Fachhochschule Regensburg

Test Nr. 3	Testteilnehmer
Programmieren in C++ Aufgabensteller: Prof. Sauer	Name: _____ Vorname: _____
April 2004 Arbeitszeit: 60 Minuten	Zugel. Hilfsmittel: Skriptum

1. Aufgabe

Bestimme

- theoretisch
- mit Hilfe einer Implementierung von Visual C++
- mit Hilfe einer Implementierung in GNU C++

welche der folgenden 4 überladenen Funktionen aufgerufen wird:

```
#include <iostream.h>

void f(const double&, double, void*)                    // 1
{
 cout << "Aufruf von f(const double&,double, void*)" << endl;
}
void f(double&, int, const char* = 0)                   // 2
{
 cout << "Aufruf von f(double&, int, const char* = 0)" << endl;
}
void f(int)                                             // 3
{
 cout << "Aufruf von f(int)" << endl;
}
void f(long, int = 0)                                   // 4
{
 cout << "Aufruf von f(long, int = 0)" << endl;
}

void main(void)
{
 const double pi = 3.14;
 f(1.1, 1, f);              //
 f(1, 1, "Y");
 f(3, 1.1, 0);              //
 f(1.1, 3, (void*) 0);      //
 f(1, 1);                   //
 f(pi, pi, "Y");            //
 f(0);
 f(0L);
 f(0L, 'a');
 f(pi);                     //
}
```

Das Ergebnis dieser Überlegungen bzw. Implementierungen trage in die folgende Tabelle ein:

Aufruf	Theorie	Visual C++	Gnu C++	BloodSheed
`f(1.1, 1, f);`				
`f(1, 1, "Y");`				
`f(3, 1.1, 0);`				
`f(1.1, 3, (void*) 0);`				
`f(1, 1);`				
`f(pi, pi, "Y");`				
`f(0);`				
`f(0L);`				
`f(0L, 'a');`				
`f(pi);`				

2. Aufgabe

Wende den Algorithmus zur Homonymenauflösung beim Überladen von Funktionen (Funktionsnamen) an und gib an, ob die folgenden Deklarationen zu Funktionen illegal oder legal sind!

a)

```
void f(int);  //
int f(int);   // Illegal,da gleiche Parameterliste

/* Der Rückgabetyp wird nicht in den Namen der Funktion hineincodiert */
```

b)

```
void f(int);   //
void f(int&);  // Illegal, da Unterschied nur Referenz

/* Es wird unterschieden zwischen einem Argument als value-Parameter und
einem Argument gleichen Typs als Referenz-Parameter */
```

c)

```
void f(int&);       //
void f(const int&); // legal, da es sich hier um eine andere Funktion
                    // handelt

/* Man unterscheidet: konstante und nicht konstante Parameter */
```

d)

```
void f(int);
void g(void)
{
 void f(double);   // verdeckt f(int)
 ……
}
```

e)

```
void f(int);
void f(int &); // Illegal, weil nicht unterscheidbar

/* Es ist ohne Belang, ob ein Parameter als Referenz vereinbart wurde oder
nicht. Beide sind hier völlig gleichwertig */
```

f)

```
void f(int&);
void f(const int&);  // legal, weil unterscheidbar

/* Es wird zwischen Referenzen und konstanten Referenzen unterschieden */
```

3. Aufgabe

a) Folgende Prototypen sind gegeben:

```
void f(int, char);
void f(float, int);
```

Ist der folgende Aufruf legal?

```
f(1,2);  // illegal, da keine Funktion dafür am besten passt
```

Begründe die soeben gemachte Angabe.

Am besten für den 1. Parameter passt `f(int,char)`, für den zweiten Parameter passt `f(float,int)` besser.

Keine Funktion ist am besten für beide Parameter.

Ist der folgende Aufruf in Ordnung?

```
f(1.0,2);  // Aufruf von f(float, int)
```

Begründe die soeben gemachte Angabe!

Beide Funktionen sind gleich gut für den ersten Parameter (vom Typ double, nicht float; Standardkonversion von double nach float). Der 2. Parameter passt auf `f(float,int)`, so dass eine eindeutige Entscheidung getroffen werden kann.

b) Folgende Prototypen sind gegeben

```
void f(const int*);
void f(int*)
```

Sind folgende Funktionsaufrufe legal?

```
int* p;
const int* q;
f(p);          // Aufruf von f(int*)
f(q);          // Aufruf von f(const int*)
```

Begründe die soeben gemachte Angabe!

Obwohl Zeiger eigentlich genauso gut passen wie `const`-Zeiger, wählt der Compiler doch lieber die Funktion mit dem Parameter ohne `const`, wenn der aktuelle Parameter nicht `const` ist. Bei einem aktuellen `const`-Parameter kann die Funktion ohne den formalen `const`-Parameter nicht gewählt werden.

c) Folgende Prototypen sind gegeben:

```
void f(signed char);
void f(unsigned char);
```

Ist der folgende Funktionsaufruf legal?

```
char c;
f(c);   // illegal, passt gleich gut
```

Begründe die soeben gemachte Angabe!

Der C++-Compiler unterscheidet drei char-Typen. Alle Unterscheidungen zwischen beliebigen zwei von ihnen verlangt eine Standardkonversion. Ob der Typ `char` inhaltlich gleichbedeutend mit `signed char` oder `unsigned char` ist, ist implementierungsabhängig und weder im C-oder C++-Standard festgelegt.