Einführung in die Programmierung mit C++

Rekursion

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Inhalt

```
1  #include<iostream>
2  #include<cassert>
3
4  unsigned int f(unsigned int n) {
5    if (n==0) return 1;
6    return n*f(n-1);
7  }
8
9  int main(int argc, char* argv[]) {
10   assert(argc==2);
11   int n=std::stoi(argv[1]);
12   assert(n>=0);
13   std::cout << f(n) << std::endl;
14   return 0;
15 }
```

- Als Rekursion bezeichnet man den Aufruf einer Funktion durch sich selbst bzw. durch andere Funktionen, die durch sich selbst aufgerufen wurden.
- Entsprechend werden mehrere Instanzen des lokalen Datensegments im statischen Datenspeicher (auf dem *stack*) alloziert.
- Mindestens eine Abbruchbedingung wird benötigt, um die Terminierung des Programms zu ermöglichen.
- Beispiel: Fakultät

→ gdb / Tafel: Speicherlayout

AUFWACHEN!

Was wird bei Aufruf des Programms mittels `./f.exe 1 ? XXX y2k` ausgegeben?

```
1  #include<iostream>
2
3  int f(const int n) {
4    static int c=0;
5    c++;
6    if (c==n) std::cout << c << std::endl;
7    if (n==0) return 1;
8    return n*f(n-1);
9  }
10
11 int main(int c, char*[]) {
12   std::cout << f(c) << std::endl;
13   return 0;
14 }
```

Wach?

Was wird bei Aufruf des Programms mittels `./f.exe 1 ? XXX y2k` ausgegeben?

```
1  #include<iostream>
2
3  int f(const int n) {
4    static int c=0;
5    c++;
6    if (c==n) std::cout << c << std::endl;
7    if (n==0) return 1;
8    return n*f(n-1);
9  }
10
11 int main(int c, char*[]) {
12   std::cout << f(c) << std::endl;
13   return 0;
14 }
```

3
120

WACH BLEIBEN!

Was wird bei Aufruf des Programms mittels `./f.exe y2K ... xxx` ausgegeben?

```
1  #include<iostream>
2
3  int f(const int n) {
4    static int c=0;
5    c++;
6    if (c==n) std::cout << c << std::endl;
7    if (n==0) return 1;
8    return n*f(n-1);
9  }
10
11 int main(int c, char*[]) {
12   std::cout << f(c) << std::endl;
13   return 0;
14 }
```

Noch Wach?

Was wird bei Aufruf des Programms mittels `./f.exe y2K ... xxx` ausgegeben?

```
1  #include<iostream>
2
3  int f(const int n) {
4    static int c=0;
5    c++;
6    if (c==n) std::cout << c << std::endl;
7    if (n==0) return 1;
8    return n*f(n-1);
9  }
10
11 int main(int c, char*[]) {
12   std::cout << f(c) << std::endl;
13   return 0;
14 }
```

24

Inhalt

Rekursion

... vs. Iteration


```
1  #include<iostream>
2
3  template<typename T>
4  T max_iteratively(size_t n, T x[]) {
5    T s=std::numeric_limits<T>::min();
6    for (size_t i=0;i<n;i++) s=std::max(x[i],s);
7    return s;
8  }
9
10 template<typename T>
11 T max_recursively(size_t n, T x[]) { /* ??? */ }
12
13 int main() {
14   const size_t n=7; int x[n]={1,3,2,42,-9,-32,18};
15   std::cout << max_iteratively(n,x) << std::endl;
16   return 0;
17 }
```

Gesucht ist die Funktion max_recursively mit der gegebenen Signatur so, dass die Funktionalität von max_iteratively mittels Rekursion (in max_recursively) umgesetzt wird.

→ Live

Rekursion

... vs. Iteration


```
1  #include<iostream>
2
3  template<typename T>
4  T max_iteratively(size_t n, T x[]) {
5    T s=std::numeric_limits<T>::min();
6    for (size_t i=0;i<n;i++) s=std::max(x[i],s);
7    return s;
8  }
9
10 template<typename T>
11 T max_recursively(size_t n, T x[]) {
12   if (n==1) return x[0];
13   return std::max(x[n-1],max_recursively(n-1,x));
14 }
15
16 int main() {
17   const size_t n=7; int x[n]={1,3,2,42,-9,-32,18};
18   std::cout << max_iteratively(n,x) << std::endl;
19   return 0;
20 }
```

Inhalt

AUFWACHEN!

Was wird bei Aufruf des Programms mittels `./f.exe 10` ausgegeben?

```
1  #include<iostream>
2  #include<cassert>
3
4  int f(const int n) {
5    if (n==0) return 0;
6    else if (n==1) return 1;
7    else return f(n-2)+f(n-1);
8  }
9
10 int main(int c, char* v[]) {
11   assert(c==2); int n=std::stoi(v[1]); assert(n>=0);
12   for (int i=0;i<n;i++) std::cout << f(i) << std::endl;
13   return 0;
14 }
```

Wach?

Was wird bei Aufruf des Programms mittels `./f.exe 10` ausgegeben?

```
1  #include<iostream>
2  #include<cassert>
3
4  int f(const int n) {
5    if (n==0) return 0;
6    else if (n==1) return 1;
7    else return f(n-2)+f(n-1);
8  }
9
10 int main(int c, char* v[]) {
11   assert(c==2); int n=std::stoi(v[1]); assert(n>=0);
12   for (int i=0;i<n;i++) std::cout << f(i) << std::endl;
13   return 0;
14 }
```

Folge der 10 ersten Fibonacci-Zahlen; Verbesserungspotential?

→ Tafel: Programmlogik

Inhalt

Zur Erinnerung: Z.B. $f(x) = 3 \cdot x^2$

- symbolisch

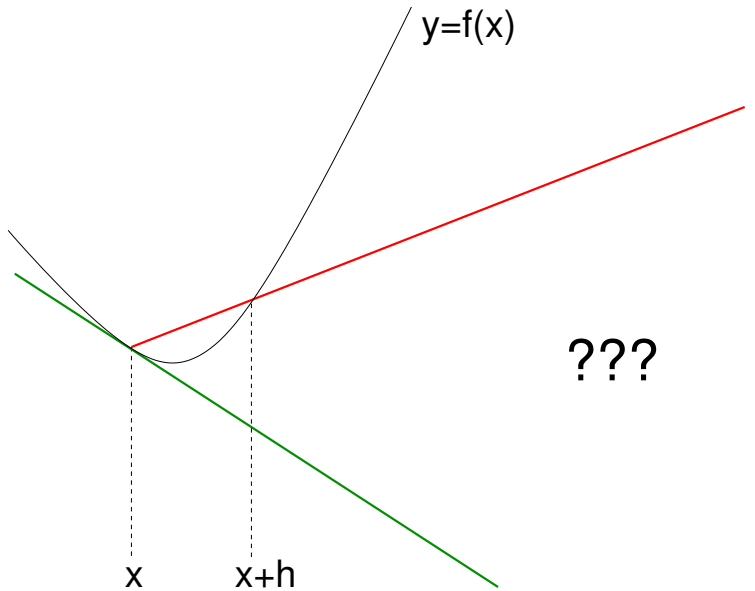
$$f' = \frac{\partial f}{\partial x} = 6x$$

- numerisch (Vorwärtsdifferenzen)

$$f' = \frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

- numerisch (zentrale Differenzen)

$$f' = \frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2 \cdot h}$$



```
1  template<typename T>
2  T f(T x) { return /* z.B. */ exp(x); }
3
4  template<typename T>
5  T df(size_t o, const T& x, T h /* =? */) {
6    if (o==1) return (f(x+h)-f(x-h))/(2*h);
7    return (df(o-1,x+h,h)-df(o-1,x-h,h))/(2*h);
8  }
```

→ Live: Experimente

Inhalt

Zur Erinnerung: Z.B. $f(x) = 2 \cdot x^3 - 42$

Das Newtonverfahren approximiert die Lösung der nichtlinearen Gleichung

$$y = f(x) = 0$$

für einen gebenen Startwert x iterativ mittels

$$x = x - \frac{f(x)}{f'(x)} .$$

Die Iteration endet sobald das Konvergenzkriterium $f(x) \leq \epsilon$ für ein gegebenes $0 < \epsilon \ll 1$ erfüllt ist.

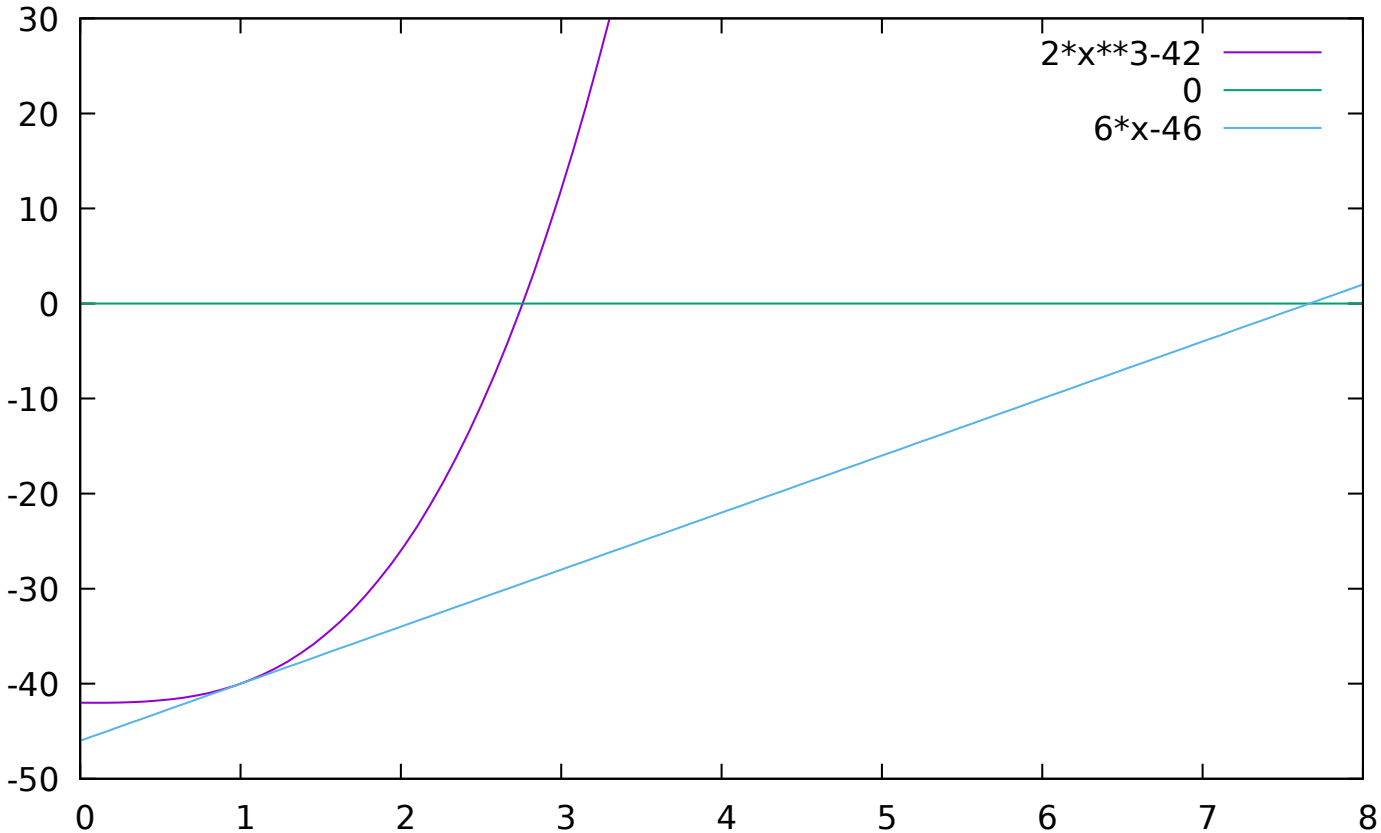
Konvergenz ist im Allgemeinen nicht garantiert.

```
1  #include<iostream>
2  #include<cmath>
3
4  template<typename T>
5  T newton(const T& a, T x, const unsigned int b, const T& c, const T& eps,
6           T (*f)(const T&, const T&, const unsigned int, const T&),
7           T (*dfdx)(const T&, const T&, const unsigned int, const T&)) {
8    T r=f(a,x,b,c); int i=1;
9    while (fabs(r)>eps) {
10     x-=r/dfdx(a,x,b,c);
11     std::cout << i++ << ": " << x << std::endl;
12     r=f(a,x,b,c);
13   }
14   return x;
15 }
```

Gesucht ist eine rekursive Variante der Funktion newton mit der gegebenen Signatur so, dass exakt die gleiche Ausgabe, wie bei Ausführung der gegebenen iterativen Variante, erzeugt wird. → Live

```
1  #include<iostream>
2  #include<cmath>
3
4  template<typename T>
5  T newton(const T& a, T x, const unsigned int b, const T& c, const T& eps,
6                    T (*f)(const T&, const T&, const unsigned int, const T&),
7                    T (*dfdx)(const T&, const T&, const unsigned int, const T&)) {
8    static int i=1;
9    T r=f(a,x,b,c);
10   if (fabs(r)<=eps) return x;
11   x-=r/dfdx(a,x,b,c);
12   std::cout << i++ << ": " << x << std::endl;
13   return newton(a,x,b,c,eps,f,dfdx);
14 }
```

Beispiel: $f(x) = 2 \cdot x^3 - 42$; Startpunkt: $x = 1$; erste Iteration



Zusammenfassung

Prinzip

Rekursion vs. Iteration

Rekursion und Redundanz

Fallstudie: Numerische Differentiation

Fallstudie: Nichtlineare Gleichung mit Newton