0.1 Konsoleneingabe und Ausgabe

Im Ordner `Code` findest du die Quelldateien `Exercise0_Name_and_Age.c` und `Exercise0_Name_and_Age.cpp`. Beide Quelldateien beinhalten Code, um von einem Nutzer den Vor- und Nachnamen einzulesen und danach wieder auszugeben. Die C-Quelldatei bedient sich dabei ausschließlich Funktionalitäten, die durch C zur Verfügung gestellt werden. Die C++-Datei löst die Aufgabe durch Verwendung der Standard-Bibliothek.

1. Zur C-Variante:
   a) Wir betrachten zuerst die C-Lösung. Erstelle dazu ein neues Projekt mit der C-Quelldatei als Main-Datei. Führe das Projekt aus und gib deinen Namen ein.
   b) Welche Nutzereingabe führt dazu, dass etwas anderes ausgegeben wird als eingegeben wurde? Verändere dazu nicht den Code, probiere aber gerne verschiedene Eingaben aus.
   c) Der Nutzer soll nun auch sein Alter direkt nach seinem Namen angeben können. Passe das `scanf_s`-Statement entsprechend so an, dass am Schluss ein ganzzahliger Wert eingegeben werden kann. Füge den `printf`-Statements sinnvolle Ausgaben hinzu.
2. Zur C++-Variante:
   a) Öffne die C++-Quelldatei, finde heraus, warum sie sich noch nicht starten lässt und behebe den Fehler.
   b) Gib nun erneut die Eingabe von Teilaufgabe 1.b ein und schaue, ob es zum selben Problem kommt.
   c) Führe Teilaufgabe 1.c auch für die C++ Version durch.
   d) Ermögliche dem Nutzer, sein Alter als Gleitkommazahl einzugeben, für den Fall, dass er sehr stolz auf seine fünfeinhalb Jahre ist.

0.2 Compiler Fehler

Gegeben sei der folgende Code:

```
#include <iostream.h>

int Main()
{
    std::STRING s = "Servus Welt! ";
    std::cOut << S << '\n';
}
```

a) Welche Fehler fallen dir in dem gegebenen Codeabschnitt auf?

b) Compiliere den Code und untersuche die Fehlermeldungen. Hast du alle Fehler gefunden, die der Compiler gefunden hat? Hat der Compiler alle gefunden, die du gefunden hast? Welche Hinweise gibt der Compiler auf die gefundenen Fehler?

c) Behebe die vom Compiler angegebenen Fehler einem nach dem anderen und compiliere den Code nach jedem behobenen Fehler. Untersuche, wie sich die Fehlermeldungen ändern.

0.3 Typ-Deduktion

In den folgenden Teilaufgaben sind Deklarationen gegeben, in denen eine automatische Typ-Auflösung stattfindet. Gib an, welcher Datentyp der Variable jeder Aufgabe durch den Compiler zugewiesen wird.

a) `auto a{ true };`

b) `auto b('x');`

c) `auto c{"Hello"};`

d) `auto d{"Hello"s};`

e) `auto e = &a;`

f) `auto& f = a;`

g) `auto g{1.5};`

h) `decltype(a) h;`

i) `decltype(g) i{'x'};`

j) `decltype(b + g) j;`

k) `auto k{new uint32_t(5)};`

l) `auto l = *k;`

m) `auto& m = k;`

n) `auto n = &k;`

o) `static auto o{3.2f};`

p) `auto p{o};`

q) `decltype(o)* q;`

r) `auto r = new char[3];`

s) `auto& s{ r[2] };`

t) `const auto t{5};`

u) `auto const u{5};`

v) `auto v = &t;`

w) `const auto w = &t;`

x) `auto const x = &t;`

y) `const auto y{new int(3.0)};`

z) `auto z{new const int(3.0)};`

0.4 Geldumrechnung

Ein US-Amerikaner beauftragt dich, sein Reisegeld in Euro umzurechnen. Um ihn bestmöglich zu unterstützen, bietest du ihm an ein Programm zu schreiben, bei dem er die Anzahl seiner „pennies“ (1-Cent Stücke), „nickels“ (5-Cent Stücke), „dimes“ (10-Cent Stücke), „quarters“ (25-Cent Stücke), „half dollars“ (50-Cent Stücke) und „dollars“ (100-Cent Stücke) eingeben kann.

a) Erstelle das Programm und schreibe eine eigene Abfrage für jedes Geldstück. z.B.: „How many pennies do you have?“

b) Lass das Programm seine aufgelisteten Geldstücke nach der kompletten Eingabe ausgeben. Achte darauf, dass das Programm auf die Grammatik achtet und bei einem Geldstück auch nur den Singular ausgibt. z.B.: „You have 1 dime“ statt „You have 1 dimes“.

c) Gib die Summe seines Reisegeldes in US-Dollar oder US-Cent aus.

d) Rechne den Geldbetrag mit dem aktuellen Wechselkurs in Euro um und gib den Betrag aus.

e) Nun möchte der US-Amerikaner seine Reise in weitere Länder fortsetzen. Füge dem Programm eine Eingabe für das Land hinzu, in welches er reisen möchte, und gib seinen Geldbetrag in der örtlichen Währung aus. Als Länder sollten zugelassen sein: England, Deutschland, Südafrika und Japan. Nutze dafür `IF`-Statements.

f) Verwende statt `IF`-Statements nun `Switch`-Statements und lasse zusätzlich die folgenden Länder als mögliche Eingabe zu: Schweiz, Frankreich und Neuseeland. *Freiwillig*: Schweden, Russland, Australien, Brasilien.

0.5 Pointer, Referenzen, Array, Struct, Enum

Die Quelldatei `Exercise0_Array.cpp` enthält ein Programm, welches die Elemente von zwei Integer-Arrays vergleicht. Dazu gibt es die Funktionen `compReference()` und `compCopy()`, welche die Werte der Elemente vergleichen. Die Funktion `compPointer()` vergleicht hingegen die Adressen der Elemente.

a) Versuche, den Inhalt der `main`-Funktion zu verstehen und führe den Code aus.

b) Ändere bei der Ausgabe der Werte und Vergleiche den Zugriff auf die Elemente, indem du die äquivalente Indizierung verwendest. D.h. die Schreibweise mit `[]`.

> *Tipp:* Ein Array ist ein Pointer auf das erste Element des Arrays im Speicher. Das bedeutet, dass mit der Schreibweise `*(a1)` der Wert des ersten Elements des Arrays `a1` ausgegeben wird. Die Indizierung `a1[0]` bedeutet, dass vom Pointer auf das erste Element nicht weitergegangen werden muss und dieser Wert ausgegeben werden soll. Ein Array gewährleistet, dass alle weiteren Elemente des Arrays in aufeinanderfolgenden Positionen im Speicher hinterlegt sind. Dadurch kann man die Adresse des Pointers auf das erste Element erhöhen, um einen Pointer auf ein späteres Element zu erhalten. Entsprechend ist `*(a1 + 5)` dasselbe wie `a1[5]`.

c) In der Datei sind ein Enum und ein Struct vorbereitet, welche ermöglichen sollen, Mitarbeitende eines Unternehmens zu verwalten und miteinander zu vergleichen. Erstelle einen Mitarbeitenden mit dem Namen „Hans Wurst“, der als „Intermediate“ bereits seit 5 Jahren im Unternehmen angestellt ist. Schreibe außerdem eine geeignete Funktion namens `printEmployee`, mit der Du die Informationen eines `Employee` geeignet ausgibst.

d) Erzeuge ein Array mit mindestens drei Mitarbeitenden und schreibe die Funktion `printEmployeeArray`, die alle Mitarbeitenden des Arrays ausgibt.

0.6 Binärzahlen

Erstelle ein Programm, bei dem ein Nutzer zwei bis zu 15-stellige Binärzahlen eingeben kann. Gib nach jeder Eingabe die entsprechende Dezimalzahl aus. Gehe davon aus, dass der Nutzer immer eine korrekte Eingabe tätigt. Berechne aus diesen beiden Binärzahlen die Summe sowie das Ergebnis des binären OR, AND und XOR und gib die Resultate wieder als Binärzahlen auf der Konsole aus.

Tipp 1: Verwende zur Konsolenausgabe der Binärzahl die Funktion aus der Vorlesung (Kapitel: Grundlagen → Arithmetische Operatoren → Schiebeoperatoren).
Tipp 2: In der Datei `Exercise0_Binary.cpp` findest du ein Programm, welches die Quersumme einer eingegebenen Binärzahl berechnet.
Tipp 3: Bevor du mit dem Programmieren beginnst, überlege dir zuerst, welche Ergebnisse du für die Eingaben 0 und 0, 0 und 1, 1 und 1, 11111 und 11111, sowie 101010 und 010101 erwarten würdest.

a) Teste dein Programm mit den Eingaben 0 und 0, 0 und 1, 1 und 1 sowie 111111111111111 und 111111111111111 (je 15 mal die 1). Ist die Ausgabe deines Programms plausibel?

b) Teste dein Programm mit den Eingaben 11111 und 11111, sowie 101010 und 010101.

c) Welche Ausgabe entsteht, wenn je 16 Einsen eingegeben werden? Stimmt die Ausgabe für alle Operationen?

d) Welche Ausgabe erzeugt dein Programm bei den Eingaben 42 und 12? Warum entsteht diese Ausgabe?

e) Welches Verhalten erwartest du bei den Eingaben $(3 + 4)$ und *Hallo*? Entspricht das Verhalten deines Programms deiner Erwartung?

f) Verbessere dein Programm so, dass nur gültige Binärzahlen akzeptiert werden. Bei einer ungültigen Eingabe soll der Nutzer darauf hingewiesen werden, was eine Binärzahl ist und eine neue Eingabe tätigen.