Datentypen

Agenda

- Standard Datentypen und Literale
- Konstanten
- Type-Casts
- Enums
- Braced Initializer
- „Auto“ Typen Erkennung
- Copy und move
- volatile, Nullpointer,

Standard Datentypen

- Welche Probleme macht short, int, long?
  - Datengröße Compiler- und Plattformabhängig
  - Programm kann sich dadurch auf verschiedenen Plattformen anders verhalten
- Lösung in C++11
  - Fixed width integer types
  - int16_t ist zum Beispiel garantiert 16 Bit groß, wenn erforderlich
  - Vorsicht mit (u)int_fast_t: Weitet Datentyp immer auf Cacheline Größe aus
  - https://godbolt.org/g/h1BdU3
  - Ähnliches Existiert auch für float und double:
    siehe http://en.cppreference.com/w/c/numeric/math/float_t
- Versuche immer zu initialisieren
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rc-initialize

Standard Datentypen

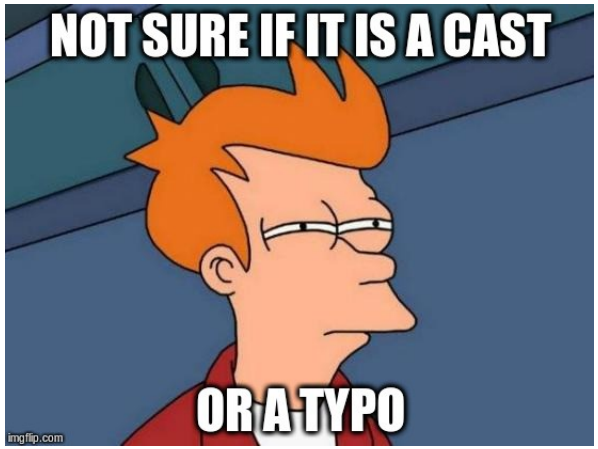
- Integer-Literale
  - Angabe des Integertypes auf der rechten Seite
  - Schon seit C++98 dabei
  - Angabe für Bytes seit C++14
  - https://godbolt.org/g/4BZL3Z
- User-defined literales
  - Angabe von eigenen Datengrößen
  - Syntax mal wieder furchtbar
  - Kann auch Funktionen beinhalten
  - https://godbolt.org/g/Tps1jt

Konstanten

- Definition mit const (bereits aus C bekannt)
  - **NIEMALS** mit #define → keine Typensicherheit
- Alles was Konstant ist, unbedingt mit const angeben
  - Damit weiß der Compiler, was sich ändern kann und was nicht
  - Damit weiß ein anderer Programmierer was, was sich ändern kann und was nicht
- Syntax mit Pointern nicht intuitiv
  - https://godbolt.org/g/Bc7AbB
  - Referenzen verwenden !!!

Type-Casts

- C Casts haben viele Probleme
  - Unklar was man Castet (nur const weg, int→ float, int→ ASCII zeichen)
  - Unklar wann man Castet (Compilezeit, Laufzeit)
  - Unklar welche Weg der Cast nimmt
- C++ Casts versuchen das besser
  - const_cast : const „wegcasten“ zur Compilezeit
  - static_cast : expliziter Compilezeit Cast
  - dynamic_cast: Laufzeit Polymorphie Cast (für Klassen)
  - reinterpret_cast: Laufzeit Neuinterpretierung der Daten
  - Warnung für C-Casts: -Wold-style-cast
  - https://godbolt.org/g/erVLBk
- Versuche Casts zu vermeiden!!
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es48-avoid-casts
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es49-if-you-must-use-a-cast-use-a-named-cast



Enum

- Enums sind für implizite Casts extrem Problematisch
  - https://godbolt.org/g/A5YbfB
- Lösung in C++ 11: enum classes (scoped enums)
  - Keine impliziten Casts
  - Klar definierter Typ
  - https://godbolt.org/g/nz7kMQ

Braced Initializer

- Initialisierung mit „ = “ erlaubt implizite Konvertierung
- Seit C++ 11 gibt es sogenannte Braced Initializer
  - Int i{10};
  - Ungewohnte Syntax
  - Aber keine Konvertierung erlaubt
  - Bei i = {10} wird eine Initialisierungsliste erstellt !!!!
  - https://godbolt.org/g/eA3Na7
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es23-prefer-the--initializer-syntax

„Auto“ Typen Erkennung

- Mittels „auto“ kann man Typen zur Compilezeit erkennen
- Extrem Praktisch bei langen Datentypenangaben / Templates
- ACHTUNG BEI
  - Integerdatentypen. Entweder Literale verwenden oder explizit angeben
  - = {} initialisierung. Datentyp ist dann eine Initialisierungsliste
- https://godbolt.org/g/5BJEaV

Copy und move

- Kurze Exkursion: „Value-Types“
- L-Values sind Datentypen auf der Linken Seite der Berechnung
  - Hat einen Namen und eine Adresse
  - Int i = 5 //das ist der L-Value
- R-Value sind Datentypen auf der Rechten Seite der Berechnung
  - Ist Temporär, kein Name
  - Int i = 5 // das ist der R-Value
- L-Value Reference verhält sich wie ein R-Value, Kann aber einen Namen haben.
- Neu in C++11. R-Value Reference oder universal Reference
  - Tolle Syntax mit && - Operator

Copy und move

- Zuweisungen sind normalerweise immer Kopien (solange der Optimizer aus ist)
- Darum bei großen Datentypen Referenzen verwenden (nur so groß wie die Adresse). Außer man braucht ein Kopie

- Neu in C++11 : move
  - Verschiebt die Referenz eines R- oder L-Ref-Value
  - Dadurch werden nicht Kopierbare Daten zuweisbar
  - Alte Variable wird dadurch undefiniert
  - Extrem Effizient da, da nur Adresse verschoben wird und das Objekt nicht verändert wird
  - Bonus: Wenn der Compiler auf C++11 eingestellt ist und ein move möglich ist, optimiert er das!!!!

Volatile

- Volatile heißt, dass der Compiler die Variable nicht wegoptimieren darf
  - Notwendig bei Werten die durch die Hardware gesetzt werden
  - Jedoch verbietet das auch jegliche Optimierung
- Nur einsetzten, wenn man sicher ist was man tut !!!!

Nullpointer

- Wie ist ein Nullpointer definiert??
- MISRA sagt: 0 verbieten, NULL hernehmen
- Joint Strike Fighter Guidelines sagt: NULL verbieten, 0 hernehmen

- Die C++11 lösung: nullprt als fester Datentyp
- Klare benennung, Typsicherheit, Compiletimechecks

Nächstes Mal:
Einführung in Klassen

- Weitere Links:
  - eli.thegreenplace.net/2011/12/15/understanding-lvalues-and-rvalues-in-c-and-c
  - stackoverflow.com/questions/3106110/what-are-move-semantics
  - stackoverflow.com/questions/28002/regular-cast-vs-static-cast-vs-dynamic-cast
  - possibility.com/Cpp/const.html
  - herbsutter.com/2013/08/12/gotw-94-solution-aaa-style-almost-always-auto/
  - youtube.com/watch?v=ZCGyvPDM0YY
  - akrzemi1.wordpress.com/2012/08/12/user-defined-literals-part-i/