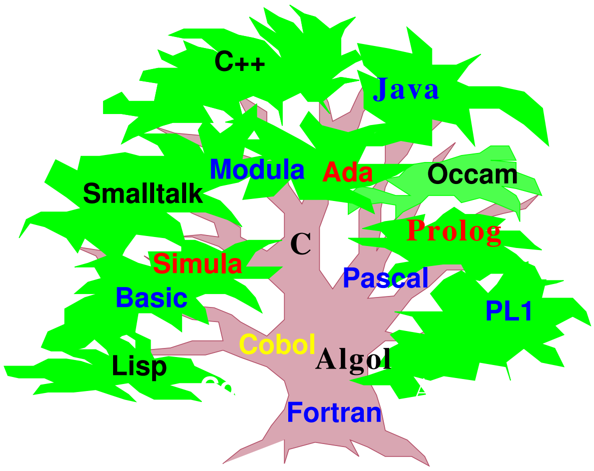
6. Ausnahmebehandlung (Exception Handling)

- Grundlagen
- Ausnahmebehandlung in PL1
- Ausnahmebehandlung in C++



Ausnahmebehandlung

- Ausnahmebedingung

  Programme sollen sich unter einem breiten Spektrum von Bedingungen "vernünftig" verhalten, selbst dann, wenn
  - in der betreffenden Hardware/Software ein Fehler vorliegt
  - ein so seltenes Ereignis eintritt, daß man es als anormal ansieht
  - ungültige Eingabedaten gelesen werden

  . . .
  - **jedoch nicht** bei einem "katastrophalen" Fehler

  ⇒ Bei einer Ausnahmebedingung oder einem "anormalen" Verarbeitungszustand kann die gerade aktive Programmeinheit nicht "normal" fortfahren, d.h. die nächste Anweisung ausführen.

- Wenn ein Programm keine anormalen Verarbeitungszustände erkennen und angemessen auf sie reagieren kann, wird es **nur unter bestimmten Umständen** ordnungsgemäß arbeiten ⇒ fehlende Robustheit, mangelnde Fehlertoleranz

- Traditionelle Programmiersprachen (wie MODULA-2) bieten **keine speziellen Hilfsmittel** zur ordnungsgemäßen Behandlung von Ausnahmebedingungen (außer PL/I)

- In neueren Programmiersprachen wird versucht, systematisch **Funktionen für die Ausnahmebehandlung** bereitzustellen. Sie erlauben, alle Aspekte der Bearbeitung von Anomalien aus dem normalen Programmfluß herauszuhalten.

Ausnahmebehandlung (2)

- Wesentliche Problemaspekte
  - vordefinierte vs. benutzerdefinierte Ausnahmen
  - Deklaration und Geltungsbereich einer Ausnahme (exception)
  - Erzeugung oder Signalisierung einer Ausnahme
  - Spezifikation von Maßnahmen zur Behandlung einer Ausnahme (Ausnahmebehandlungsroutinen bzw. Exception Handler)
  - Aufruf der betreffenden Behandlungsroutine nach Eintritt der Ausnahme
  - Fortsetzung der Programmausführung nach Behandlung der Ausnahme
  - Deaktivieren von Ausnahmen

- **Wichtigste Entwurfsentscheidung:** Wo wird das unterbrochene Programm fortgesetzt?

Wiederaufnahmemodell: Rückgabe der Kontrolle an den Punkt, an dem die Ausnahme eintrat (z.B. in PL/I)
  - leistungsfähiger und flexibler Mechanismus
  - aber oft Kurieren an Fehlersymptomen und nicht Beseitigung ihrer Ursachen

Terminierungsmodell: Beenden der Programmeinheit, in der die Ausnahme auftrat (z.B. Ada)
  - unterbrochene Aktion kann nicht fortgesetzt werden
  - Übergabe der Kontrolle an Exception Handler

Ausnahmebehandlung in PL/I

- Unterstützung zahlreicher vordefinierter Bedingungen sowie von Benutzer-Bedingungen
  - Ein-/Ausgabe-Bedingungen (dateibezogen): ENDFILE, ENDPAGE, ...
  - Speicher-Bedingungen (Zuordnung von Speicher)
  - sonstige Ablauffehler: ZERODIVIDE, SUBSCRIPTRANGE, OVERFLOW, UNDERFLOW, CONVERSION...
  - Benutzer-Bedingungen: vereinbarte Bedingungen mit Namen
  - Mit der Anweisung SIGNAL kann der Eintritt einer Bedingung simuliert werden

- Definition von Exception Handlern: **ON-Anweisung**

```
ON Bedingungsname Anweisung; /* z.B. BEGIN-Block */
```

  - Anweisungen der *ON-Einheit* werden immer dann durchlaufen, wenn zugehörige Bedingung auftritt
  - die Bedingungen können analysiert und manchmal behoben werden
  - in jedem Fall kann eine anwendungsbezogene Fehlermeldung abgesetzt werden
  - ist keine ON-Einheit vereinbart, tritt eine in PL/I festgelegte ON-Einheit (SYSTEM) an ihre Stelle
  - On-Einheiten haben keine Parameter

Ausnahmebehandlung in PL/I (2)

- dynamische Bindung von Exception Handlern zu Bedingungen !
  - es können mehrere On-Einheiten pro Bedingung spezifiziert werden
  - es wird stets die bei der Programmausführung als letztes durchlaufene On-Einheit einer Bedingung zur Ausführung gebracht
  - die Bindung einer On-Einheit zu einer Bedingung endet daher mit Durchlaufen einer neuen On-Einheit oder mit Beendigung des Blocks, in dem die On-Einheit definiert wurde
  - bei Schachtelung gilt ggf. die On-Einheit des dynamisch übergeordneten Blocks

- Beispiel

```
    BEGIN;
       ...
       ON SUBSCRIPTRANGE BEGIN;  PUT LIST (‘ Fehler - Indexbedingung für Feld A ‘);
                                 GO TO FIXIT;
       END;
       ...
       A (I) = I+1;
       ...
       ON SUBSCRIPTRANGE BEGIN;  PUT LIST (‘ Fehler - Indexbedingung für Feld B‘);
                                 STOP RUN;
       END;
       ...
LABEL1: ...
       B (J) = SUM;
       ...
    END;
```


Ausnahmebehandlung in C++

- alle Ausnahmebedingungen müssen explizit ausgelöst werden: `throw`-Anweisung

- Exception Handler
  - Definition mit `catch`-Anweisung
  - Zuordnung zu sogenannten `try`-Blöcken

- Beispiel

```
int main { ...
    ifstream Eingabe (“file1.txt”);
  try {
       if (!Eingabe)      // Fehler waehrend Oeffnen
            throw 1;
       ... }

    catch (int) {
            cout << “Fehler beim Öffnen der Eingabedatei!” << endl
            ... }
```

- Exception Handler haben keinen Namen; Auswahl erfolgt über Parametertyp !

Ausnahmebehandlung in C++ (2)

- Ausnahmebehandlung vor allem vorteilhaft bei Verwendung von Klassen
  - Implementierer der Klasse legt fest, welche Ausnahmen wann ausgelöst werden
  - Nutzer der Klasse spezifizieren, wie auf Ausnahmen reagiert wird (durch Festlegung der Exception Handler)

- Beispiel

```
class Vector {    int* p;
                  int sz;
    public:       class Range {};    // Range-Bedingung
                  class Size {};     // Size-Bedingung
                  Vector (int sz) { if (sz < 0 || sz>1000) throw Size();
                              /* ... */ }
                  int& operator[] (int i) {
                              if (0 <= i && i < sz) return p[i];
                              throw Range();   }

void f () { // ...
    try { use_vectors (); }
    catch (Vector::Range) { // ... Anpassungen vornehmen etc.
                              f ();   // erneuter Versuch
       }
    catch (Vector::Size) { cerr << "Fehler Vector:Size !"; exit (99); }
}
```


Ausnahmebehandlung in C++ (3)

- Parameter können genutzt werden, Daten an Exception Handler zur gezielteren Fehlerbehandlung zu übergeben

- Bestimmung des auszuführenden Exception Handler
  - zu dem try-Block, in dem der Fehler aufgetreten ist, wird die Liste der Exception Handler durchsucht
  - der erste Exception Handler mit “passender” Typdeklaration wird ausgeführt
  - findet sich kein passender Handler, wird der try-Block verlassen und die Handler-Liste des in der dynamischen Aufrufhierarchie übergeordneten try-Blocks durchsucht
  - wird kein anwendbarer Exception Handler gefunden, erfolgt Programmabbruch

- nach Ausführung eines Exception Handlers erfolgt kein Rücksprung an die auslösende Stelle im Programm, sondern es wird mit der Anweisung fortgefahren, die nach der Handler-Liste des ausgeführten Handlers steht

- Fehler während Fehlerbehandlung
  - Exception Handler kann selbst try-Block mit zugeordneten Exception Handlern enthalten
  - findet sich im Exception Handler kein passender Handler, wird Suche im übergeordneten try-Block fortgesetzt

Ausnahmebehandlung in C++ (4)

- Gruppenbildung für Ausnahmen: Einsatz von Klassenhierarchien

```
class Matherr {};
class Overflow: public Matherr {};
class Underflow: public Matherr {};
class Zerodivide: public Matherr {};
...

try {
    // ...
    }
catch (Overflow) {
      // Behandlung von Überlauffehlern
    }

catch (Matherr) {
      // Behandlung aller übrigen Matherr-Fehler (außer Overflow)
    }
```

  - unterstützt spätere Erweiterbarkeit um zusätzliche Fehlerbedingungen
  - Reihenfolge der Handler wichtig: speziellere Handler immer zuerst!

Zusammenfassung

- Mechanismen zur Ausnahmebehandlung unterstützen
  - Trennung der Programmlogik für Normalfall und Fehlerfall
  - bessere Lesbarkeit und Änderbarkeit von Programmen
  - größere Robustheit, vor allem für große Programme
- Entwurfsaspekte
  - vordefinierte vs. benutzerdefinierte Ausnahmebedingungen
  - Auslösen von Ausnahmen
  - Auswahl des Exception Handlers • • •
- Weiterarbeit nach Ausnahmebehandlung: Wiederaufnahmemodell vs. Terminierungsmodell
- PL1
  - umfassende Unterstützung
  - erschwerte Nutzbarkeit aufgrund dynamischer Geltungsbereiche für Exception Handler
- C++
  - Übergabe von Parameterwerten an Exception Handler
  - kooperative Fehlerbehandlung zwischen Klassenentwickler und -nutzer