

Funktionen und Programmstruktur in C

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K7_C-Funktionen & Programmstruktur.ppt, Seite 1

Funktionen in C

- In diesem Kapitel wollen wir zeigen,
 - ➔ was Funktionen (Unterprogramme) sind
 - ➔ wie man in C Funktionen schreibt
 - ➔ wie ein C-Programm mit Funktionen aufgebaut ist (Programmstruktur)
 - ➔ welchen Gültigkeitsbereich die Variablen in einem C-Programm haben

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K7_C-Funktionen & Programmstruktur.ppt, Seite 2

Funktionen (Unterprogramme) in C

- Was sind Funktionen?
- Funktionen sind in sich abgeschlossene Teilprogramme (**Module**), die Algorithmen mit unterschiedlichen Variablen durchführen.
- Eine Funktion ist **flexibler** als eine Programmschleife: Es können auch Programmteile mit **nicht abzählbaren Variablen wiederholt** durchgeführt werden.
 - In C gibt es **nur Funktionen (functions)**.
- Programmiersprachen unterscheiden **Funktionen (functions)** und **Unterprogramme (subroutines)**.

19.11.2004



Unterprogramme: Wann einsetzen?

- Unterprogramme werden eingesetzt, um
 - ➔ **gleiche Teilprogramme mit unterschiedlichen Variablen** ablaufen zu lassen
 - ➔ **Programmspeicher zu sparen**
 - ➔ einmal geschriebene **Programme öfter einzusetzen**. Dabei können auch Unterprogramme aus Programmbibliotheken eingebunden werden
 - ➔ Programme besser zu strukturieren (**Modulstruktur**).

19.11.2004



Unterprogramme: Welche Vor- und Nachteile?

Vorteile

- ➡ geringerer Programmspeicheraufwand
- ➡ übersichtlicher
- ➡ sicherere Programmierung
- ➡ leichtere Programmänderung (leichter Service)
- ➡ modulare Software
- ➡ weniger Entwicklungsaufwand durch mehrfache Verwendung

Nachteil

- ➡ langsamer

19.11.2004



Funktionsvereinbarung und Funktionsaufruf

- Bei Funktionen unterscheidet man
 - ➡ **Funktionsvereinbarung** (Deklaration) und
 - ➡ **Funktionsaufruf**

• Funktionsvereinbarung

```
➡ float betrag ( float x )
{
  if ( x < 0.f ) return -x ;
  else return x ;
}
```

• Funktionsaufrufe

```
#include <math.h>
#include <stdio.h>
float betrag (float) ;
int main(void)
{
  float y, abs, pi = 4*atan ( 1. ) ;
  ➡ y = sin ( 3*pi/2 ) ;
  ➡ abs = betrag ( y );
  ➡ printf("|sin(3*pi/2)| = %f", abs) ;
  return 0 ; }
```

19.11.2004



Funktionsvereinbarung (1)

- Bei der Kompilierung werden die Programmbefehle in einer Funktionsvereinbarung **nicht ausgeführt**, sondern vom C-Compiler nur **syntaktisch überprüft**.
 - Eine Funktionsvereinbarung besteht aus:
 - dem  **Funktionskopf**: `float betrag (float x)`
 - dem  **Funktionskörper** mit den Programmbefehlen: `{...}`
 - den  **Rücksprunganweisungen** (ggf. optional) mit den Rückgabewerten: `return x` bzw. `return -x`
- ```

 float betrag (float x)
 {
 if (x < 0.f) return -x ; 
 else return x ; 
 }

```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 7

### Funktionsvereinbarung (2)

- Die Programmbefehle in einer Funktionsvereinbarung werden **nicht ausgeführt**, sondern vom C-Compiler nur **syntaktisch überprüft**.
- Die Funktionsvereinbarung erlaubt dem C-Compiler u.a.
  -  die **Überprüfung der korrekten Anzahl der Parameter**
  -  die **Typkonversion der Funktionsvariablen** zu den **Variablen des aufrufendem Programms**.
- Die Funktionsvereinbarung kann auch in einer externen C-Quelltext-Datei (**Header-Datei**) erfolgen.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 8

## Funktionskopf (1)

- Der **→ Funktionskopf** besteht aus:
  - dem **→ Funktionsnamen** *betrag*
  - den **→ formalen Eingangsparametern mit Typangabe** *float x*
  - der **→ Typangabe für den Funktionswert** (Rückgabewert der Funktion) *float*


  
**float betrag ( float x )**
  
 {
   
 if ( x < 0.f ) return -x ;
   
 else return x ;
   
 }

## Eingangs- und Ausgangsparameter (1)

- Die Funktion *betrag* erwartet einen
  - Eingangsparameter** vom Typ *float*
  - und gibt an das **aufrufende Programm**
  - einen Rückgabewert** vom Typ *float* zurück.


  
**float betrag ( float x )**
  
 {
   
 if ( x < 0.f ) return -x ;
   
 else return x ;
   
 }

- Den **Rücksprung ins aufrufende Programm** und die Rückgabe des Funktionswertes nimmt der Befehl *return -x ;* bzw. *return x ;* vor.
- *return ..* ist immer **dynamisch** der letzte Befehl der Funktion.

## Eingangs- und Ausgangsparameter (2)

- Auf die **formalen Eingangsparameter** wird im **Programmkörper** Bezug genommen.
- Mit ihnen und ggf. weiteren in der Funktion vereinbarten Variablen werden die gewünschten Operationen definiert.

```

float betrag (float x)
{
 if (x < 0.f) return -x ;
 else return x ;
}

```

19.11.2004



## Funktionsaufruf

- Beim Funktionsaufruf werden die **formalen Parameter** durch die **aktuellen Parameter** ersetzt.
- Die Eingangsparameter werden **immer kopiert**, d.h. Veränderungen der Eingangsparameter im Unterprogramm haben **keine Auswirkung** auf die entsprechenden Werte im Hauptprogramm: **Wertaufruf (call by value)**

```

#include <math.h>
#include <stdio.h>
int main (void)
{
 float y, abs, pi = 4*atan (1) ;
 y = sin (3*pi/2) ;
 abs = betrag (y) ;
 printf ("|sin (3*pi/2)| = %f", abs) ;
 return 0 ;
}

```

19.11.2004



### Parameterübergabe (1)

- Bei der Parameterübergabe zwischen aufrufendem Programm und Unterprogramm kann man die folgenden **4 Fälle** unterscheiden:

➡ mit Eingangsparametern und ➡ mit Funktionswert

```

 ↓ ↓ ↓
long poshoch (short i1, short i2)
{ /* poshoch berechnet i1^i2 für i1 && i2 > 0 */
 long N = 1 ;
 if (i2 <= 0 || i1 <= 0) return 0 ; /* Fehlerrueckgabe = 0! */
 else
 {
 while (i2--) N *= i1 ;
 return N ;
 }
}

```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 13

### Parameterübergabe (2)

- Parameterübergabe

➡ ohne Eingangsparameter ➡ mit Funktionswert

```

#include <math.h>
...
 ↓ ↓
float pi (void)
{ /* Funktion gibt pi zurueck */
 return 4 * atan (1.) ;
}

```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 14

### Parameterübergabe (3)

- Parameterübergabe

→ mit Eingangsparametern → ohne Funktionswert

Eine → return-Anweisung am Ende der Funktion ist in diesem Fall nicht erforderlich!

```

void wait (long k)
{ /* Warteschleife mit Schleifenparameter */
 if (k < 0) return ; /* Fehler: k < 0: Ruecksprung */
 while (k--) ; /* Warteschleife bis k==0 */
 /* hier ist kein return
 notwendig */
}

```

### Parameterübergabe (4)

- Parameterübergabe

→ ohne Eingangsparameter → ohne Funktionswert

Eine → return-Anweisung am Ende der Funktion ist nicht erforderlich!

```

void wait (void)
{ /* feste Warteschleife ohne Schleifenparameter */
 long k;
 printf(" Wartezeit > 0 (long) ?:"); scanf ("%ld",&k);
 if (k < 0)
 {
 printf (" Fehler: Wartezeit < 0!");
 return ; /* Fehler: k < 0: Ruecksprung */
 }
 while (k--) ; /* Warteschleife bis k==0 */
 /* hier kein return notwendig */
}

```

## Funktionsaufruf einer Funktion ohne Eingangsparameter

- Beispiel: Aufruf einer Funktion  
**ohne Eingangsparameter** und ➡ **ohne Funktionswert**

```
include <conio.h> /* Bibliothek conio.h einbinden */
...
int main (void)
{
 ...
 _getch () ; /* Funktion _getch (= get character)
 aus der Bibliothek conio.h:
 auf Tasteneingabe warten */
 ...
}
```

19.11.2004



## Parameterübergabe (5)

- Sonderfälle der Parameterübergabe:  
➡ **kein Funktionstyp angegeben**
- Ist **kein Funktionstyp** angegeben, so nimmt der C-Compiler **int** an.

```
func (int m) = int func (int m)
{

}
{

}
```

19.11.2004



### Parameterübergabe (6)

- Sonderfälle der Parameterübergabe:

➔ **kein Funktionswert angegeben**

- Es ist zulässig - aber nicht sinnvoll – **keinen Rückgabewert** anzugeben.  
In diesem Fall erzeugt der C-Compiler eine Warnung.
- Es wird **kein Funktionsrückgabewert erzeugt**. 💣\*

```

long func1 (int m) int func2 (int m)
{
 ...
 return ; {
 ...
 ...
 }

```

Diagramm: Ein roter Pfeil zeigt von der leeren Rückgabewert-Position in `func1` nach unten, und ein weiterer roter Pfeil zeigt von der leeren Rückgabewert-Position in `func2` nach rechts.

### Funktionsprototypen (1)

- In C ist es üblich, zu Beginn des Programms nur die  
➔ **Funktionsprototypen**  
statt der kompletten Unterprogramme anzugeben.
- Das komplette Unterprogramm (Funktionsdefinition) wird  
später, ggf. in einem separaten Quelltext geschrieben.
- Die Typen der Parameter müssen **im Prototyp und in der  
Funktionsdefinition komplett übereinstimmen**.
- Es ist **nicht erforderlich**, ➔ **formale Parameter** anzugeben.

↓

➔ **double func ( float ) ;** /\* Prototyp der Funktion *func*,  
die einen *float*-Eingangs-  
parameter erwartet und einen  
*double*-Rückgabewert erzeugt. \*/

### Funktionsprototypen (2)

- Die **Prototypendeklaration** erlaubt dem C-Compiler z.B.
  - ➔ die **Überprüfung der korrekten Anzahl der Parameter**
  - ➔ die **Typkonversion** zwischen Funktion und aufrufendem Programm.
  - ➔ die Ausgabe von **Fehlermeldungen** während der Syntaxprüfung des Programms.
- Auch die Prototypendeklaration kann in einer externen C-Quelltext-Datei erfolgen.  
Sie wird dann dem C-Präprozessor z.B. mit Hilfe der
  - ➔ **#include < ... .h >** - Anweisung bekannt gegeben.
- 💡: Wenn eine **#include**-Anweisung fehlt, so generieren C-Compiler **nur eine Warnung!**

19.11.2004



### Funktionsprototypen (3)

- Die Deklaration von **Prototypen** hat einige Vorteile:
  - ➔ die Funktionen müssen **nicht vorab** geschrieben werden.
  - ➔ die Programmierung der Funktionen kann zwischen mehreren Entwicklern aufgeteilt werden:  
**modulare Programmierung.**
  - ➔ zur Koordinierung der Entwicklung sollten die Programmentwickler die **Module und die Schnittstellen in jedem Fall vorab definieren.**

19.11.2004



### Programmbeispiel mit Prototypenvereinbarung

```
#include <math.h>
#include <stdio.h>

➡ float betrag (float); ⬅

int main(void)
{
 float abs, pi = 4*atan (1.) ;
 abs = betrag (sin (3*pi/2)) ;
 printf ("|sin (3*pi/2)| = %f", abs) ;
 return 0 ;
}

float betrag (float x)
{
 if (x < 0.f) return -x ;
 else return x ;
}
```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 23

### iterative und rekursive Unterprogramme

- Funktionen können häufig sowohl  
➡ **iterativ** als auch  
➡ **rekursiv**  
geschrieben werden.

- iterative Funktion

$$2^N = 2 * 2 * 2 * \dots * 2$$



iterativ:  
Schleife **N** mal  
wiederholen

- rekursive Funktion

$$2^N = \begin{cases} 1 & \text{für } N = 0 \\ 2 * 2^{N-1} & \text{für } N > 0 \end{cases}$$



rekursiv:  
bei jedem neuen Aufruf: **\*2**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 24

### iteratives Unterprogramm

- iterative Funktion

$$i^N = i * i * i * \dots * i$$



iterativ: N mal  
**long poshoch (short i, short N)**  
**{** */\* poshoch berechnet 'i hoch N'*  
     **für** N && i > 0 **iterativ** *\*/*  
     **long** R = 1L ;  
     **if** ( N <= 0 || i <= 0 ) **return** 0 ; */\* Fehlerrückgabe = 0! \*/*  
     **else**  
     {  
         **while** ( N-- ) R \*= i ;  
     **return** R ;  
     }  
**}**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
 Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
 K7\_C-Funktionen & Programmstruktur.ppt, Seite 25

### rekursives Unterprogramm

- rekursive Funktion

$$a^j = \begin{cases} 1 & \text{für } j=0 \\ a * a^{j-1} & \text{für } j>0 \end{cases}$$



rekursiv:  
 bei jedem neuen Aufruf: \*a

**long double power ( float a, short j )**  
**{** */\* power berechnet 'a hoch j' rekursiv \*/*  
     **if** ( j == 0 ) **return** 1.L ;  
     **else**  
     ➔ **return** a \* power ( a, j-1 ) ;  
     */\* rekursiver Aufruf von power \*/*  
**}**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
 Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
 K7\_C-Funktionen & Programmstruktur.ppt, Seite 26

## Bibliotheksfunktionen

- In C gibt es umfangreiche **Bibliotheken**.
- Sie Enthalten **vorcompilierte Funktionen** für vielfältige Aufgaben.
- Häufig werden **mathematische Funktionen** benötigt.
  - Sie sind in der Bibliothek **math.h** enthalten.
- Möchte man die mathematischen Funktionen von **math.h** verwenden, so muss man die Bibliothek dem Compiler im Kopf des C-Programms mit der Anweisung: **#include <math.h>** bekannt geben.

Es gibt weitere wichtige Bibliotheken, z.B. **string.h** zur Stringverarbeitung.

19.11.2004



## Bibliotheksfunktionen in **math.h**

- Alle Funktionsargumente in **math.h** sind vom Typ **double**.

| Funktion                                         | Erläuterung                                                       |
|--------------------------------------------------|-------------------------------------------------------------------|
| <code>double sin ( double x )</code>             | Sinus, x im Bogenmaß                                              |
| <code>double cos ( double x )</code>             | Kosinus, x im Bogenmaß                                            |
| <code>double tan ( double x )</code>             | Tangens, x im Bogenmaß                                            |
| <code>double asin ( double x )</code>            | Arcus Sinus, $-1 \leq x \leq +1$ , $-\pi/2 \leq asin \leq +\pi/2$ |
| <code>double acos ( double x )</code>            | Arcus Kosinus, $-1 \leq x \leq +1$ , $0 \leq acos \leq \pi$       |
| <code>double atan ( double x )</code>            | Arcus Tangens, $-\pi/2 \leq atan \leq +\pi/2$                     |
| <code>double atan2 ( double y, double x )</code> | Arcus Tangens, $atan(y/x)$ $-\pi \leq atan2 \leq \pi$             |
| <code>double sinh ( double x )</code>            | Sinus Hyperbolicus                                                |
| <code>double cosh ( double x )</code>            | Kosinus Hyperbolicus                                              |
| <code>double tanh ( double x )</code>            | Tangens Hyperbolicus                                              |
| <code>double exp ( double x )</code>             | $e^x$                                                             |
| <code>double log ( double x )</code>             | natürlicher Logarithmus $\ln(x)$ , $x > 0$                        |
| <code>double log10 ( double x )</code>           | dekadischer Logarithmus $\log(x)$ , $x > 0$                       |
| <code>double pow ( double x, double y )</code>   | $x^y$                                                             |
| <code>double sqrt ( double x )</code>            | $\sqrt{x}$                                                        |
| <code>double ceil ( double x )</code>            | aufrunden                                                         |
| <code>double floor ( double x )</code>           | abrunden                                                          |
| <code>double fabs ( double x )</code>            | $ x $                                                             |

19.11.2004



### Programmstruktur

- Typischer Aufbau eines C-Programms

|                                                                                                                                                                                                                                            |                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include .... float func1 (int, long); int func2 (int); .... int i, j, k; long l1, l2; .... int main (void) { int k; ... } .... float func1 (int j, long l3) { long l1;   int i, n; ...} .... int func2 (int i) { long l1; ...}</pre> | <p>← Bibliotheken einbinden</p> <p>← Prototypen deklarieren</p> <p>← globale Variable definieren</p> <p>← Hauptprogramm main</p> <p>← Funktionen<br/>func1, func2, ...</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 29

19.11.2004

### Gültigkeitsbereich von Variablen (1)

- Variable in **verschiedenen Funktionen** dürfen **dieselben Namen** haben.
- Namen für Variablen und Prototypen sind dem C-Compiler **ab der Stelle bekannt, an der der Compiler auf sie trifft**.
- Formale Parameter und in Funktionen vereinbarte Variable (lokale Variable) werden am Ende der Funktion wieder vergessen.
- **Namen können verdeckt werden:** Globale Variable mit demselben Namen werden von formalen Parametern und von in Funktionen vereinbarten Variablen **abgeschattet**.

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 30

19.11.2004

### Gültigkeitsbereich von Variablen (2)

```
#include
...
float func1 (int, long);
int func2 (int);
...
int i, j, k;
long l1, l2;
...
int main (void)
{ int k; ...
}
...
float func1 (int j, long l3)
{ long l1;
 int i, n; ...}
...
int func2 (int i)
{ long l2; ...}
```

Diese **globalen Variablen** sind dem Hauptprogramm und allen Funktionen bekannt, sofern sie nicht durch lokale Variable verdeckt sind

Die **lokale Variable k** verdeckt die **globale Variable k**

Die **lokalen Variablen j, l1 und i** verdecken die **globalen Variablen j, l1 und i**. **l3 und n** sind nur in **func1** bekannt.

**l2 und i** verdecken **l2 und i**.



### Gültigkeitsbereich von Variablen (3)

- Wir haben die **Vereinbarung von Variablen** bisher relativ strikt gehandhabt:  
alle Variablen wurden bisher **am Anfang eines Programms oder einer Funktion** vereinbart.
- C bietet aber folgende weitere Möglichkeiten zur Vereinbarung von Variablen:
  - Externe Variable müssen **nicht am Anfang** stehen.
  - Variable dürfen **zu Anfang jeder Verbundanweisung** definiert werden.

Auch dabei gilt:

**neue lokale Variable verdecken externe (globale) Variable gleichen Namens.**



### Gültigkeitsbereich von Variablen (4)

- Externe Variable müssen **nicht am Anfang** stehen.

```
float func1 (int j, long l3);
{
 long l1;
 int i, n;
 ...
}
...
long double dou1; ←
int func2 (int i);
{
 long l2;
 ...
}
```

/\* Die globale Variable **dou1**  
ist ab hier bekannt. \*/

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 33

### Gültigkeitsbereich von Variablen (5)

- In C dürfen Variable  
**am Anfang jeder Verbundanweisung**  
vereinbart werden.

```
#include <stdio.h>
int main(void)
{
 short i = 5;
 printf ("i = %d\n", i);

 {
 int i = 10; ←
 printf ("i = %d\n", i);
 }

 printf ("i = %d\n", i); ←
 return 0;
}
```

/\* Die Variable **i** wird hier  
neu definiert.  
Die Neudefinition gilt nur  
für die Verbundanweisung  
**{ ... }**.  
**i** wird verdeckt\*/

/\* Ab hier ist  
**i** wieder gültig\*/

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 34

### Globale Variable (1)

- **Globale Variable sind fehlerträchtig**, weil sie
  - ➔ bei versehentlicher erneuter Deklaration verdeckt werden
  - ➔ in Funktionen versehentlich überschrieben werden können.
- Diese unerwünschten **Seiteneffekte** können zu Fehlern in ganz Programmteilen führen.
- Fehler durch Seiteneffekte sind schwierig zu finden.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 35

### Globale Variable (2)

- **Globale Variable sind fehlerträchtig**, weil sie
  - ➔ bei versehentlicher erneuter Deklaration verdeckt werden
- Die Fehler durch diese Seiteneffekte sind schwierig zu finden.
- Sicherheitsmaßnahmen:
  - ➔ Man verwendet globale Variable **nur dann, wenn man sie nicht vermeiden kann!**
  - ➔ Man deklariert globale Variable **immer zu Anfang des Programms!**
  - ➔ Man bezeichnet nur die globalen Variablen immer mit einem Präfix z.B. mit **g\_... !**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 36

## **iterative und rekursive Funktionen**

Schreiben Sie ein C-Programm zur Berechnung von

$$y = a \cdot 2^i \quad i \geq 0$$

1. als iterative Funktion
2. als rekursive Funktion

Testen Sie die Funktion mit Hilfe eines geeigneten Hauptprogramms!



## **iterative und rekursive Funktionen (1)**

### • iterative Form:

```
#include <stdio.h>
float it (unsigned int i, float a)
{
 float y = 1.f ;
 int n ;
 for (n = 1; n <= i; n++) y *= 2 ;
 y = a * y ;
 return y ;
}

int main (void)
{
 int i ;
 float a ;
 printf ("Eingabe: int i, float a ? ") ;
 scanf ("%d, %f", &i, &a) ;
 printf ("%d\n", it (i, a)) ;
 return 0 ;
}
```



## ✓ iterative und rekursive Funktionen (2)

### • rekursive Form:

```
#include <stdio.h>
float rek (unsigned int i, float a)
{
 if (i == 0)
 return a ;
 else
 return 2 * rek (i-1, a) ;
}

int main (void)
{
 int i ;
 float a ;
 printf ("Eingabe: int i, float a ? ") ;
 scanf ("%d, %f", &i, &a) ;
 printf ("%d\n", rek (i, a)) ;
 return 0 ;
}
```



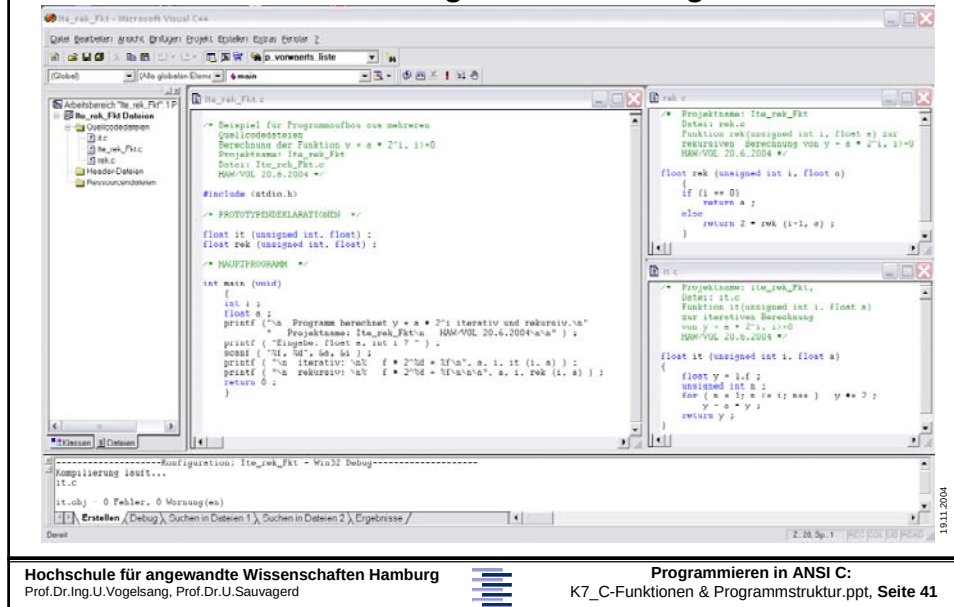
## Programmaufbau in MSVC++

- Um das Debuggen von komplizierteren Programmen zu erleichtern, soll in Zukunft **jedes Unterprogramm in eine getrennte Quellcodedatei** geschrieben werden.
- Wie bisher wird das **Hauptprogramm** **int main (void)** in einer **Quellcodedatei** unter mit dem Projektnamen gespeichert: z.B. **Projekt.c**.  
Zur besseren Übersicht wird der **Name des Projektes gleich dem der Hauptprogrammquelldatei gewählt** und mindestens am Anfang **groß geschrieben**.
- Die **Dateinamen der Funktionen sind mit den Funktionsnamen identisch** und erhalten ebenfalls die Dateierweiterung **c**: z.B. **func1.c**.  
**Alle Funktionsnamen werden klein geschrieben.**



Beispiel für Programmaufbau in *MSVC++* (1)

- Eine Variante der vorhergehenden Übung:



Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd

Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 41

Beispiel für Programmaufbau in *MSVC++* (2)

- Damit der Compiler beim Kompilieren des Hauptprogramms die Datentypen der Funktionen prüfen kann, benötigt er **Prototypenvereinbarungen**.
- Die Prototypen haben **den selben Aufbau und die selben Datentypen** wie die Funktionen.  
**Formale Namen** müssen nicht vergeben werden.
- Im Gegensatz zur Funktionsvereinbarung werden die **Prototypenvereinbarungen immer mit Semikolon** abgeschlossen.
- Zu Anfang jeder Datei werden jeweils **nur diejenigen Bibliotheken eingebunden, in der Quelldatei benötigt werden**.

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd

Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 42

### Beispiel für Programmaufbau in MSVC++ (3)

- Quellcode des Hauptprogramms **lte\_rek\_Fkt.c** (1):

```
/* Beispiel für Programmaufbau aus mehreren Quellcodedateien
 Berechnung der Funktion $y = a * 2^i$, $i \geq 0$
 Projektname: lte_rek_Fkt
 Datei: lte_rek_Fkt.c
 HAW / VGL 20.6.2004 */
```

```
/* BIBLIOTHEKEN */
```

```
#include <stdio.h>
```

```
/* PROTOTYPENDEKLARATIONEN */
```

```
float it (unsigned int, float) ;
float rek (unsigned int, float) ;
```

```
/* Fortsetzung auf nächster Folie ... */
```

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 43

19.11.2004

### Beispiel für Programmaufbau in MSVC++ (4)

- Quellcode des Hauptprogramms **lte\_rek\_Fkt.c** (2):

```
/* ... Fortsetzung von vorheriger Folie */
```

```
/* HAUPTPROGRAMM */
```

```
int main (void)
{
 int i ;
 float a ;
 printf ("\n Programm berechnet $y = a * 2^i$ iterativ und rekursiv.\n"
 " Projektname: lte_rek_Fkt\n HAW/VGL "
 " 20.6.2004\n\n") ;
 printf ("Eingabe: float a, int i ? ") ;
 scanf ("%f, %d", &a, &i) ;
 printf ("\n iterativ: \n% f * 2^%d = %f\n", a, i, it (i, a)) ;
 printf ("\n rekursiv: \n% f * 2^%d = %f\n\n", a, i, rek (i, a)) ;
 return 0 ;
}
```

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 44

19.11.2004

**Beispiel für Programmaufbau in MSVC++ (5)**

- Quellcode in der Datei **it.c**:

```
/* Projektname: lte_rek_Fkt Datei: it.c
 Funktion it (unsigned int i, float a)
 zur iterativen Berechnung von
 $y = a * 2^i$, $i \geq 0$
 HAW/VGL 20.6.2004 */
```

```
float it (unsigned int i, float a)
{
 float y = 1.f ;
 unsigned int n ;
 for (n = 1; n <= i; n++) y *= 2 ;
 y = a * y ;
 return y ;
}
```

19.11.2004

**Beispiel für Programmaufbau in MSVC++ (6)**

- Quellcode in der Datei **rek.c**:

```
/* Projektname: lte_rek_Fkt Datei: rek.c
 Funktion rek (unsigned int i, float a)
 zur rekursiven Berechnung von
 $y = a * 2^i$, $i \geq 0$
 HAW/VGL 20.6.2004 */
```

```
float rek (unsigned int i, float a)
{
 if (i == 0)
 return a ;
 else
 return 2 * rek (i-1, a) ;
}
```

19.11.2004



## Beispiel für Programmaufbau in MSVC++ (7)

- Programmausdruck:

```

C:\Dokumente und Einstellungen\Ulrich\Desktop\Aktenkoffer\MS Vis...
Programm berechnet y = a * 2^i iterativ und rekursiv.
Projektname: Ite_rek_Fkt
HAU/UGL 20.6.2004

Eingabe: float a, int i ? 2.25, 3

iterativ:
2.250000 * 2^3 = 18.000000

rekursiv:
2.250000 * 2^3 = 18.000000

Press any key to continue

```

## Beispiel für Programmaufbau in MSVC++ (8)

- In MSVC ist bei getrennten Quellcodedateien eines Projekts folgendes zu beachten:
- Getrennt geschriebene Quellcodedateien können in MSVC, wenn **im Arbeitsbereichfenster** Quellcodedateien (sources) aktiviert ist, im Menü **Projekt (project) → Dem Projekt hinzufügen (add to project) → Dateien (files)** hinzugefügt werden.
- Das Betätigen der **Entf**-Taste bei einer **im Arbeitsbereichfenster** markierten **Quellcodedatei** entfernt die Datei nur aus dem Projekt. Sie **bleibt im Projektverzeichnis erhalten**.

**Beispiel für Programmaufbau in MSVC++ (9)**

- In MSVC ist bei getrennten Quellcodedateien eines Projekts folgendes zu beachten:
- Das Betätigen der -Schaltleiste (**Strg+F7**) bewirkt **nur** das Übersetzen der **im Editor-Fenster aktuell markierten Datei**.
- **Alle Dateien im Projekt** werden mit der -Schaltleiste (**F7**) übersetzt.
- Beim Übersetzen werden im Nachrichtenfenster (**message window**) die Fehler getrennt nach **Quellcodedateien** aufgelistet.

19.11.2004

** Lokale und globale Variable (1)**

Gegeben ist ein Programm, das geschrieben wurde, um die Anzahl der Funktionsaufrufe zu zählen.

1. Ist das Programm korrekt ?  
Falls es Fehler enthält, geben Sie den Fehler und die Fehlerursache an!
2. Verbessern Sie das Programm!

19.11.2004





## Lokale und globale Variable (2)

```
#include <stdio.h> /* Bibliothek fuer printf(...) einbinden */
short count (short); /* Prototypenvereinbarung */
int main (void) /* Hauptprogramm */
{
 short i, j = 0; /* Variablendeklaration */
 for (i = 1 ; i < 5 ; i++)
 {
 j = count (j); /* count 4mal aufrufen */
 printf ("\n i = %d j= %d\n", i, j); /* i, j ausgeben */
 }
 return 0; /* Ruecksprung ins Betriebssystem */
}

short count (short i)
 /* count soll die Anzahl der Funktionsaufrufe zaehlen */
{
 short m ;
 if (i == 0) m = 1; else m++; /* Funktionsaufrufe zaehlen */
 return m ;
}
```



## ✓ Lokale und globale Variable (1)

### • Fehler

Die lokale Variable **m** in der Funktion **count** wird beim **Verlassen der Funktion wieder vergessen**;  
**m** kann daher die Anzahl der Funktionsaufrufe **nicht zählen**.

Lokale Variable werden normalerweise nicht initialisiert: Für **i ≠ 0** ist **m** nicht initialisiert!

### • Abhilfe 1:

Parameter der Funktion **direkt** zurückgeben!  
Mit dieser Version können **nur die Aufrufe einer Funktion** gezählt werden.



## ✓ Lokale und globale Variable (2)

```
#include <stdio.h> /* Bibliothek fuer printf(...) einbinden */
/* Variante ohne globale Variable */
short count_1 (short); /* Prototypenvereinbarung */

int main (void) /* Hauptprogramm */
{
 short i, j = 0; /* Variablendeklaration */
 for (i = 1; i < 5; i++)
 {
 j = count_1 (j); /* count_1 4mal aufrufen*/
 printf ("\n i = %d j = %d\n", i, j); /* i, j ausgeben*/
 }
 return 0; /* Ruecksprung ins Betriebssystem */
}

short count_1 (short i)
/* count_1 zaehlt die Anzahl der Funktionsaufrufe */
{
 if (i == 0) i = 1; else i++; /* Funktionsaufrufe zaehlen */
 return i;
}
```



## ✓ Lokale und globale Variable (3)

• Fehler

Die lokale Variable **m** in der Funktion **count** wird beim  
**Verlassen der Funktion wieder vergessen;**  
**m** kann daher die Anzahl der Funktionsaufrufe **nicht**  
**zählen.**

Lokale Variable werden normalerweise nicht  
initialisiert: Für **i ≠ 0** ist **m** nicht initialisiert!

• Abhilfe 2:

Zählvariable **m** als **globale Variable g\_m** definieren!  
Sie kann dann auch **in anderen Funktionen**  
**hochgezählt werden!**



## ✓ Lokale und globale Variable (4)

```

#include <stdio.h> /* Bibliothek fuer printf(...) einbinden */
short count_g (void) ; /* Prototypenvereinbarung */
int g_m = 0 ; /* Zaehlvariable m global definieren! */
int main (void) /* Hauptprogramm */
{
 int i ; /* Variablendeklaration */
 for (i = 1; i < 5; i++)
 {
 j = count_g () ; /* count_g 4mal aufrufen*/
 printf ("\n i = %d j= %d\n", i, j) ; /* i, j ausgeben*/
 }
 return 0 ; /* Ruecksprung ins Betriebssystem */
}
short count_g (void)
{
 /* count_g zaehlt die Anzahl der Funktionsaufrufe */
 j++ ; /* Funktionsaufrufe zaehlen */
 return j ;
}

```



## BildschirmAusgabe löschen

- MSVC 6.0 stellt das Kommando `clrscr()` (BorlandC) zum **Löschen des Bildschirms** nicht zur Verfügung.
- Man muss statt dessen die Betriebssystem-Funktion  
     ➔ `cls` in der Bibliothek `stdlib.h` verwenden.
  - **Unmittelbar vor dem Aufruf** von  
     ➔ `system ( "cls" ) ;`  
     muss die Tastatureingabe mit  
     ➔ `fflush ( stdin ) ;`  
     gelöscht werden.

Beispiel:

```

#include <stdlib.h>
/* ... */
fflush (stdin) ; /* Tastaturpuffer loeschen */
system ("cls") ; /* DOS-Betriebssystemkommando: Bildschirm
 loeschen */

```



**Bildschirmausgabe verzögern: *Sleep (..)***

- Unter *MSVC 6.0* steht das Kommando *delay (..)* (*BorlandC*) zur **Verzögerung der Bildschirmausgabe** nicht zur Verfügung.
- Man muss statt dessen die Funktion *Sleep (..)* in der Bibliothek *windows.h* verwenden.

➡ *Sleep ( Millisekunden )*

**verzögert die Bildschirmausgabe** um eine Wartezeit, die in Millisekunden angegeben wird.

➡ *Millisekunden* ist eine **32-Bit unsigned Integerzahl**.

Beispiel:

```
#include <windows.h>
/* ... */
Sleep (1000) ; /* verzogert Bildschirmausgabe um 1 Sekunde */
```

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 57

19.11.2004

**Tastaturabfrage im Hintergrund: *\_kbhit (..)***

- Unter *MSVC 6.0* steht das Kommando *\_kbhit ( )* zur Verfügung, um die Tastatur **im Hintergrund** abzufragen.
- Um *\_kbhit ( )* zu benutzen, muss man die Bibliothek *conio.h* einbinden.
- Das Beispiel auf der folgenden Folie veranschaulicht die Funktion von *\_kbhit ( )*.

Hochschule für angewandte Wissenschaften Hamburg  
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:  
K7\_C-Funktionen & Programmstruktur.ppt, Seite 58

19.11.2004

### Beispiel für `_kbhit (..)`

```

/* HAW VGL 1.12.2003, Dateiname Test_kbhit.c */
/* Eine Schleife wird durchlaufen bis eine Taste gedrueckt ist.
In diesem Falle gibt _kbhit einen Wert ≠ 0 zurueck: ein Zeichen
wertet im Sandardeingabepuffer darauf abgeholt zu werden . Das
Programm kann _getch () oder _getche () aufrufen, um den Wert im
Puffer abzuholen. Der Ablauf wird mit Sleep (..) verlangsamt.*/
#include <conio.h>
#include <stdio.h>
#include <windows.h>

void main (void)
{
 /* Display-Ausgabe bis Taste gedrueckt ist */
 while (!_kbhit ())
 {
 Sleep (500) ;
 _cputs("Bitte Taste druecken!\n") ; }
 /* _getch benutzen um das Zeichen im Puffer zu loeschen. */
 printf ("\n Taste '%c'\n gedrueckt!", _getch ()) ;
 _getch () ;
}

```

19.11.2004



### BildschirmAusgabe des Beispielprogramms für `_kbhit (..)`

```

Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!
Bitte Taste druecken!

Taste 'v'gedrueckt!

```

19.11.2004

