Funktionspointer

- Deklaration
- Bisektionsverfahren

Funktionspointer

- Funktionsaufruf ist Sprung an eine Adresse
  - Pointer speichern Adressen
  - kann daher Fkt-Aufruf mit Pointer realisieren
- Deklaration eines Funktionspointers:
  - `<return value> (*pointer)(<input>)`; deklariert Pointer `pointer` für Funktionen mit Parametern `<input>` und Ergebnis vom Typ `<return value>`
- Bei Zuweisung müssen Pointer `pointer` und Funktion dieselbe Struktur haben
  - gleicher Return-Value
  - gleiche Input-Parameter-Liste
- Aufruf einer Funktion über Pointer wie bei normalem Funktionsaufruf!

Elementares Beispiel

```
 1  #include <stdio.h>
 2
 3  void output1(char* string) {
 4    printf("*%s*\n",string);
 5  }
 6
 7  void output2(char* string) {
 8    printf("#%s#\n",string);
 9  }
10
11  main() {
12    char string[] = "Hello World";
13    void (*output)(char* string) = NULL;
14
15    output = output1;
16    output(string);
17
18    output = output2;
19    output(string);
20  }
```

- **Output:**

```
*Hello World*
#Hello World#
```


Bisektionsverfahren

```
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  double bisection(double (*fct)(double x),
 5                   double a, double b,
 6                   double tol) {
 7    double m = 0;
 8
 9    while ( b-a > tol) {
10      m = (a+b)/2;
11      if ( fct(a)*fct(m) <= 0 ) {
12        b = m;
13      }
14      else {
15        a = m;
16      }
17    }
18    return m;
19  }
20
21  double f(double x) {
22    return x*x+exp(x)-2;
23  }
24
25  main() {
26    double a = 0;
27    double b = 10;
28    double tol = 1e-12;
29
30    double x = bisection(f,a,b,tol);
31
32    printf("Nullstelle x=%1.15e\n",x);
33  }
```

- Approximation der Nullstelle von $f(x) = x^2 + e^x - 2$

Struktur für Funktionen 1/2

```
1  #ifndef FUNCTION
2  #define FUNCTION
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <math.h>
7
8  typedef struct _Function_ {
9    double a,b;            // domain for f:[a,b]->R
10   double (*f)(double x); // function evaluation f(x)
11 } Function;
12
13 // store domain [a,b] of function f
14 void setFunctionA(Function* f, double a);
15 void setFunctionB(Function* f, double b);
16
17 // define function
18 void setFunction(Function* f, double (*fct)(double x));
19
20 // return domain [a,b] of function
21 double getFunctionA(Function* f);
22 double getFunctionB(Function* f);
23
24 // evaluate function and return f(x)
25 double evalFunction(Function* f, double x);
26
27 #endif
```

- Funktionspointer kann Teil einer Struktur sein
  - z.B. Problemdefinition für Nullstellensuche

Struktur für Funktionen 2/2

```
1  #include "function.h"
2
3  void setFunctionA(Function* f, double a) {
4    f->a = a;
5  }
6
7  void setFunctionB(Function* f, double b) {
8      f->b = b;
9  }
10
11 void setFunction(Function* f, double (*fct)(double x)) {
12     f->f = fct;
13 }
14
15 double getFunctionA(Function* f) {
16   return f->a;
17 }
18
19 double getFunctionB(Function* f) {
20   return f->b;
21 }
22
23 double evalFunction(Function* f, double x) {
24   return f->f(x);
25 }
```

- Erinnerung:

```
typedef struct _Function_ {
  double a,b;            // domain for f:[a,b]->R
  double (*f)(double x); // function evaluation f(x)
} Function;
```


Bisektionsverfahren v2

- und noch einmal das Bisektionsverfahren...

```
1  #include "function.h"
2  #include "function.c"
3
4  double bisection(Function* f, double tol) {
5    double a = getFunctionA(f);
6    double b = getFunctionB(f);
7    double m = 0;
8
9    while ( b-a > tol) {
10     m = (a+b)/2;
11     if ( evalFunction(f,a)*evalFunction(f,m) <= 0 ) {
12       b = m;
13     }
14     else {
15       a = m;
16     }
17   }
18   return a;
19 }
20
21 double f(double x) {
22   return x*x+exp(x)-2;
23 }
24
25 main() {
26   double tol = 1e-12;
27   double x = 0;
28
29   Function* fct = malloc(sizeof(Function));
30   setFunctionA(fct,0);
31   setFunctionB(fct,10);
32   setFunction(fct,f);
33
34   x = bisection(fct,tol);
35   printf("Nullstelle x=%1.15e\n",x);
36 }
```


C++

- Was ist C++
- Wie erstellt man ein C++ Programm?
- `Hello World!` mit C++

- `main`
- `cout, cin, endl`
- `using std::`
- Scope-Operator `::`
- Operatoren <<, >>
- `#include <iostream>`

Was ist C++

- Weiterentwicklung von C
  - Entwicklung ab 1979 bei AT&T
  - Entwickler: Bjarne Stroustrup
- C++ ist abwärtskompatibel zu C
  - keine Syntaxkorrektur
  - aber: stärkere Zugriffskontrolle bei "Strukturen"
    - Datenkapselung
- Compiler:
  - frei verfügbar in Unix/Mac: `g++`
  - Microsoft Visual C++ Compiler
  - Borland C++ Compiler

Objektorientierte Programmiersprache

- C++ ist objektorientiertes C
- Objekt = Zusammenfassung von Daten + Fktn.
  - Funktionalität hängt von Daten ab
  - vgl. Multiplikation für Skalar, Vektor, Matrix
- Befehlsreferenzen
  - `http://en.cppreference.com/w/cpp`
  - `http://www.cplusplus.com`

Wie erstellt man ein C++ Prg?

- Starte Editor Emacs aus einer Shell mit `emacs &`
  - Die wichtigsten Tastenkombinationen:
    - `C-x C-f` = Datei öffnen
    - `C-x C-s` = Datei speichern
    - `C-x C-c` = Emacs beenden
- Öffne eine (ggf. neue) Datei `name.cpp`
  - Endung `.cpp` ist Kennung für C++ Programm
- Die ersten beiden Punkte kann man auch simultan erledigen mittels `emacs name.cpp &`
- Schreibe *Source-Code* (= C++ Programm)
- Abspeichern mittels `C-x C-s` nicht vergessen
- Compilieren z.B. mit `g++ name.cpp`
- Falls Code fehlerfrei, erhält man *Executable* `a.out` unter Windows: `a.exe`
- Diese wird durch `a.out` bzw. `./a.out` gestartet
- Compilieren mit `g++ name.cpp -o output` erzeugt Executable `output` statt `a.out`

Hello World

```
1  #include <iostream>
2
3  int main() {
4    std::cout << "Hello World!\n";
5    return 0;
6  }
```

- C++ Bibliothek für Ein- und Ausgabe ist iostream
- `main` hat zwingend Rückgabewert `int`
  - `int main()`
  - `int main(int argc, char* argv[])`
    - insbesondere `return 0;` am Programmende
- Scope-Operator `::` gibt *Name Space* an
  - alle Fktn. der Standardbibliotheken haben `std`
- `cout` ist die Standard-Ausgabe (= Shell)
  - Operator `<<` übergibt rechtes Argument an `cout`

```
1  #include <iostream>
2  using std::cout;
3
4  int main() {
5    cout << "Hello World!\n";
6    return 0;
7  }
```

- `using std::cout;`
  - `cout` gehört zum *Name Space* `std`

Shell-Input für Main

```
1  #include <iostream>
2  using std::cout;
3  using std::endl;
4
5  int main(int argc, char* argv[]) {
6    int j = 0;
7    cout << "This is " << argv[0] << endl;
8    cout << "got " << argc-1 << " inputs:" << endl;
9    for (j=1; j<argc; ++j) {
10     cout << j << ": " << argv[j] << endl;
11   }
12   return 0;
13 }
```

- `<<` arbeitet mit verschiedenen Typen
- kann mehrfache Ausgabe machen `<<`
- `endl` ersetzt `"\n"`
- Shell übergibt Input als C-Strings an Programm
  - argc = Anzahl der Parameter
  - argv[0] = Programmname
  - d.h. argc−1 echte Input-Parameter
- Output für `./a.out Hello World!`

```
This is ./a.out
got 2 inputs:
1: Hello
2: World!
```

Eingabe / Ausgabe

```
 1  #include <iostream>
 2  using std::cin;
 3  using std::cout;
 4  using std::endl;
 5
 6  int main() {
 7    int x = 0;
 8    double y = 0;
 9    double z = 0;
10
11    cout << "Geben Sie einen Integer ein: ";
12    cin >> x;
13    cout << "Geben Sie zwei Double ein: ";
14    cin >> y >> z;
15
16    cout << x << " * " << y << " / " << z;
17    cout << " = " << x*y/z << endl;
18
19    return 0;
20  }
```

- `cin` ist die Standard-Eingabe (= Tastatur)
  - Operator `>>` schreibt Input in Variable rechts
- Beispielhafte Eingabe / Ausgabe:

```
Geben Sie einen Integer ein: 2
Geben Sie zwei Double ein: 3.6 1.3
2 * 3.6 / 1.3 = 5.53846
```

- `cin` / `out` gleichwertig mit `printf` / `scanf` in C
  - aber leichter zu bedienen
  - keine Platzhalter + Pointer