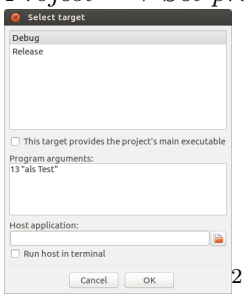
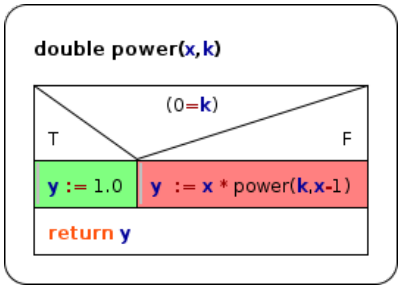
Draft der C++-Vorlesung vom 27.März 2020

1. Funktion `main`: Siehe [Haase20, §7.6] und Bsp. v_5b[1].
   Das Hauptprogramm ist eine Funktion mit dem vorgeschriebenem Namen `main` und dem Interface
   ```
   int main( int const argc, char const* argv[] )
   ```
   - Der Wert des Rückgabeparameters vom Typ `int` kann in der aufrufenden Programmumgebung ausgewertet werden. Ein Wert 0 steht hierbei für fehlerfreie Programmabarbeitung. Daher auch die standardmäßige Zeile
     `return 0;`
     am Ende der Funktion `main`.
   - Umgekehrt kann die Programmumgebung der Funktion `main` eine beliebige Anzahl von *Kommandozeilen*parametern übergeben, allerdings nur als klassische C-Strings (`'\0'` terminated). Folglich sind
     - `argc` die Anzahl dieser C-Strings und
     - `argv[]` is ein (C-)Array mit solchen C-Strings.
     - Es gilt stets `argc`≥ 1, da `argv[0]` stets den Namen des laufenden Programmes enthält.
     - Welchen Nutzen habe ich von Kommandozeilenparametern?
       Sie können Eingabeparameter an Ihr Programm ohne lästiges (mehrfaches) Eintippen oder neues Compilieren übergeben (Erhöhung der Flexibilität).
   - Wie kann ich solche Kommandozeilenparameter an mein Programm übergeben?
     - Aus der Kommandozeile im Terminal mit Leerzeichen getrennt, z.B.
       `v_5b.exe 13 ‘‘als Test‘‘`
     - In CodeBlocks über die Einstellung
       Project $\longrightarrow$ *Set program's arguments* $\longrightarrow$ *P. arguments* $\longrightarrow$ 13 “als Test“
       

       [2]
   - Wie nutze ich diese Kommandozeilenparameter in meinem Programm?
     - Wurden Kommandozeilenparameter übergeben? $\longrightarrow$ `main.cpp:24`
     - Falls nicht, dann Eingabe über Tastatur $\longrightarrow$ `main.cpp:32-37`
     - Falls ja, dann Konvertierung in benötigten Datentyp, $\longrightarrow$ `main.cpp:26`
       in diesem Fall wird der C-String mittels der Funktion `atoi(argv[1])` (sprich AsciiToInteger) auf eine Variable des Datentyps `int` umgewandelt.
     - Eine Auswahl von Konvertierungsfunktionen:
       `atoi`, `atol`, `atof`, `strtol`, `strtod`, ... `#include <cstdlib>`
       `stoi`, `stol`, `stof`, `stod` `#include <string>`
   - Die umwandlung von numerischen Werten in C++-Strings erfolgt mittels der Funktion `to_string`, siehe Bsp. to_string[3].

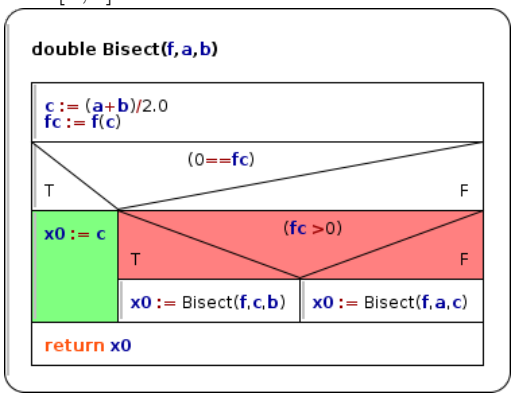
[1] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/v_5b.zip
[2] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/v_5b_arguments.png
[3] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/to_string.zip

2. Rekursive Funktion: Bsp. Power, siehe [Haase20, §7.7] und Bsp. Ex770.cpp[4].



Jede Rekursion hat einen (mehrfache) Test(s) auf Abbruch woraufhin Standardwerte im Abbruchzweig zurückgegeben werden. Ansonsten wird die Funktion mitgeänderten Parametern im Rekursionsteil aufgerufen.

3. Rekursive Funktion: Bsp. Bisektion, Bestimme eine Nullstelle der Funktion $f(x)$ mit $x \in [a, b]$.



Den Anfang von [Haase20, §7.8] aufschlagen und lesen.

$$x_0 := \text{Bisect}(a, b, \varepsilon)$$
$$:= \begin{cases} c := (a+b)/2 & \text{falls } f(c) == 0 \\ \text{Bisect}(c, b, \varepsilon) & \text{sonst, falls } f(c) > 0 \\ \text{Bisect}(a, c, \varepsilon) & \text{sonst, falls } f(c) < 0 \end{cases}$$

- Wegen numerischer Rundungsfehler der Gleitkommazahlen kann es passieren, daß der Test des Funktionswertes auf die exakte 0, $0 == fc$, niemals erfüllt wird. Um diese *Endlosrekursion* zu vermeiden und um nur bis zu einer *sinnvollen Genauigkeit* zu rechnen, wird dieser Test durch $|fc| < \varepsilon$ ersetzt. Das ε kann/sollte als Inputparameter an die Funktion übergeben werden, wie in Bisect1.cpp:15[5].
  Die math. Funktion $\sin(x) - x/2$ ist in `Bisect1.cpp:49` direkt im Funktionskörper implementiert, dies geht flexibler.

- **C++-Funktionen als Inputparameter** von Funktionen:
  Die in `Bisect` benötigte Funktion $f(x)$ besitzt die Signatur `double f(double)`, siehe Bisect3.cpp:18[6]. Eine Funktion dieser Signatur mit spezifiziertem Rückgabetypen kann via
  ```
  const std::function<double(double)>& func
  ```
  als Inputputparameter einer Funktion deklarieriert werden. Damit ergibt sich die Funktionsdeklaration (`Bisect3.cpp:49`)
  ```
  double Bisect3(const std::function<double(double)>& func,
                 const double a, const double b, const double eps=1e-6);
  ```
  welche in `Bisect3.cpp:67` aus dem Hauptprogramm aufgerufen wird.
  Benötigt `#include <functional>` und die Compileroption '-std=c++11'.

- In obiger Deklaration ist der Parameter `double eps=1e-6` ein optionaler Parameter mit dem Standardwert 1e-6 (default). Dies erlaubt den Aufruf der Funktion als
  - `Bisect3(g, anfang, ende, tol*tol)` , aber auch
  - `Bisect3(g, anfang, ende)` `wobei dann eps=1e-6` gilt.

[4] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/../Beispiele/Ex770.cpp
[5] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/../Beispiele/Bisect1.cpp
[6] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/../Beispiele/Bisect3.cpp

Bislang galt stets die Voraussetzung $f(a) > 0 > f(b)$ welche in der abschließenden Implemenierung v_5d[7] nicht mehr nötig ist.

- Vollständige Fallunterscheidung bzgl. der Relationen von $f(a)$, 0 und $f(b)$, siehe dazu [Haase20, p.67f] und `v_5d/main.cpp:113-137`.
- Flexible Auswahl aus mehreren, vordefinierten Funktionen für $f(x)$, siehe dazu die Zuweisungen der Variablen
  `std::function<double(double)> ff;`
  über die Fallauswahl in `v_5d/main.cpp:78-100`.

Literatur

[Haase20] Gundolf Haase: Einführung in die Programmierung mit C++ (2020), *www*[8].

[Stroustrup10] Bjarne Stroustrup: Einführung in die Programmierung mit C++. Pearson Studium, München (2010).

[7] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS20/v_5d.zip
[8] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf