Programmiermethoden in der Mathematik WS 02/03:
Grundstruktur eines C++ Programms am Beispiel von `prog1.cc`

- C++ unterscheidet zwischen Groß- und Kleinschreibung.
- **Kommentare** werden vom Compiler ignoriert. Sie können auf zwei Arten gekennzeichnet werden:
  - Nach `//` ist der Rest der Zeile ein Kommentar (C++ Stil).
  - Alles, was zwischen `/*` und `*/` steht, ist ein Kommentar, auch über mehrere Zeilen (C Stil).
- Als erstes ruft der Compiler den Präprozessor auf, der alle Zeilen, die mit `#` beginnen, liest und einfache Ersetzungen durchführt. **Präprozessoranweisungen** beginnen also mit `#`. Es gibt mehrere Arten, u.a.:
  - Die Anweisung `#include <iostream.h>` bindet die Header- oder include-Datei `iostream.h` ein, d.h. der Inhalt der Datei wird an der Stelle der Präprozessoranweisung eingesetzt.
    - Wird die Header-Datei in spitzen Klammern (`<..>`) angegeben, so sucht der Präprozessor sie in den Verzeichnissen, wo die Standard C++ Header-Dateien normalerwesie abgelegt sind (normalerweise `/usr/include`).
    - Wird die Header-Datei dagegen in Anführungsstrichen ("...") angegeben, so sucht der Präprozessor sie im aktuellen Verzeichnis. Diese Form wird also besonders für selbstgeschriebene Header-Dateien benutzt. Mit einer Compileroption kann man auch Verzeichnisse angeben, in denen nach Header-Dateien gesucht wird.

    Header-Dateien haben meistens die Endung `.h`, die neueren System-Header-Dateien haben jedoch gar keine Endung mehr. Für selbstgeschriebene Header-dateien empfiehlt sich aber weiterhin, diese Endung zu benutzen.
  - Mit einer `define` Anweisung werden Textersetzungen vom Präprozessor durchgeführt:

    ```
    #define N 10
    ```

    ersetzt im ganzen Programmtext den Text `N` (wenn er allein, also getrennt durch Leerzeichen, steht) durch den Text `10`.
- Jedes C++ Programm besteht aus **Anweisungen**, die sequentiell von oben nach unten und links nach rechts abgearbeitet werden. Jede Anweisung wird mit einem Semikolon (Strichpunkt) abgeschlossen. Leerzeichen, Einrückungen und Zeilenumbrüche haben keine Bedeutung, sie dienen nur der besseren Lesbarkeit des Quellcodes. Mehrere Anweisungen können innerhalb geschweifter Klammern `{...}` zu einem *Block* zusammengefasst werden, z.B. bei Schleifen. Ein Programm wird wesentlich übersichtlicher, wenn die Anweisungen, die zu einem Block gehören, eingerückt sind, und wenn in jeder Zeile nur eine Anweisung steht.
- Alle Anweisungen eines Programms werden in einem mit geschweiften Klammern eingeschlossenen Block, dem **Hauptprogramm**, zusammengefasst. Das Hauptprogramm muss den Namen `main` haben. Dieser Name steht vor dem Block mit den Anweisungen des Programms in der Form

  ```
  main()
  {
    // Anweisungen:
    ...
  }
  ```

  - Das Hauptprogramm kann Eingabewerte bekommen, die beim Aufruf des Programms mit eingegeben werden. Der `g++` z.B. nimmt als Eingabe den Namen der zu compilierenden Quelldatei entgegen (`g++ prog1.cc`). Der Datentyp (ganzahlig, Text etc.) der Eingabewerte wird in den runden Klammern hinter dem Wort `main` angegeben. Sind die Klammern leer (wie oben) bzw. steht dort `void` (engl. *leer*, wie in `prog1.cc`), dann hat das Programm keine Eingabewerte.
  - Analog kann ein Programm auch einen Wert zurückgeben. Damit kann man z.B. Fehler anzeigen. Dies tun viele Unix-Systemprogramme. Der Typ des zurückgegeben Wertes wird vor dem Wort `main` angegeben. Wie im Hauptprogramm ein Wert zurückgegeben wird, werden wir später sehen.

  Im oberen Beispiel nimmt das Programm also weder ein Wert entgegen, noch gibt es einen zurück. Dies kann man deutlicher auch als

  ```
  void main(void)
  ```

  schreiben. In `prog1.cc` kann das Programm einen `int` Wert zurückgeben.
- In einem Programm benutzt man **Variablen**, denen man im Laufe des Programms verschiedene Werte zuweisen kann.
  - Variablen haben einen Namen, der
    - eine beliebige Kombination aus Buchstaben, Zahlen und dem Unterstrich `_` sein kann,

    * aber nicht mit einer Zahl beginnen darf
    * und kein reserviertes Wort in C++ sein darf, also z.B. `for, int, short` usw.
  - Variablen haben einen eindeutigen Typ, der in der **Variablendeklaration** festgelegt wird. Dabei steht zuerst der Typname und dahinter der Name der Variablen, z.B.

    ```
    int i;
    ```

    Mehrere Variablen des selben Typs können, durch Komma getrennt, zusammen deklariert werden:

    ```
    int i,j,k;
    ```

    Variablen *müssen* deklariert werden, bevor sie benutzt werden können. Sie dürfen aber nur einmal deklariert werden. Eine gewisse Ausnahme gibt es bei der *for*-Schleife, s.u.

- Nach der Deklaration haben Variablen noch keinen speziellen Wert, sie sind noch nicht *initialisiert*. Manche Compiler initialisieren alle Variablen automatisch, dies ist jedoch nicht Standard. Mit Compileroptionen kann man eine automatische **Initialisierung** steuern. Es ist jedoch immer sicherer, alle Variablen explizit zu initialisieren. Dies passiert in `prog1.cc` durch die Anweisung

  ```
  wert=0;
  ```

  Deklaration und Initialisierung können zusammen erfolgen:

  ```
  int wert=0;
  ```

- Im Gegensatz zu Variablen haben **Konstanten** *einen festen Wert* während des ganzen Programms. Sie werden wie Variablen deklariert, aber mit einem vorangestellten `const`. Sie müssen direkt bei der Deklaration initialisiert werden, also:

  ```
  const int wert=0;
  ```

- Die allgemeine Form der **for-Schleife** wurde schon im Tutorium besprochen. Bei der *for*-Schleife benutzt man meistens eine ganzzahlige Zählvariable, die in jedem Schleifendurchlauf verändert wird. Diese Variable muss wie jede andere auch deklariert werden. Es gibt dazu zwei Möglichkeiten:
  - Die übersichtlichere und vor allem sicherere ist die folgende: Die Zählvariable wird direkt in der *for*-Anweiseung, d.h. in der ersten Anweisung in den runden Klammern nach dem Schlüsselwort `for`, deklariert:

    ```
    for(int i=1;i<=10;i++) ...;
    ```

    Sie *existiert dann nur innerhalb der* for-*Schleife* und ist außerhalb nicht definiert. Gibt es außerhalb der *for*-Schleife ebenfalls eine Variable mit demselbem Namen, also z.B.

    ```
    int i;
    for(int i=1;i<=10;i++) ...;
    ```

    so handelt es sich um *eine andere Variable*. Ihr Wert wird durch die *for*-Schleife nicht verändert. Die in der *for*-Anweisung deklarierte Variable hat also eine sehr beschränkte, nur *lokale* Gültigkeit.
  - Die früher einzig mögliche Variante ist, außerhalb der for-Schleife eine Variable zu deklarieren und diese in der for-Schleife zu benutzen:

    ```
    int i;
    for(i=1;i<=10;i++) ...;
    ```

    Die Variable `i` ist dann im gesamten Programm bekannt und kann auch außerhalb der for-Schleife benutzt werden. Nach der *for*-Schleife ist aber der Wert, den die Variable vorher hatte, überschrieben. Dies kann zu Fehlern führen, die nur schwer zu entdecken sind.

- **Ausgaben auf den Bildschirm** und Eingaben über die Tastatur werden durch die sogenannten *Output- und Input-Streams* `cout` und `cin` durchgeführt. Um sie zu benutzen, muss die Header-Datei `iostream.h` eingebunden werden.
  - Mit dem Operator `<<` werden Variablen gewissermassen in den Output-Stream `cout` "hineingeschoben". Mehrere Variablen werden dabei jeweils durch einen `<<` Operator voneinander getrennt. Ausgabe auf den Bildschirm erfolgt also mit einer Anweisung der Form

    ```
    cout  <<  wert;
    ```

    Texte werden in doppelten, einzelne Zeichen in einfachen Hochkommas eingeschlossen, ein solcher Text (*string*) gilt als eine Variable. Ein Zeilenumbruch wird durch ein `endl` (end line) angegeben, oder innerhalb eines strings durch das Zeichen `\n`. Also erzeugen

    ```
    cout  <<  "Das Ergebnis ist " << wert << endl;
    ```

    und

    ```
    cout  <<  "Das Ergebnis ist " << wert << "\n";
    ```

    (wie in `prog1.cc`) dieselbe Ausgabe. Die zweite Variante ist dabei eher C-Stil.
  - Eingaben von der Tastatur erfolgen analog mit

    ```
    cin  >>  wert;
    ```

  Wie Ausgaben speziell formatiert werden, werden wir später sehen.