BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

*Fachbereich 7*
MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
*Praktische Informatik / Numerik*

e-mail: Juergen.Buhl@math.uni-wuppertal.de

# Informatik I

## WS98/99 – Übungsblatt 9

## Bearbeitungstermin: 2. Kalenderwoche

**Aufgabe 1.** *Objekte mit „lazy evaluation“ redundanter Attribute*

Testen Sie das folgende Programm:

```
//////////////////////////////////
// Datei:   Function.cc
// Version: 1.0
// Zweck:   lazy evaluation
// Autor:   Hans-Juergen Buhl
// Datum:   16.12.1998
//////////////////////////////////

#include <iostream>
#include <string>
#include <math.h>
#include <assert.h>

//using namespace std;

typedef double (*FKT)(double);

class Function{

  double start_x, end_x;
  int nr_intervals; // > 0
  FKT fkt;

  bool last_int_value_valid;
  double last_int_value;
// valid only iff (last_int_value_valid == true)

  bool last_slope_value_valid;
  double last_slope_value;
// valid only iff (last_slope_value_valid == true)
```



```
  // ...

public:

  Function(FKT f, double x_l = 0.0, double x_r = 2.0, int n = 8):
        start_x(x_l), end_x(x_r), fkt(f),
        last_int_value_valid(false),
        last_slope_value_valid(false)
        {
           assert(n>0);
           nr_intervals = n;
        };

  void set_nr_intervals(const int n)
  {
      assert(n>0);
      nr_intervals = n;
      last_int_value_valid = false;
      last_slope_value_valid = false;
  };

  int get_nr_intervals() const
  {
      return nr_intervals;
  };

  double get_integral_value()
  {
      if (!last_int_value_valid)
      {
               // ...
               // algorithm for numerical computation of integral value,
               // last_int_value = algorithm result;
               last_int_value_valid = true;
      };
      return last_int_value;
   };
};

double p(const double x)
{
      return ((15.0 * x + 4.0) * x + 3.5) * x + 2.25;
};

int main()
{
 Function sin0_2pi(sin, 0.0, 2.0*M_PI, 16);
 // test usage of Function methods
```

```
 Function poly(p);
 // further tests ...

 return 0;

}
```

Ergänzen Sie dazu die Methode `get_integral_value()` um einen „numerischen Integrationsalgorithmus“ nach der Trapezregel. Planen Sie Tests für diese Implementierung und führen Sie diese durch Modifikation der `main()`-Funktion durch.

Ergänzen Sie analog eine Methode `get_slope_value()` und testen Sie.

Warum ist

```
public:
     Function(double x_l = 0.0, double x_r = 2.0, int n = 8, FKT f) ...
```

nicht möglich?

Modifizieren Sie die Methode `Function(FKT f, double x_l = 0.0, double x_r = 2.0, int n = 8)` und neue Methoden `set_start_x(const double x_l)` sowie `set_end_x(const double x_r)` so, daß *erzwungenermaßen* die Klasseninvariante „`start_x < end_x`“ gilt. Vergessen Sie nicht, einen entsprechenden Kommentar in der Zeile „`double start_x, end_x;`“ unterzubringen!

Diskutieren Sie Vor- und Nachteile der „lazy evaluation“.

**Aufgabe 2.** *Value- und Referenzparameter*

Variieren Sie die Parameter der Funktion `power2(double x, int exp)` zu

```
power2(const double x, const int exp)
power2(double &x, int exp)
power2(double &x, int &exp)
power2(const double &x, int exp)
```

Erläutern Sie jeweils die Semantik der Parameterübergabe und sagen Sie mögliche Fehlermeldungen beim Übersetzen und Nebeneffekte voraus.

Testen Sie Ihre Vorhersagen.

**Aufgabe 3.** *Value- und Referenzparameter (Forts.)*

Mehrere skalare Funktionsergebnisse können durch Referenz-Parameter geliefert werden:

```
//////////////////////////////////
// Datei:   Point2.cc
// Version: 2.0
// Zweck:   mehrere Funktionsergebnisse
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>
```



```
//using namespace std;

class Point{

 double x,y;

public:

 Point(double a = 0.0, double b = 0.0):
    x(a), y(b) {};

 void print()
 {
    cout << "( " << x << " , " << y << " )";
 };

 double get_x() const
 {
    return x;
 };

 double get_y() const
 {
    return y;
 };

 // ...

};

void mirrorPoint(Point& p)
{
  p.print(); cout << endl;
  p = Point(-p.get_y(), p.get_x());
  p.print(); cout << endl;
};

int main()
{

 Point x(2.0, 3.0);
 x.print(); cout << endl;

 mirrorPoint(x);
 x.print(); cout << endl;

 return 0;
}
```

Testen Sie!

Welche andere Möglichkeit zur Funktionsergebnisrückgabe mehrerer Skalare sehen Sie?

Wann sollten Funktionsreferenzparameter vom Typ class außer zur Rückgabe von Ergebnissen (Nebeneffekten) eingesetzt werden?

Geben Sie Beispiele für den sinnvollen Einsatz von Parametern der Art:

```
Point p
const Point p
Point &p
const Point &p
```