Technisches Programmieren in C++

Das Konzept von C++

Karl Riedling
Institut für Sensor- und Aktuatorsysteme

Das Konzept von C++

- Allgemeines
- Programmier-Philosophie

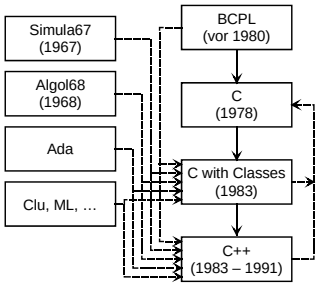
Das Konzept von C++

- **Allgemeines**
- Programmier-Philosophie

Das Konzept von C++

- Allgemeines
  - **Die Entstehung von C++**
  - Die Philosophie von C++

Die Entstehung von C++



Die Entstehung von C++

- C++ basiert auf C, weil C
  - flexibel, kompakt und verhältnismäßig Hardware-nahe,
  - auch für Systemprogrammierung geeignet,
  - portabel (Hardware und Betriebssystem) und
  - kompatibel mit der Programmierumgebung von UNIX ist.

Die Entstehung von C++

- Maschinennahe — C
- Problemnahe — C++
- Mit wenigen Ausnahmen sind alle Features von C auch unter C++ verfügbar.
- "*Gute C-Programme **sind** C++-Programme*":
- Neue Funktionen in C++ ersetzen viele Tricks, die in C notwendig waren.
- *C with Classes* und C++ wirkten auch auf (ANSI-) C zurück — z.B. *Funktions-Prototypen*.

Das Konzept von C++

- Allgemeines
  - Die Entstehung von C++
  - **Die Philosophie von C++**

Die Philosophie von C++

- Abstraktion von Daten — *Klassen* (*Classes*), in denen Datenstrukturen und die spezifischen Funktionen zu ihrer Behandlung zusammengefasst sind.
- Klassen sind *hierarchisch geordnet* — Konzept der *Vererbung* (*Inheritance*).
- *Schablonen* (*Templates*) für verwandte Klassen oder Funktionen.

Die Philosophie von C++

- Grundregeln (nach B. Stroustrup):
  - Wenn "*etwas*" als eigenständige Idee erscheint, definiere es als *Klasse*.
  - Wenn "*etwas*" eine eigenständige Einheit darstellt, definiere es als *Objekt* einer Klasse.
  - Wenn zwei Klassen "*etwas*" gemeinsam haben, mache es zu einer *Basisklasse* für beide.

Die Philosophie von C++

- In C gebräuchliche Konstrukte, die vermieden werden sollten:
  - Globale Daten;
  - Globale Funktionen;
  - Frei zugängliche Daten einer Klasse;
  - Direkte Zugriffe auf Daten anderer Objekte.

Die Philosophie von C++

- Flexibilität und Erweiterbarkeit:
  - C++ ist *funktionsorientiert*; es verwendet relativ wenige *Schlüsselworte* (*Keywords*), aber zahlreiche Standard-Bibliotheksfunktionen.
  - Standard-Bibliotheksfunktionen und Benutzer-definierte Funktionen können koexistieren.

Das Konzept von C++

- Allgemeines
- **Programmier-Philosophie**

Programmier-Philosophie

- Anforderungen an ein Programm (mit abnehmender Priorität):
  - Funktionalität
  - Benutzerfreundlichkeit
  - Wartbarkeit
  - Leistungsfähigkeit und Kompaktheit
  - Portabilität

Programmier-Philosophie

- Funktionalität:
  - Funktionsumfang des Programms *vorher* festlegen (gilt analog auch für die Funktionalität von Programm-Modulen und Funktionen!)
  - Pflichtenheft — Schutz für Auftraggeber und Auftragnehmer!

Programmier-Philosophie

- Benutzerfreundlichkeit:
  - Logisch aufgebaute Benutzerschnittstelle; intuitiv benutzbar;
  - Online-Hilfe;
  - Benutzer muss stets das Gefühl haben, zu wissen, was das Programm gerade tut;
  - kryptische Meldungen vermeiden ("Alle Dateien nicht löschen (J/N)?");
  - Nicht regelmäßig benötigte Funktionen (leicht auffindbar) "verstecken";

Programmier-Philosophie

- Benutzerfreundlichkeit:
  - Befehls- und Dateneingabe (außer bei einfachsten Hilfsprogrammen) grundsätzlich über Menüs;
  - "Abfangen" von fehlerhaften Eingaben zur Vermeidung von Laufzeitfehlern;
  - Fehlertoleranz: Auch im Fall schwerer Fehler Datenverluste möglichst vermeiden (*Autosave*);
  - Programm sollte auch von unfähigen oder bösartigen Usern nicht zum Absturz gebracht werden können.

Programmier-Philosophie

- Wartbarkeit:
  - Ziel: Einfache Erweiterung (vergrößerter Funktionsumfang) oder Korrektur (von Programmfehlern):
  - Modularer Aufbau: Aufteilung der Programmfunktionalität auf Funktionen (Unterprogramme) und der Funktionen auf Übersetzungseinheiten (Übersetzungseinheit, Modul = Datei);
  - *Einkapselung* von Funktionen: Jede Funktion hat einen exakt definierten Satz von Ein- und Ausgabedaten und eine exakt definierte Aufgabe;

Programmier-Philosophie

- Wartbarkeit:
  - Zusammenfassen zusammengehöriger Daten in Datenstrukturen (*Objekten*);
  - Mnemotechnisch sinnvolle beschreibende Namen für Variable und Funktionen;
  - Kommentare;
  - Optische Gliederung des Programmcodes;

Programmier-Philosophie

- Wartbarkeit:
  - Vermeidung von undurchsichtigen Programmkonstruktionen: Aufteilung auf aufeinander folgende Instruktionen ist
    - übersichtlicher;
    - besser abschätzbar;
    - führt zum gleichen *Object-Code*.

Programmier-Philosophie

- Leistungsfähigkeit und Kompaktheit:
  - Geschwindigkeit ist für viele Anwendungen *nicht* primär signifikant. Kritisch i.a. nur:
    - Oft durchlaufene innere Programmschleifen;
    - Hardware-naher Code (Interrupt-Handler).
  - Die Geschwindigkeit wird mehr durch den *Algorithmus* als durch die verwendete Programmiersprache bestimmt!

Programmier-Philosophie

- Leistungsfähigkeit und Kompaktheit:
  - Kompaktheit des Programmcodes ist heute nur mehr für ROM-basierte Anwendungen (*embedded controller*) relevant.

Programmier-Philosophie

- Portabilität des Programmcodes:
  - Wenn ein Programm auf unterschiedlichen Hard- und Software-Plattformen lauffähig sein soll, vermeiden:
    - hardware- oder compilerspezifische Erweiterungen der Programmiersprache;
    - "schmutzige Tricks" aller Arten.

Programmier-Philosophie

- "Schönheit" des Programmcodes:
  - "Schönheit" um ihrer selbst willen ist *sinnlos* und daher *nicht anzustreben*. Konsequenz:
  - Verwendung von Programmiertechniken wie objektorientiertes Programmieren, rekursive Funktionsaufrufe,… *dort und nur dort*, wo dies *Vorteile* bringt.
  - Nicht das "eleganteste" Programm ist auch das beste.
  - In *Ausnahmefällen* können auch "Verstöße" gegen die zuvor aufgestellten Regeln sinnvoll sein (z.B. Verwendung globaler Variablen, "Spaghetti-Code",…).