9. Funktionen Teil II

Prof. Dr. Markus Gross
Informatik I für D-ITET (WS 03/04)

- Inline Funktionen
- Referenz-Variablen
- Pass by Reference
- Funktionsüberladung
- Templates

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Inline Funktionen

- *Inline* Funktionen dienen der Programmbeschleunigung
- Anatomie eines Funktionsaufrufs
  - Speichern der aktuellen Adresse
  - Sichern der lokalen Variablen auf dem Stack
  - Abarbeiten der Instruktionen
  - Rücksprung
  - Wiederherstellung der Variablen vom Stack
- Bei Inline-Funktionen wird Code direkt eingesetzt
- Mehraufwand für Sprung gespart
  - Lohnt sich nur für sehr kurze Funktionen
- Verwendung des Keywords **`inline`**

 3 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_1:Inline-Funktionen

```
// inline.cpp -- use an inline function

#include <iostream>
using namespace std;

// an inline function must be defined before first use

inline double square(double x) { return x * x; }

int main()
{
    double a, b;
    double c = 13.0;

    a = square(5.0);
    b = square(4.5 + 7.5);  // can pass expressions
    cout << "a = " << a << ", b = " << b << "\n";
    cout << "c = " << c;
    cout << ", c squared = " << square(c++) << "\n";
    cout << "Now c = " << c << "\n";
    return 0;
}
```

Prototyp + Definition

Aufrufe der Funktion

 4 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Referenzen

- Bisher Referenzen in Form von Pointern verwendet
- C++ bietet *Referenzvariablen* auf höherer Abstraktionsebene
- Eine Referenz ist ein anderer (fester) Name für die gleiche Variable
  - Erlauben Zugriff auf Variable
  - Können als Funktionsargumente verwendet werden

- Elegante Alternative zu Pointern
- Werden mittels des **&**-Operators definiert
- Beispiel:

```
int rats;
int & rodents = rats; //Initialisierung einer
   Referenz auf rats
```

- Lesender und schreibender Zugriff erlaubt

 5 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_2: Referenz

```
// firstref.cpp -- defining and using a reference

#include <iostream>
using namespace std;

int main()
{
    int rats = 101;
    int & rodents = rats;  // rodents is a reference

    cout << "rats = " << rats;
    cout << ", rodents = " << rodents << "\n";
    rodents++;
    cout << "rats = " << rats;
    cout << ", rodents = " << rodents << "\n";

// some implementations require type casting the following
// addresses to type unsigned
    cout << "rats address = " << &rats;
    cout << ", rodents address = " << &rodents << "\n";
    return 0;
}
```

Definition und Initialisierung

Schreibzugriff

 6 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Referenzen und Pointers

- Nach Initialisierung kann Referenz genau wie eine Variable verwendet werden
- Referenz hat Aehnlichkeit mit Pointer

```
int rats = 10;
int & rodents = rats; // Referenz
int * prats = &rats; // Pointer
```

- Unterschiede:
  - Bei Zugriff muss Pointer explizit dereferenziert werden
  - Referenz ist es bereits
  - Referenz muss bei Definition initialisiert werden
  - Kann nachträglich nicht mehr auf andere Variable gesetzt werden
- Eine Referenz verhält sich wie ein *konstanter, dereferenzierter* Pointer
  - Analogon stimmt nicht immer !

 7 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_3: Verwendung der Ref.

```
// secref.cpp -- defining and using a reference

#include <iostream>
using namespace std;

int main()
{
    int rats = 101;
    int & rodents = rats;   // rodents is a reference

    cout << "rats = " << rats;
    cout << ", rodents = " << rodents << "\n";

    cout << "rats address = " << &rats;
    cout << ", rodents address = " << &rodents << "\n";

    int bunnies = 50;
    rodents = bunnies;      // can we change the reference?
    cout << "bunnies = " << bunnies;
    cout << ", rats = " << rats;
    cout << ", rodents = " << rodents << "\n";

    cout << "bunnies address = " << &bunnies;
    cout << ", rodents address = " << &rodents << "\n";
    return 0;
}
```

Definition und Initialisierung

Zuweisung

 8 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Referenzen

- Zuweisungen einer Referenz ändern die Originalvariable
- Konstanter, dereferenzierter Pointer

```
int rats = 10;
int & rodents = rats; // Referenz
int * prats = &rats; // Pointer
```

  - Initialisierung: Referenz entspricht Adresse
  - Zuweisung: Referenz entspricht Inhalt

- Referenzen können als Funktionsargumente verwendet werden (*call by reference*)
- Wesentlich eleganter als Pointer
- Referenz wird in Funktion genau wie eine lokale Variable verwendet

 9 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_4: Referenz als Argument I

```
// swaps.cpp -- swapping with references and with pointers
#include <iostream>

using namespace std;
void swapr(int & a, int & b);   // a, b are aliases for ints
void swapp(int * p, int * q);   // p, q are addresses of ints
void swapv(int a, int b);
        // a, b are new variables
int main()
{
    int wallet1 = 300;
    int wallet2 = 350;

    cout << "wallet1 = $" << wallet1;
    cout << " wallet2 = $" << wallet2 << "\n";

    swapr(wallet1, wallet2);   // pass variables

    cout << "wallet1 = $" << wallet1;
    cout << " wallet2 = $" << wallet2 << "\n";

    swapp(&wallet1, &wallet2); // pass addresses of variables

    cout << "wallet1 = $" << wallet1;
    cout << " wallet2 = $" << wallet2 << "\n";

    swapv(wallet1, wallet2);   // pass values of variables

    cout << "wallet1 = $" << wallet1;
    cout << " wallet2 = $" << wallet2 << "\n";
    return 0;
}
```

Referenz

Pointer

Wert

 10 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_4: Referenz als Argument II

```
void swapr(int & a, int & b)    // use references
{
    int temp;

    temp = a;       // use a, b for values of variables
    a = b;
    b = temp;
}

void swapp(int * p, int * q)    // use pointers
{
    int temp;

    temp = *p;      // use *p, *q for values of variables
    *p = *q;
    *q = temp;
}

void swapv(int a, int b)        // try using values
{
    int temp;

    temp = a;      // use a, b for values of variables
    a = b;
    b = temp;
}
```

 11

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Referenzen auf Strukturen

- Sollen Referenzen in Funktionen nicht geändert werden, so müssen sie **`const`** gesetzt werden
- Referenzen können auf Structs gesetzt werden
  - Sowohl als Funktionsargument, als auch als Rückgabetyp
- Zuweisung eines Wertes zu einer Funktion wird damit möglich
  - Funktion kann auf linker Seite der Zuweisung stehen
- Nachstehendes Beispiel:

```
use(looper) = morph;
// entspricht
use(looper) // returns reference
looper = morph; // copies morph to looper
```

- Referenz wird zurückgegeben

 12

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_5: Structs I

```
// strtref.cpp -- using structure references
#include <iostream>
using namespace std;
struct sysop
{
    char name[26];
    char quote[64];
    int used;
};

sysop & use(sysop & sysopref);  // function with a reference
     return type

int main()
{
// NOTE: some implementations require using the keyword static
// in the two structure declarations to enable initialization
    sysop looper =
    {
        "Rick \"Fortran\" Looper",
        "I'm a goto kind of guy.",
        0
    };

    use(looper);           // looper is type sysop
    cout << looper.used << " use(s)\n";

    use (use(looper));     // use(looper) is type sysop
    cout << looper.used << " use(s)\n";
```

Prototyp

 13 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_5: Structs II

Zuweisung

```
sysop morf =
    {
        "Polly Morf",
        "Polly's not a hacker.",
        0
    };
    use(looper) = morf;     // can assign to function!
    cout << looper.name << " says:\n" << looper.quote << '\n';
    return 0;
}

// use() returns the reference passed to it
sysop & use(sysop & sysopref)
{
    cout << sysopref.name << " says:\n";
    cout << sysopref.quote << "\n";
    sysopref.used++;
    return sysopref;
}
```

 14 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Default Argumente

- *Default Argumente* sind Werte für Funktionsargumente, die automatisch verwendet werden, falls sie nicht anders gesetzt werden
- Werden in C++ im Funktionsprototypen gesetzt
- Wenn Argument bei Funktionsaufruf gesetzt, überschreibt es den Default-Wert
- Beispiel:

```
int harpo(int n, int m = 4, int j = 5);
...
harpo(2); // entspricht harpo(2,4,5)
harpo(2,7); // entspricht harpo(2,7,5)
```

- Gleiche Funktion kann mit verschiedener Anzahl von Parametern aufgerufen werden

 15 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Ueberladung von Funktionen

- *Ueberladene* (*overloaded*) oder *polymorphe* Funktionen erlauben die Verwendung des gleichen Namens für verschiedene Funktionen
- *Polymorphismus* ist ein Merkmal objektorientierter Programmiersprachen
- Hier: Eine erste Bekanntschaft

- Beispiel: Prototypen

```
void print(const char * str, int width);
void print(long l, int width);
.....
print("pancakes", 15);
print(1200, 15);
```

- Bei korrekt definierten Prototypen kann Compiler aus Kontext heraus entscheiden

 16 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_6:Ueberladung I

```
// leftover.cpp -- overloading the left() function
#include <iostream>

using namespace std;

unsigned long left(unsigned long num, unsigned ct);
char * left(const char * str, int n = 1);

int main()
{
    char * trip = "Hawaii!!";   // test value
    unsigned long n = 12345678; // test value
    int i;
    char * temp;

    for (i = 1; i < 10; i++)
    {
        cout << left(n, i) << "\n";
        temp = left(trip,i);
        cout << temp << "\n";
        delete [] temp; // point to temporary storage
    }
    return 0;

}
```

Prototypen für left

Aufruf left 1

Aufruf left 2

 17 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Beispiel_6:Ueberladung II

```
unsigned long left(unsigned long num, unsigned ct)
{
    unsigned digits = 1;
    unsigned long n = num;

    if (ct == 0 || num == 0)
        return 0;        // return 0 if no digits
    while (n /= 10)
        digits++;
    if (digits > ct)
    {
    ct = digits - ct;
    while (ct--)
        num /= 10;
    return num;          // return left ct digits
    }
    else                 // if ct >= number of digits
        return num;      // return the whole number
}


char * left(const char * str, int n)
{
    if(n < 0)
        n = 0;
    char * p = new char[n+1];
    int i;
    for (i = 0; i < n && str[i]; i++)
        p[i] = str[i];  // copy characters
    while (i <= n)
        p[i++] = '\0';  // set rest of string to '\0'
    return p;
}
```

 18 ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Funktionstemplates

- *Funktionstemplates* sind *generische* Funktionsbeschreibungen, deren Typen nur in allgemeiner Form angegeben werden

- Man spricht auch von *parametrisierten* Typen
- Templates sind keine Funktionsdefinitionen
  - Beschreiben dem Compiler, wie er Funktion definieren soll
- Beispiel:

```
template <class Any>
void Swap(Any &a, Any &b)
{
   Any temp;
   temp = a;
   a = b;
   b = temp;
}
```

Beispiel_7:Templates

```
#include <iostream.h>
using namespace std;

template <class Any>  // or typename Any
void Swap(Any &a, Any &b);

int main()
{
    int i = 10;
    int j = 20;
    cout << "i, j = " << i << ", " << j << ".\n";
    cout << "Using compiler-generated int swapper:\n";
    Swap(i,j);  // generates void swap(int &, int &)
    cout << "Now i, j = " << i << ", " << j << ".\n";

    double x = 24.5;
    double y = 81.7;
    cout << "x, y = " << x << ", " << y << ".\n";
    cout << "Using compiler-generated double swapper:\n";
    Swap(x,y);  // generates void swap(double &, double &)
    cout << "Now x, y = " << x << ", " << y << ".\n";

    return 0;
}

template <class Any>  // or typename Any
void Swap(Any &a, Any &b)
{
    Any temp;   // temp a variable of type Any
    temp = a;
    a = b;
    b = temp;
}
```

Template Prototyp

Template Definition