer Fehlerausgabe abbrechen.

{ReqFunc23} Der Name der Variablen aus dem Parameter varname (text) aus dem Tag @variable soll im generierten Code übernommen werden.

{ReqFunc24} Wird kein Variablenname über das Tag gewählt, so ist ein geeigneter Default-Name zu verwenden (z.B.: %INPUTFILENAME%01 (Uppercase ohne Extention .txt, % steht für zu ersetzenden Namen %)). Bei mehreren Variablen ohne Namen aus dem selben File soll eine laufende Nummer vergeben werden.

{ReqFunc25} Der Inhalt der Variablen soll so codiert werden wie über den Parameter seq (ESC, HEX, OCT, RAWHEX) aus dem Tag @variable vorgegeben. ESC bedeutet normaler ASCII Text mit Escape-Sequenzen. HEX bedeutet hexadezimal codierte ASCII Zeichen als Escapesequenz (z.B.
x43 = C). OCT bedeutet Oktal codierte ASCII Zeichen als Escapesequenz (z.B. 153 = c) und RAWHEX bedeutet Hexwerte (z.B. 0x63 = c).

{ReqFunc26} Zeilenumbrüche sollen so gehandhabt werden wie über den Parameter nl (DOS, MAC, UNIX) aus dem Tag @variable vorgegeben (DOS = CR LF, MAC = CR, UNIX = LF). Der Defaultwert dieser Einstellung ist UNIX.

{ReqFunc27} Wenn der Parameter addtextpos (true, false) aus dem Tag @variable auf true gesetzt ist, dann soll die Zeilennummer der Variablen im Inputfile als Kommentar im Headerfile bei der Variablen mit angegeben werden. Der Defaultwert dieser Einstellung ist false.

{ReqFunc28} Wenn der Parameter addtextsegment (true, false) aus dem Tag @variable auf true gesetzt ist, dann soll der Originaltext der Variablen aus dem Inputfile im Sourcefile als Kommentar unterhalb der Variablendefinition ausgegeben werden. Der Defaultwert dieser Einstellung ist false.

{ReqFunc29} Der über den Parameter doxygen (text) aus dem Tag @variable definierte Text soll als Kommentar bei der Deklaration der Variablen im Headerfile ausgegeben werden. Der Defaultwert dieser Einstellung ist kein Kommentar.

2.1.5 Datenverwaltung

Um die einzelnen Ausgabeformate besser zu strukturieren zu können muss zu den einzelnen Wandlungsfunktionen noch eine Basiklasse erstellt werden. Diese wird dann später in die vier spezifischen Wandlungsklassen für Escape-, Oktal-, Hexadezimal- und roh Hexadezimal abgeleitet.

{ReqFunc30} Es soll eine Basisklasse CTextToCPP erstellt werden.

{ReqFunc31} Die Basisklasse CTextToCPP soll eine Datenverwaltung für den zu wandelnden Text enthalten.

{ReqFunc32} Die Basisklasse CTextToCPP soll die Verkettung von Instanzen von Basis- und abgeleiteten Klassen ermöglichen.

{ReqFunc33} Die Basisklasse CTextToCPP soll die Methode writeDeclaration enthalten, die den Inhalt einer Headerdatei generiert.

{ReqFunc34} Die Basisklasse CTextToCPP soll die Methode writeImplementation enthalten, die den Inhalt einer Sourcedatei generiert.

{ReqFunc35} Die Basisklasse CTextToCPP soll die Methode addElement enthalten, die ein weiteres Element an das Ende der Verkettung anfügt.

{ReqFunc36} Die Basisklasse CTextToCPP soll die Methode sort enthalten, die verkettete Elemente nach Alphabet aufsteigend sortiert.

{ReqFunc37} Die Basisklasse CTextToCPP soll die Methode clear enthalten, die alle Elemente löscht und den Speicher dieser Elemente danach wieder frei gibt.

{ReqOptFunc3} Zur Basisklasse CTextToCPP soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc4} Zur Basisklasse CTextToCPP soll ein Unittest erstellt werden.

{ReqFunc38} Erstellen Sie die Klasse CTextToEscSeq in der nicht darstellbare Zeichen mit alternativen Escape Sequenzen ersetzt werden (Tabelle Escape Sequenzen siehe Anhang).

{ReqFunc39} Leiten Sie die Klasse CTextToEscSeq von der Basisklasse CTextToCPP und implementieren Sie die Methoden der Basisklasse.

{ReqOptFunc5} Zur Klasse CTextToEscSeq soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc6} Zur Klasse CTextToEscSeq soll ein Unittest erstellt werden.

{ReqFunc40} Erstellen Sie die Klasse CtextToOctSeq in der alle Zeichen mit oktalen Escape Sequenzen (\000 ... \177) ersetzt werden.

{ReqFunc41} Leiten Sie die Klasse CtextToOctSeq von der Basisklasse CTextToCPP und implementieren Sie die Methoden der Basisklasse.

{ReqOptFunc7} Zur Klasse CtextToOctSeq soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc8} Zur Klasse CtextToOctSeq soll ein Unittest erstellt werden.

{ReqFunc42} Erstellen Sie die Klasse CtextToHexSeq in der alle Zeichen mit hexadezimalen Escape Sequenzen (\x00 ... \x7F) ersetzt werden.

{ReqFunc43} Leiten Sie die Klasse CtextToHexSeq von der Basisklasse CTextToCPP und implementieren Sie die Methoden der Basisklasse.

{ReqOptFunc9} Zur Klasse CtextToHexSeq soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc10} Zur Klasse CtextToHexSeq soll ein Unittest erstellt werden.

{ReqFunc44} Erstellen Sie die Klasse CTextToRawHexSeq in der alle Zeichen mit hexadezimalen Zeichen (0x00 ... 0x7F) ersetzt werden.

{ReqFunc45} Leiten Sie die Klasse CTextToRawHexSeq von der Basisklasse CTextToCPP und implementieren Sie die Methoden der Basisklasse.

{ReqOptFunc11} Zur Klasse CTextToRawHexSeq soll eine englischsprachige Dokumentation erstellt werden.

{ReqOptFunc12} Zur Klasse CTextToRawHexSeq soll ein Unittest erstellt werden.

{ReqFunc46} Wenn ein Zeichen nicht ASCII-darstellbar ist (Zeichen im Bereich (0x80 ... 0xFF)), dann soll das Programm abbrechen die fehlerhafte Zeile aus der Quelle melden.

{ReqFunc47} Für alle vier Klassen (CtextToEscSeq, CtextToOctSeq, CtextToHexSeq und CTextToRawHexSeq) gilt, dass am Ende der gewandelten Zeichenkette kein NewLine Zeichen eingefügt werden soll.

2.1.6 Codegenerierung

Die Aufgabe Ihres Programms ist es aus einer oder mehreren gegebenen Input-Dateien Output-Dateien in Form von Source- und Headerdateien zu erstellen. Beachten Sie dass bei Angabe von mehreren Input-Dateien natürlich auch mehrere outputfilenames über das @global Tag definiert sein können. Der Variablen-Text aus dem Inputfile ist in der Source-Datei (*.c/*.cpp) der Outputdateien in der gewünschten Codierung als Variable anzulegen, und die Variable ist in der Header-Datei (*.h) zu deklarieren.

Variablennamen

Der Variablenname ist dem @variable Tag zu entnehmen, bzw. zu generieren.

Dateinamen

Die Dateinamen der Header- und der Source Datei (Outputdateien) sollen aus der Kommandozeile oder den Tags in der Inputdatei entnommen werden. Ist keine Angabe gemacht worden, ist ein Default-Name zu wählen (Bsp.: inputfilename.h und inputfilename.cpp)

Header Datei

Um die Header Datei mehrfach includieren zu können ist eine Mehrfacheinbindung vorzusehen. Diese soll in die erzeugte Datei geschrieben werden, dass sie sowohl unter Windows MinGW wie auch unter UNIX GNU-C Compiler funktioniert. Das Kommando #pragma once ist nicht erwünscht.

2.1.7 Requirements Codegenerierung

{ReqFunc48} Jeder Textabschnitt der zur Variable verarbeitet werden soll, soll in der jeweiligen Source-Datei als Variable in der gewünschten Codierung definiert werden. Der Datentyp ist hierbei ein ASCII-Character String vom Type char mit konstantem Inhalt und konstanter Adresse.

{ReqFunc49} Jeder Textabschnitt der zur Variable verarbeitet werden soll, soll in der jeweiligen Header-Datei als Variable deklariert werden. Der Datentyp ist hierbei ein ASCII-Character String vom Type char mit konstantem Inhalt und konstanter Adresse und externer Implementierung.

{ReqFunc50} Die erzeugte Header Datei soll Mehrfacheinbindung unterstützen und die Mehrfacheinbindung der erzeugten Header Datei soll kompatibel zum Compiler Windows MinGW wie auch unter UNIX zu GNU-C sein.

{ReqFunc51} Die Includierung der generierten Header Datei in die generierte Sourcedatei soll so erfolgen, dass sie pfadunabhängig ist.

2.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen sind umzusetzen. Im Gegensatz zu funktionalen Anforderungen, sind nichtfunktionale Anforderungen nicht direkt an Features des Programms gebunden sondern beschreiben die Umgebung und das Tooling.

2.2.1 Tools und Build Umgebungen

Unterstütze Plattformen und Compiler

Um die Software auf den Systemen der DHBW Ravensburg Aussenstelle Friedrichshafen erstellen zu können ist zwingend erforderlich einen Compiler zu verwenden, der die gleichen Kommandozeilenparameter versteht und frei von Lizenzen ist. Die Abnahme der Software zur Benotung erfolgt in der DHBW Ravensburg Aussenstelle Friedrichshafen. Des weiteren ist auch MinGW64 ohne Windows spezifische Libraries zugelassen (kein mfc, kein windows.h). Sollten Zusatzlibraries zugelassen sein, so sind diese im Anhang aufgeführt.

{ReqNonFunc1} Die Software soll mit GNU-C/C++ für UNIX compilierbar sein.

{ReqNonFunc2} Die Software soll mit MinGW64 compilierbar sein.

Erstellung der ausführbaren Datei

Um die Software frei von graphischen Entwicklungsumgebungen erstellen zu können wird ein Makefile benötigt. Dieses kann dann unter mingw32-make (Windows) oder make (UNIX) erstellt werden. Da der Umfang dieses von Hand zu erstellen den Umfang des Programmentwurfs überschreiten würde, wird hier auf CMake als Erstellungstool für das Makefile verwiesen.

{ReqNonFunc3} Das Erstellen der Software soll mit CMake erfolgen

Dokumentation des Programmentwurf

Um den Dozenten Th. Staudacher und A. Maus die Chance zu geben, die Struktur und Funktionalität des Programmentwurf zu überblicken ist es wichtig, dass die Dokumentation des Codes im doxygen-Format erfolgt. Durch gute Dokumentation werden Rückfragen im Fehlerfall minimiert und es können ggf. auch besser Teilpunkte vergeben werden. Das Dokumentieren kann in deutsch oder englisch erfolgen.

{ReqOptFunc13} Die Dokumentation des Codes soll mit dem Syntax von doxygen erfolgen

Einlesen der Optionen

Um eine einheitliche Basis zu schaffen und das Programm portabel zu halten wird zum Parsen der Optionen die getopt(_long) Implementierung verwendet. Das implementieren anderer Libraries oder eigengeschriebene Anwendungen ist nicht gestattet.

{ReqNonFunc4} Das Parsen der Optionen soll über die LibC Funktionalität getopt(...) und getopt_long(...) aus getopt.h erfolgen

Dateien und Verzeichnisstruktur

Um eine saubere Trennung zwischen Implementierung und Deklarierung zu bekommen ist eine Auftrennung in zwei Verzeichnisse nötig. Unterverzeichnisse in dieser Auftrennung sind ebenfalls möglich.

{ReqNonFunc5} Die Header- und Source-Dateien ihres Programms sollen sich in verschiedenen Unterverzeichnissen befinden.

Programmierung

In dieser Sektion sind die Codierrichtlinien enthalten die dafür sorgen dass der Code der geforderten Qualität entspricht. Diese Codierrichtlinien sind unbedingt einzuhalten. Muss aus irgendeinem Grund davon abgewichen werden so ist dies in einem Kommentar zu begründen.

{ReqNonFunc6} Daten und Methoden die nur einer internen Berechnung dienen (nicht zum Export) sollen gegen externen Aufruf oder auslesen geschützt werden.

{ReqNonFunc7} Zeiger sollen immer initialisiert werden, Ausnahmen sind nur bei entsprechender Kommentierung gestattet.

- Der Gültigkeitsbereich einer Variablen soll beachtet werden.
- Werden logische Verknüpfungen gemacht so sind logische Operatoren zu verwenden.
- Berechnungen in logischen Verknüpfungen sind nicht gestattet.

{ReqNonFunc8} Werte die im weiteren Kontext nicht mehr verändert werden sollen, sind konstant zu halten.

{ReqNonFunc9} Zeiger die im weiteren Kontext nicht mehr verändert werden sollen, sind konstant zu halten.

- Das wegcasten von const ist nicht gestattet.
- Die Gültigkeit von Variablen soll an Übergabestellen überprüft werden.
- Jede Header- und Source Datei soll am Anfang ein Hinweis auf den Bearbeiter enthalten.
- Das Kommentieren des Codes soll in deutscher oder englischer Sprache erfolgen.
- Werden Variablen verwendet die eine fixe Bitlänge benötigen so soll die Definition aus cstdint verwendet werden.
- Berechnungen mit Fließkommazahlen sind nicht gestattet.
- Das Verwenden von goto ist nicht gestattet.

- Das Verwenden von #pragma ist nur gestattet solange dadurch keine Compilerabhängigkeit entsteht.
- Das verwenden des ? Operators ist gestattet
- Bei einer Kombination aus if/else if das kein else hat kann das else mit einem leeren Codeblock mit entsprechender Kommentierung hinzugefügt werden
- Das verwenden von mehrdimensionalen Arrays soll vermieden werden
- Soweit technisch möglich sollten die Header- und Source Dateien in utf-8 Zeichencodierung erstellt werden

2.3 Optionale Anforderungen

Optinale Requirements fliessen nur bei 5er Gruppen in die Bewertung ein. Es hat sich aber in den letzten Jahren gezeigt, dass eine saubere Entwicklungsumgebung viele Fehler vermeidet. Es ist angeraten einige der hier erwähnten Dinge mit umzusetzen.

2.3.1 Tools und Build Umgebungen

Libraries für C++

Es kann der ganze Umfang der mitgelieferten Headerdateien (soweit unter Windows und UNIX gleich einsetzbar) verwendet werden. Alternativ muss per Präprozessor jeweils eine Umschaltung erfolgen. Vordefinerte Macros sind hierbei __WIN32__, __UNIX__ etc. Dies sind dann ggf. in der Dokumentation das Compilers nachzuschlagen. Um die volle Funktionalität einer C++ Implementierung auspielen zu können, kann sowohl die STL wie auch Boost als Library verwendet werden wenn dies nicht durch ein Requirement ausdrücklich verboten wird.

Logging von Ereignissen

Um den Bildschirm nicht mit Meldungen zu überfrachten hat es sich eingebürgert, ein Logging zu verwenden. Das ursprünglich für Java entwickelte Log4J wurde in fast alle Programmiersprachen überführt. Im Unterricht wurden zumindestens eines der beiden Logging-Libraries Easylogging++ oder Boost Logging besprochen. Es ist dringend angeraten dieses auch im Programmentwurf ein zu setzen. Es kann auch helfen im Fehlerfall im Dialog mit den Studenten eine lauffähige Version nach Abgabe zu erzeugen, falls diese fehlschlägt.

- Das Logging der Zustände während des Programmablauf kann mit der Library Easylogging++ erfolgen
- Das Logging der Zustände während des Programmablauf kann mit der Library Boost Logging erfolgen

Formatieren der Ausgabe

Die Formatierung der Ausgabe ist kein Bestandteil des Programmentwurfs. Daher sollte keine Zeit in die Formatierung des Ergebnisses verwendet werden. Um aber eine bessere Übersicht im Arbeitsergebnis zu erhalten wäre eine Formatierung der Header- und Source Datei vorteilhaft. Hierbei bietet sich astyle an.

- Das Formatieren des erzeugten Codes kann mit Artistic Style erfolgen

Testen von Teilfunktionalitäten

Um die einzelnen Programmteile auf Funktionalität zu untersuchen ist es angeraten einen Unittest für die einzelnen Programmteile zu erstellen. Es hat sich aber in der Vergangenheit gezeigt, das sowohl die Studenten wie auch die Dozenten aus den Unittest nachvollziehen konnten wie das Programm funktioniert und häufig auch Fehler gefunden wurden. Zwei Libraries sind hier von Bedeutung, einmal die einfach zu implemntierende Catch2 die nur als reine Header Datei includiert werden kann sowie das etwas aufwendigere Framework von Boost Test. Der Vorteil von Catch2 ist die leichtere Implementierbarkeit, der Vorteil von Boost Test der bessere Funktionsumfang und die bessere Integrierbarkeit von C++.

{ReqOptFunc14} Das Testen der Module soll entweder mit der Library Catch2 oder mit der Library Boost Test erfolgen

2.4 Programmbeispiele

Hinweis: Alle Beispiele sind auf Moodle in einer ZIP Datei eingestellt.

2.4.1 Tabelle C-Escape Sequenzen

Escape Ersatz	Zahlenwert	Beschreibung
\a	0x07	Bell
\b	0x08	Backspace
\e	0x1B	Escape character
\f	0x0C	Formfeed Page Break
\n	0x0A	Newline (Line Feed)
\r	0x0D	Carriage Return
\t	0x09	Horizontal Tab
\v	0x0B	Vertical Tab
\\	0x5C	Backslash
\’	0x27	Apostrophe
\”	0x22	Double quotation mark
\?	0x3F	Question mark
\nnn	any	octal number
\xhh	any	hexadecimal number

2.4.2 Text Formatierung unter C++

Eine sehr gute Dokumentation der Formatierung unter ostream ist unter COUT Formatierung beschrieben.

Anmerkung: Zertifikat wird als nicht sicher angegeben. Es handelt sich hierbei um eine PDF.

2.4.3 Text Streamverarbeitung unter C++

Um Texte einfach zu bearbeiten bietet es sich an eine komplette Textdatei in einen istringstream zu überführen. Dieser kann dann so bearbeitet werden wie eine Datei, allerdings geht das setzen der Position einfacher wie in einem fstream.
Anmerkung: Ist der istrstream am Ende angekommen muss vor seekg() ein clear() durchgeführt werden.

2.4.4 Text Formatierung unter C++ analog printf

Sehr präzise kann Text unter C formatiert werden. Im Prinzip gibt es diese Optionen auch unter C++. Ein einfaches Mittel die C Formatierung von printf auch unter C++ zu übernehmen ist die Template Klasse von Boost Format. Der Syntax ist identisch zu printf und es gibt auch Erweiterungen.

```
//https://www.boost.org/doc/libs/1_75_0/libs/format/doc/format.html
#include <iostream>
using namespace std;

#include <boost/format.hpp>
using namespace boost;

int main() {
    cout << format("%-20s %s") % "Hello" % "World"  << endl;
    return 0;
}
```

2.4.5 String in Teilstrings zerlegen

Um Strings in Teilstrings zu zerlegen wird ein Tokenizer verwendet. Der unter C verwendete Tokenizer strtok() ist jedoch in mehrerer Hinsicht kritsich. Er nutzt statischen Speicher und ist nicht Threadfest. Ein Alternative hierfür ist der Tokenizer der in der Boost Library Tokenizer angeboten wird.

```
//https://www.boost.org/doc/libs/1_79_0/libs/tokenizer/doc/char_separator.
    htm
#include<iostream>
#include<boost/tokenizer.hpp>
#include<string>
```

```
using namespace std;
using namespace boost;

const string s = {"1,3,5,7,8,9"};

int main() {
    char_separator<char> sep(",");
    tokenizer<boost::char_separator<char> > tok(s, sep);
    for(auto it=tok.begin(); it!=tok.end(); ++it) {
        cout << *it << endl;
    }
    return 0;
}
```

2.4.6 Text in Zahlen wandeln

Wenn Texte in Zahlen gewandelt werden sollen wird meistens die Funktion atoi() und atod() verwendet.In der Boost Library LexicalCast gibt es spezialisierte Template Klasse die eine Wandlung durchführen können.

```
//https://theboostcpplibraries.com/boost.lexical_cast
#include<iostream>
#include <boost/lexical_cast.hpp>
#include<string>

using namespace std;
using namespace boost;

const string s = {"123"};

int main() {
    try
    {
        const int i = boost::lexical_cast<int>(s);
        cout << i << endl;
    }
    catch (const boost::bad_lexical_cast &e)
    {
        std::cerr << e.what() << '\n';
    }

    return 0;
}
```

2.4.7 Text in enum Werte wandeln und umgekehrt

Um einfach Enumerations von Text in enum und umgekehrt zu wandeln bietet sich Magic Enum C++ an.

Falls Sie dieses Paket verwenden wollen binden Sie bitte die Header Dateien in ein Unterverzeichnis ein das in Ihrem Projekt enthalten ist.

```
//g++ -o enum -I...Verzeichnis zu magic_enum... main.cpp --std=c++17
#include <magic_enum.hpp>

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

enum class EEnableDisable {
    DISABLED,
    ENABLED
};

int main() {
    const string input = {"ENABLED"};

    //Text wird in Enum Wert gewandelt
    auto e = magic_enum::enum_cast<EEnableDisable>(input);

    if (e.has_value())
        //Hier findet die Rueckwandlung in Text statt
        cout << magic_enum::enum_name(e.value()) << endl;

    return 0;
}
```

2.4.8 Optionale Shortopts

```
//Beispiel fuer ein optionales getopt shortopt
// -O  -> OK ... option '-O' without argument
// -O5 -> OK ... option -O with argument '5'
#include <stdio.h>
#include <getopt.h>
#include <ctype.h>

int main (int argc, char *argv[]) {
    int c;
    opterr = 0;
    while ((c = getopt (argc, argv, "O:")) != -1)
        switch (c)
        {
        case 'O':
            printf("OK ... option -%c with argument '%s'\n", c, optarg);
            break;

        case '?':
            if(optopt == 'O' && isprint(optopt))
            {
                printf("OK ... option '-O' without argument \n");
```

```
22                  break;
                }
24              else if (isprint (optopt))
                    fprintf (stderr, "ERR ... Unknown option `-%c'.\n", optopt)
    ;
26              else
                    fprintf (stderr, "ERR ... Unknown option character `\\x%x
    '.\n", optopt);
28              return -1;
            default:
30              ;
            }
32
            return 0;
34  }
```

2.4.9 JSON Library

Relativ einfach lassen sich die Parameter die bei dem Tag @lobal oder @variable übergeben werden mit einer JSON library wandeln. Erlaubt sind in dem Programmentwurf JSON-CPP und Boost JSON. Im Beispiel wird die Einbindung von JSONCPP gezeigt.
JSONCPP kann unter JSONCPP für Windows bezogen werden. Unter UNIX bitte mit dem Paket Manager installieren.
Um JSONCPP richtig hinzu zu linken im CMake File muss zwischen Windows und UNIX unterschieden werden.

```
...
2
IF (MINGW)
4   # Includieren von JSON-CPP
    # Tauschen wir als Dozent spaeter in Ihrer Source ab
6   include_directories(C:/wo auch immer/include)
    link_directories(C:/wo auch immer/jsoncpp/lib)
8 ELSE ()
    # Unter UNIX werden Sie automatisch gefunden
10 ENDIF ()

12 ...
IF (MINGW)
14    target_link_libraries(${PROJECT_NAME} ...weitere Libraries ... jsoncpp.
      dll)
ELSE (MINGW)
16    target_link_libraries(${PROJECT_NAME} ...weitere Libraries ... jsoncpp)
ENDIF ()
```

Einfache JSONCPP Beispielanwendungen gibt es hier.

2.4.10 Text in Linien fester Breite wandeln

Um Texte in eine definierte Lücke zu pressen muss der Text in Teillinien gewandelt werden. Das Verfahren heisst **Word Wrap**. Ein Beispiel wie so etwas geht wird in Implementing Word Wrap in C# beschrieben. Diese ist einfach in C überführbar, da die Funktionalität analog C# auch in C funktioniert. Der StringBuilder ist durch die std::strstream zu ersetzen. Die White Spaces können mit der Funktion isspace() aus der ctype Header Datei überprüft werden.

2.4.11 Unittest

Um Teileinheiten der Software (Klassen/Funktionen) zu testen ist ein Unittest das Mittel der Wahl. Gängige Unittestframeworks helfen hier den Test aufzusetzen. Dabei wird beim Erstellen des Programms ein kompletter Menüaufbau erstellt.

Catch2 Catch2 ist eine Bibliothek die aus einer Header Datei generiert wird. Dazu muss in eine Datei die main erstellt werden was sehr lange dauert. Die Tests wiederum werden schnell compiliert. Am besten wird der Test dann mit cmake erstellt, so das die main nur einmal erstellt werden muss.

```
#define CATCH_CONFIG_MAIN
#define CATCH_CONFIG_FAST_COMPILE
#include <catch.hpp>
//Statischer Teil der aus der Header Datei
//erstellt wird in eine Datei auslagern
//Das Object muss nur einmal erstellt werden
```

Der eigentliche Test wird in eine andere Datei verlagert.

```
#define CATCH_CONFIG_FAST_COMPILE
#include <catch.hpp>

bool isNotZero(const int value) {
    return value;
}

TEST_CASE("Catch2Demonstrator")
{
    SECTION("NotZeroTestCase")
    {
        CHECK(isNotZero(-1) == true);
        CHECK(isNotZero(1) == true);
        CHECK(isNotZero(0) == false);
    }
}
```

Boost Test Ähnlich verhält es sich mit dem Framework Boost Test. Hier ist jedoch noch entweder die statische Lib libboost_unit_test_framework.a oder die dynamische Lib libboost_unit_test_framework.so/libboost_unit_test_framework.dll dazu zu linken. Bei Verwendung einer dynamsichen Lib wird das Programm wesentlich kleiner, Funktional gibt es keinen Unterschied. Von der Funktionalität übertrifft Boost Test Catch2 bei weitem, jedoch für den Programmentwurf ist beides ausreichend gut

```
//g++ -o unittest unittest.cpp -lboost_unit_test_framework
#define BOOST_TEST_MODULE BoostTestDemonstrator
#include <boost/test/included/unit_test.hpp>

bool isNotZero(const int value) {
    return value;
}

BOOST_AUTO_TEST_CASE( NotZeroTestCase )
{
    BOOST_TEST(isNotZero(-1) == true);
    BOOST_TEST(isNotZero(1) == true);
    BOOST_TEST(isNotZero(0) == false);
}
```

Beispieldatei sample.txt Um eine Vorstellung zu bekommen wie ein Output aussehen könnte ist eine Beispiel Test Datei sample.txt auf Moodle hinterlegt worden.

```
Der Text der hier steht wird noch nicht verarbeitet da das Start Kommando
    noch kommt
Falls Steuerinformation vorhanden ist muss @start und @end als TAG am
    Zeilenanfang und ohne weiteren Text vorkommen
Die Information zwischen @start und @end wird ausgeschnitten


@start

Globale Einstellungen koennen auch zur besseren Uebersicht aufgespaltet
    werden

@global { "signperline": 60, "sortbyvarname": false }
@global { "namespace": "DHBW", "outputfilename": "sample", "outputtype": "
    CPP", "headerdir": ".\\", "sourcedir": ".\\" }

@variable { "varname": "PIZZA", "seq": "ESC", "nl": "UNIX", "addtextpos":
    true, "addtextsegment": false, "doxygen": "Pizza als Fast-Food wurde in
     den U.S.A. erfunden" }
In den USA sind zwei Typen weit verbreitet, "Chicago-style" und "New York-
    style" Pizza.
Waehrend die New Yorker Variante mit ihrem sehr duennen Boden der
    italienischen Variante aehnelt, steht die Variante aus Chicago Kopf:
    Der Teig bildet eine Schuesselform, wird mit Mozzarellascheiben
    ausgelegt und mit weiteren Zutaten gefuellt.
```

```
16 Zum Schluss wird das ganze von oben mit zerkleinerten Tomaten bestrichen
       und mit Parmesan und Oregano bestreut.
   @endvariable
18
   @variable { "varname": "QUICKBROWNFOX", "seq": "OCT", "nl": "DOS", "
       addtextpos": true, "addtextsegment": true,"doxygen": "Dieser Text wird
       mit ESC-Sequence gewandelt" }
20 The quick brown fox jumps over the Lazy dog ...
   Das ist eine typische Testsequence fuer Texte (und um auch eine '
       Datenstrecke' zu ueberpruefen).
22 @endvariable

24 @variable { "varname": "INTRODUCTION_CPP_AT_ATT", "seq": "HEX", "nl": "DOS"
       , "addtextpos": true, "addtextsegment": true, "doxygen": "C++ und seine
        Herkunft" }
   C++ ist eine von der ISO genormte Programmiersprache. Sie wurde ab 1979 von
        Bjarne Stroustrup bei AT&T als Erweiterung der Programmiersprache C
       entwickelt.
26 C++ ermoeglicht sowohl die effiziente und maschinennahe Programmierung als
       auch eine Programmierung auf hohem Abstraktionsniveau.
   Der Standard definiert auch eine Standardbibliothek, zu der verschiedene
       Implementierungen existieren.
28 @endvariable

30 @variable { "varname": "BRUSSELSRAWHEX", "seq": "RAWHEX", "nl": "MAC", "
       addtextpos": true, "addtextsegment": true, "doxygen": "Zeitungsartikel
       in rein Hexadezimal" }
   Bruessel Europa bleibt voraussichtlich eine Rezession erspart, trotz
       Ukrainekrieg und Energiekrise. Allerdings droht der Kontinent
       gegenueber den dynamischeren USA und China weiter zurueckzufallen. Ein
       wichtiger Grund dafuer ist die schleppende wirtschaftliche Entwicklung
       in Deutschland.
32 Die EU-Kommission hat ihre Wachstumsprognose fuer das laufende und das
       kommende Jahr am Montag leicht angehoben. Die Bruesseler Behoerde
       rechnet mit einem Plus von 1,0 Prozent fuer 2023 und 1,7 Prozent fuer
       2024. Die europaeische Wirtschaft sei in besserer Verfassung als noch
       im Herbst angenommen, sagte EU-Wirtschaftskommissar Paolo Gentiloni.
   @endvariable
34 @end

36 Hier ist bereits Ende, weiterer Text wird nicht geparst
   Optionen aus der Kommandozeile haben hoehere Prio als der Tag Global
```

Deklarierung in der Header Datei

```
   #ifndef _SAMPLE_
2  #define _SAMPLE_

4  namespace DHBW {
   /** Pizza als Fast-Food wurde in den U.S.A. erfunden */
6  extern const char * const PIZZA;
   /** Dieser Text wird mit ESC-Sequence gewandelt */
8  extern const char * const QUICKBROWNFOX;
```

```
/** C++ und seine Herkunft */
extern const char * const INTRODUCTION_CPP_AT_ATT;
/** Zeitungsartikel in rein Hexadezimal */
extern const char BRUSSELSRAWHEX[];
}
#endif
```

Implementierung in der Source Datei

```
#include <sample.cpp>

namespace DHBW {
const char * const PIZZA = {
"In den USA sind zwei Typen weit verbreitet, \"Chicago-style\"" \
" und \"New York-style\" Pizza.\n" \
"Waehrend die New Yorker Variante mit ihrem sehr duennen Bode" \
"n der italienischen Variante aehnelt, steht die Variante aus" \
" Chicago Kopf: Der Teig bildet eine Schuesselform, wird mit " \
"Mozzarellascheiben ausgelegt und mit weiteren Zutaten gefuel" \
"lt.\n" \
"Zum Schluss wird das ganze von oben mit zerkleinerten Tomate" \
"n bestrichen und mit Parmesan und Oregano bestreut."};

const char * const QUICKBROWNFOX = {
"\124\150\145\040\161\165\151\143\153\040\142\162\157\167\156" \
"\040\146\157\170\040\152\165\155\160\163\040\157\166\145\162" \
"\040\164\150\145\040\114\141\172\171\040\144\157\147\040\056" \
"\056\056\015\012" \
"\104\141\163\040\151\163\164\040\145\151\156\145\040\164\171" \
"\160\151\163\143\150\145\040\124\145\163\164\163\145\161\165" \
"\145\156\143\145\040\146\165\145\162\040\124\145\170\164\145" \
"\040\050\165\156\144\040\165\155\040\141\165\143\150\040\145" \
"\151\156\145\040\047\104\141\164\145\156\163\164\162\145\143" \
"\153\145\047\040\172\165\040\165\145\142\145\162\160\162\165" \
"\145\146\145\156\051\056"};

/*
Originaltext aus der Variablensektion 'QUICKBROWNFOX'

The quick brown fox jumps over the Lazy dog ...
Das ist eine typische Testsequence fuer Texte (und um auch eine '
    Datenstrecke' zu ueberpruefen).
*/

const char * const INTRODUCTION_CPP_AT_ATT = {
"\x43\x2b\x2b\x20\x69\x73\x74\x20\x65\x69\x6e\x65\x20\x76\x6f" \
"\x6e\x20\x64\x65\x72\x20\x49\x53\x4f\x20\x67\x65\x6e\x6f\x72" \
"\x6d\x74\x65\x20\x50\x72\x6f\x67\x72\x61\x6d\x6d\x69\x65\x72" \
"\x73\x70\x72\x61\x63\x68\x65\x2e\x20\x53\x69\x65\x20\x77\x75" \
"\x72\x64\x65\x20\x61\x62\x20\x31\x39\x37\x39\x20\x76\x6f\x6e" \
"\x20\x42\x6a\x61\x72\x6e\x65\x20\x53\x74\x72\x6f\x75\x73\x74" \
"\x72\x75\x70\x20\x62\x65\x69\x20\x41\x54\x26\x54\x20\x61\x6c" \
"\x73\x20\x45\x72\x77\x65\x69\x74\x65\x72\x75\x6e\x67\x20\x64" \
```

```
"\x65\x72\x20\x50\x72\x6f\x67\x72\x61\x6d\x6d\x69\x65\x72\x73" \
"\x70\x72\x61\x63\x68\x65\x20\x43\x20\x65\x6e\x74\x77\x69\x63" \
"\x6b\x65\x6c\x74\x2e\x0d\x0a" \
"\x43\x2b\x2b\x20\x65\x72\x6d\x6f\x65\x67\x6c\x69\x63\x68\x74" \
"\x20\x73\x6f\x77\x6f\x68\x6c\x20\x64\x69\x65\x20\x65\x66\x66" \
"\x69\x7a\x69\x65\x6e\x74\x65\x20\x75\x6e\x64\x20\x6d\x61\x73" \
"\x63\x68\x69\x6e\x65\x6e\x6e\x61\x68\x65\x20\x50\x72\x6f\x67" \
"\x72\x61\x6d\x6d\x69\x65\x72\x75\x6e\x67\x20\x61\x6c\x73\x20" \
"\x61\x75\x63\x68\x20\x65\x69\x6e\x65\x20\x50\x72\x6f\x67\x72" \
"\x61\x6d\x6d\x69\x65\x72\x75\x6e\x67\x20\x61\x75\x66\x20\x68" \
"\x6f\x68\x65\x6d\x20\x41\x62\x73\x74\x72\x61\x6b\x74\x69\x6f" \
"\x6e\x73\x6e\x69\x76\x65\x61\x75\x2e\x0d\x0a" \
"\x44\x65\x72\x20\x53\x74\x61\x6e\x64\x61\x72\x64\x20\x64\x65" \
"\x66\x69\x6e\x69\x65\x72\x74\x20\x61\x75\x63\x68\x20\x65\x69" \
"\x6e\x65\x20\x53\x74\x61\x6e\x64\x61\x72\x64\x62\x69\x62\x6c" \
"\x69\x6f\x74\x68\x65\x6b\x2c\x20\x7a\x75\x20\x64\x65\x72\x20" \
"\x76\x65\x72\x73\x63\x68\x69\x65\x64\x65\x6e\x65\x20\x49\x6d" \
"\x70\x6c\x65\x6d\x65\x6e\x74\x69\x65\x72\x75\x6e\x67\x65\x6e" \
"\x20\x65\x78\x69\x73\x74\x69\x65\x72\x65\x6e\x2e"};

/*
Originaltext aus der Variablensektion 'INTRODUCTION_CPP_AT_ATT'

C++ ist eine von der ISO genormte Programmiersprache. Sie wurde ab 1979 von
     Bjarne Stroustrup bei AT&T als Erweiterung der Programmiersprache C
    entwickelt.
C++ ermoeglicht sowohl die effiziente und maschinennahe Programmierung als
    auch eine Programmierung auf hohem Abstraktionsniveau.
Der Standard definiert auch eine Standardbibliothek, zu der verschiedene
    Implementierungen existieren.
*/

const char BRUSSELSRAWHEX[] = {
 0x42, 0x72, 0x75, 0x65, 0x73, 0x73, 0x65, 0x6c, 0x20, 0x45, \
 0x75, 0x72, 0x6f, 0x70, 0x61, 0x20, 0x62, 0x6c, 0x65, 0x69, \
 0x62, 0x74, 0x20, 0x76, 0x6f, 0x72, 0x61, 0x75, 0x73, 0x73, \
 0x69, 0x63, 0x68, 0x74, 0x6c, 0x69, 0x63, 0x68, 0x20, 0x65, \
 0x69, 0x6e, 0x65, 0x20, 0x52, 0x65, 0x7a, 0x65, 0x73, 0x73, \
 0x69, 0x6f, 0x6e, 0x20, 0x65, 0x72, 0x73, 0x70, 0x61, 0x72, \
 0x74, 0x2c, 0x20, 0x74, 0x72, 0x6f, 0x74, 0x7a, 0x20, 0x55, \
 0x6b, 0x72, 0x61, 0x69, 0x6e, 0x65, 0x6b, 0x72, 0x69, 0x65, \
 0x67, 0x20, 0x75, 0x6e, 0x64, 0x20, 0x45, 0x6e, 0x65, 0x72, \
 0x67, 0x69, 0x65, 0x6b, 0x72, 0x69, 0x73, 0x65, 0x2e, 0x20, \
 0x41, 0x6c, 0x6c, 0x65, 0x72, 0x64, 0x69, 0x6e, 0x67, 0x73, \
 0x20, 0x64, 0x72, 0x6f, 0x68, 0x74, 0x20, 0x64, 0x65, 0x72, \
 0x20, 0x4b, 0x6f, 0x6e, 0x74, 0x69, 0x6e, 0x65, 0x6e, 0x74, \
 0x20, 0x67, 0x65, 0x67, 0x65, 0x6e, 0x75, 0x65, 0x62, 0x65, \
 0x72, 0x20, 0x64, 0x65, 0x6e, 0x20, 0x64, 0x79, 0x6e, 0x61, \
 0x6d, 0x69, 0x73, 0x63, 0x68, 0x65, 0x72, 0x65, 0x6e, 0x20, \
 0x55, 0x53, 0x41, 0x20, 0x75, 0x6e, 0x64, 0x20, 0x43, 0x68, \
 0x69, 0x6e, 0x61, 0x20, 0x77, 0x65, 0x69, 0x74, 0x65, 0x72, \
 0x20, 0x7a, 0x75, 0x72, 0x75, 0x65, 0x63, 0x6b, 0x7a, 0x75, \
 0x66, 0x61, 0x6c, 0x6c, 0x65, 0x6e, 0x2e, 0x20, 0x45, 0x69, \
 0x6e, 0x20, 0x77, 0x69, 0x63, 0x68, 0x74, 0x69, 0x67, 0x65, \
```

```
94  0x72, 0x20, 0x47, 0x72, 0x75, 0x6e, 0x64, 0x20, 0x64, 0x61, \
    0x66, 0x75, 0x65, 0x72, 0x20, 0x69, 0x73, 0x74, 0x20, 0x64, \
96  0x69, 0x65, 0x20, 0x73, 0x63, 0x68, 0x6c, 0x65, 0x70, 0x70, \
    0x65, 0x6e, 0x64, 0x65, 0x20, 0x77, 0x69, 0x72, 0x74, 0x73, \
98  0x63, 0x68, 0x61, 0x66, 0x74, 0x6c, 0x69, 0x63, 0x68, 0x65, \
    0x20, 0x45, 0x6e, 0x74, 0x77, 0x69, 0x63, 0x6b, 0x6c, 0x75, \
100 0x6e, 0x67, 0x20, 0x69, 0x6e, 0x20, 0x44, 0x65, 0x75, 0x74, \
    0x73, 0x63, 0x68, 0x6c, 0x61, 0x6e, 0x64, 0x2e, 0x0d, \
102 0x44, 0x69, 0x65, 0x20, 0x45, 0x55, 0x2d, 0x4b, 0x6f, 0x6d, \
    0x6d, 0x69, 0x73, 0x73, 0x69, 0x6f, 0x6e, 0x20, 0x68, 0x61, \
104 0x74, 0x20, 0x69, 0x68, 0x72, 0x65, 0x20, 0x57, 0x61, 0x63, \
    0x68, 0x73, 0x74, 0x75, 0x6d, 0x73, 0x70, 0x72, 0x6f, 0x67, \
106 0x6e, 0x6f, 0x73, 0x65, 0x20, 0x66, 0x75, 0x65, 0x72, 0x20, \
    0x64, 0x61, 0x73, 0x20, 0x6c, 0x61, 0x75, 0x66, 0x65, 0x6e, \
108 0x64, 0x65, 0x20, 0x75, 0x6e, 0x64, 0x20, 0x64, 0x61, 0x73, \
    0x20, 0x6b, 0x6f, 0x6d, 0x6d, 0x65, 0x6e, 0x64, 0x65, 0x20, \
110 0x4a, 0x61, 0x68, 0x72, 0x20, 0x61, 0x6d, 0x20, 0x4d, 0x6f, \
    0x6e, 0x74, 0x61, 0x67, 0x20, 0x6c, 0x65, 0x69, 0x63, 0x68, \
112 0x74, 0x20, 0x61, 0x6e, 0x67, 0x65, 0x68, 0x6f, 0x62, 0x65, \
    0x6e, 0x2e, 0x20, 0x44, 0x69, 0x65, 0x20, 0x42, 0x72, 0x75, \
114 0x65, 0x73, 0x73, 0x65, 0x6c, 0x65, 0x72, 0x20, 0x42, 0x65, \
    0x68, 0x6f, 0x65, 0x72, 0x64, 0x65, 0x20, 0x72, 0x65, 0x63, \
116 0x68, 0x6e, 0x65, 0x74, 0x20, 0x6d, 0x69, 0x74, 0x20, 0x65, \
    0x69, 0x6e, 0x65, 0x6d, 0x20, 0x50, 0x6c, 0x75, 0x73, 0x20, \
118 0x76, 0x6f, 0x6e, 0x20, 0x31, 0x2c, 0x30, 0x20, 0x50, 0x72, \
    0x6f, 0x7a, 0x65, 0x6e, 0x74, 0x20, 0x66, 0x75, 0x65, 0x72, \
120 0x20, 0x32, 0x30, 0x32, 0x33, 0x20, 0x75, 0x6e, 0x64, 0x20, \
    0x31, 0x2c, 0x37, 0x20, 0x50, 0x72, 0x6f, 0x7a, 0x65, 0x6e, \
122 0x74, 0x20, 0x66, 0x75, 0x65, 0x72, 0x20, 0x32, 0x30, 0x32, \
    0x34, 0x2e, 0x20, 0x44, 0x69, 0x65, 0x20, 0x65, 0x75, 0x72, \
124 0x6f, 0x70, 0x61, 0x65, 0x69, 0x73, 0x63, 0x68, 0x65, 0x20, \
    0x57, 0x69, 0x72, 0x74, 0x73, 0x63, 0x68, 0x61, 0x66, 0x74, \
126 0x20, 0x73, 0x65, 0x69, 0x20, 0x69, 0x6e, 0x20, 0x62, 0x65, \
    0x73, 0x73, 0x65, 0x72, 0x65, 0x72, 0x20, 0x56, 0x65, 0x72, \
128 0x66, 0x61, 0x73, 0x73, 0x75, 0x6e, 0x67, 0x20, 0x61, 0x6c, \
    0x73, 0x20, 0x6e, 0x6f, 0x63, 0x68, 0x20, 0x69, 0x6d, 0x20, \
130 0x48, 0x65, 0x72, 0x62, 0x73, 0x74, 0x20, 0x61, 0x6e, 0x67, \
    0x65, 0x6e, 0x6f, 0x6d, 0x6d, 0x65, 0x6e, 0x2c, 0x20, 0x73, \
132 0x61, 0x67, 0x74, 0x65, 0x20, 0x45, 0x55, 0x2d, 0x57, 0x69, \
    0x72, 0x74, 0x73, 0x63, 0x68, 0x61, 0x66, 0x74, 0x73, 0x6b, \
134 0x6f, 0x6d, 0x6d, 0x69, 0x73, 0x73, 0x61, 0x72, 0x20, 0x50, \
    0x61, 0x6f, 0x6c, 0x6f, 0x20, 0x47, 0x65, 0x6e, 0x74, 0x69, \
136 0x6c, 0x6f, 0x6e, 0x69, 0x2e, 0x00 };

138 /*
    Originaltext aus der Variablensektion 'BRUSSELSRAWHEX'
140

    Bruessel Europa bleibt voraussichtlich eine Rezession erspart, trotz
        Ukrainekrieg und Energiekrise. Allerdings droht der Kontinent
        gegenueber den dynamischeren USA und China weiter zurueckzufallen. Ein
        wichtiger Grund dafuer ist die schleppende wirtschaftliche Entwicklung
        in Deutschland.
142 Die EU-Kommission hat ihre Wachstumsprognose fuer das laufende und das
        kommende Jahr am Montag leicht angehoben. Die Bruesseler Behoerde
```

```
    rechnet mit einem Plus von 1,0 Prozent fuer 2023 und 1,7 Prozent fuer
    2024. Die europaeische Wirtschaft sei in besserer Verfassung als noch
    im Herbst angenommen, sagte EU-Wirtschaftskommissar Paolo Gentiloni.
*/
}
```