Fungsi Lanjutan & Operator Overloading

Pertemuan 9

Outline

- Calling Advanced Functions
  - Overloaded Member Function
  - Using Default Values
  - The Copy Constructor
- Operator Overloading
  - Writing an Increment Method
  - Overloading the Postfix Operator
  - Overloading the Addition Operator
  - **operator=**
  - Conversion Operators
  - The **int()** Operator

Dasar Teori

- Program C++ mempunyai fasilitas operator overloading, yaitu penamaan fungsi tidak seperti pada umumnya, melainkan nama fungsi berupa kata kunci operator diikuti dengan lambang operator yang digunakan. Misalnya nama fungsi operator= yang dapat dipanggil dengan hanya menuliskan operator = seperti pada operasi penugasan biasa, sehingga dapat dibuat fungsi penugasan versi sendiri untuk tipetipe data tertentu, seperti kita dapat menyatakan Time t2 = t1, dan sebagainya. .

- Dibawah ini adalah operator-operator yang dapat dioverload :

  + - * / = += -= *= /= <> < <= >
  >= == != <= >= ++ -- % & ^ ! |
  ~ &= ^= |= && || %= [] () new delete

- Untuk mengoverload operator kita cukup menuliskan satu fungsi anggota class dengan nama operator diikuti dengan operator apa yang akan dioverload

1. Overloaded Member Function

Rectangle.cpp

```
1: #include <iostream>
2:
3: class Rectangle
4: {
5:         public:
6:                   Rectangle(int width, int height);
7:                   ~Rectangle(){}
8:
9:                   void drawShape() const;
10:                  void drawShape(int width, int height) const;
11:
12:        private:
13:                  int width;
14:                  int height;
15: };
16:
17: Rectangle::Rectangle(int newWidth, int newHeight)
18: {
19:        width = newWidth;
20:        height = newHeight;
21: }
22:
```

…

```
23: void Rectangle::drawShape() const
24: {
25:       drawShape(width, height);
26: }
27:
28: void Rectangle::drawShape(int width, int height) const
29: {
30:       for (int i = 0; i < height; i++)
31:       {
32:                 for (int j = 0; j < width; j++)
33:                 {
34:                           std::cout << "*";
35:                 }
36:                 std::cout << "\n";
37:       }
38: }
39:
40: int main()
41: {
42:       Rectangle box(30, 5);
43:       std::cout << "drawShape(): \n";
44:       box.drawShape();
45:       std::cout << "\ndrawShape(40, 2): \n";
46:       box.drawShape(40, 2);
47:       return 0;
48: }
```

2. Using Default Values

Rectangle2.cpp

```
1: #include <iostream>
2:
3: class Rectangle
4: {
5:         public:
6:                   Rectangle(int width, int height);
7:                   ~Rectangle(){}
8:                   void drawShape(int aWidth, int aHeight,
9:                   bool useCurrentValue = false) const;
10:        private:
11:                  int width;
12:                  int height;
13: };
14:
15: Rectangle::Rectangle(int aWidth, int aHeight)
16: {
17:        width = aWidth;
18:        height = aHeight;
19: }
20: void Rectangle::drawShape(
21:        int aWidth,
22:        int aHeight,
23:        bool useCurrentValue
24: ) const
```

...

```
25: {
26:         int printWidth;
27:         int printHeight;
28:
29:         if (useCurrentValue == true)
30:         {
31:                     printWidth = width;
32:                     printHeight = height;
33:         }
34:         else
35:         {
36:                     printWidth = aWidth;
37:                     printHeight = aHeight;
38:         }
39:
40:         for (int i = 0; i < printHeight; i++)
41:         {
42:                     for (int j = 0; j < printWidth; j++)
43:                     {
44:                                 std::cout << "*";
45:                     }
46:                     std::cout << "\n";
47:         }
48: }
49:
50: int main()
51: {
52:         Rectangle box(20, 5);
53:         std::cout << "drawShape(0, 0, true)...\n";
54:         box.drawShape(0, 0, true);
55:         std::cout <<"drawShape(25, 4)...\n";
56:         box.drawShape(25, 4);
57:         return 0;
58: }
```

3. The Copy Constructor

DeepCopy.cpp

```
1: #include <iostream>
2:
3: class Tricycle
4: {
5:         public:
6:         Tricycle(); // default constructor
7:         Tricycle(const Tricycle&); // copy constructor
8:         ~Tricycle(); // destructor 9: int getSpeed() const { return *speed; }
10:        void setSpeed(int newSpeed) *speed = newSpeed; }
11:        void pedal();
12:        void brake();
13:
14:        private:
15:                    int *speed;
16: };
17:
18: Tricycle::Tricycle()
19: {
20:        speed = new int;
21:        *speed = 5;
22: }
23:
24: Tricycle::Tricycle(const Tricycle& rhs)
25: {
26:        speed = new int;
27:        *speed = rhs.getSpeed();
28: }
29:
```

...

```
30: Tricycle::~Tricycle()
31: {
32:         delete speed;
33:         speed = NULL;
34: }
35:
36: void Tricycle::pedal()
37: {
38:         setSpeed(*speed + 1);
39:         std::cout << "\nPedaling " << getSpeed() << " mph\n";
40: }
41: void Tricycle::brake()
42: {
43:         setSpeed(*speed - 1);
44:         std::cout << "\nPedaling " << getSpeed() << " mph\n";
45: }
46:
47: int main()
48: {
49:         std::cout << "Creating trike named wichita ...";
50:         Tricycle wichita;
51:         wichita.pedal();
52:         std::cout << "Creating trike named dallas ...\n";
53:         Tricycle dallas(wichita);
54:         std::cout << "wichita's speed: " << wichita.getSpeed() << "\n";
55:         std::cout << "dallas's speed: " << dallas.getSpeed() << "\n";
56:         std::cout << "setting wichita to 10 ...\n";
57:         wichita.setSpeed(10);
58:         std::cout << "wichita's speed: " << wichita.getSpeed() << "\n";
59:         std::cout << "dallas's speed: " << dallas.getSpeed() << "\n";
60:         return 0;
61: }
```

4. Operator Overloading

Counter.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:        public:
6:                  Counter();
7:                  ~Counter(){}
8:                  int getValue() const { return value; }
9:                  void setValue(int x) { value = x; }
10:
11:       private:
12:                 int value;
13: };
14:
15: Counter::Counter():
16: value(0)
17: {}
18:
19: int main()
20: {
21:       Counter c;
22:       std::cout << "The value of c is " << c.getValue()
23:       << "\n";
24:       return 0;
25: }
```

5. Writing an Increment Method

Counter2.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:        public:
6:                  Counter();
7:                  ~Counter(){}
8:                  int getValue() const { return value; }
9:                  void setValue(int x) { value = x; }
10:                 void increment() { ++value; }
11:                 const Counter& operator++();
12:
13:       private:
14:                 int value;
15: };
16:
17: Counter::Counter():
18: value(0)
19: {}
20:
```

...

```
21: const Counter& Counter::operator++()
22: {
23:       ++value;
24:       return *this;
25: }
26:
27: int main()
28: {
29:       Counter c;
30:       std::cout << "The value of c is " << c.getValue()
31:       << "\n";
32:       c.increment();
33:       std::cout << "The value of c is " << c.getValue()
34:       << "\n";
35:       ++c;
36:       std::cout << "The value of c is " << c.getValue()
37:       << "\n";
38:       Counter a = ++c;
39:       std::cout << "The value of a: " << a.getValue();
40:       std::cout << " and c: " << c.getValue() << "\n";
41:       return 0;
42: }
```

6. Overloading the Postfix Operator

Counter3.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:         public:
6:                   Counter();
7:                   ~Counter(){}
8:                   int getValue() const { return value; }
9:                   void setValue(int x) { value = x; }
10:                  const Counter& operator++(); // prefix
11:                  const Counter operator++(int); // postfix
12:
13:        private:
14:                  int value;
15: };
16:
17: Counter::Counter():
18: value(0)
19: {}
20:
21: const Counter& Counter::operator++() // prefix
22: {
23:        ++value;
24:        return *this;
25: }
26:
```

…

```
27: const Counter Counter::operator++(int) // postfix
28: {
29:       Counter temp(*this);
30:       ++value;
31:       return temp;
32: }
33:
34: int main()
35: {
36:       Counter c;
37:       std::cout << "The value of c is " << c.getValue()
38:       << "\n";
39:       c++;
40:       std::cout << "The value of c is " << c.getValue()
41:       << "\n";
42:       ++c;
43:       std::cout << "The value of c is " << c.getValue()
44:       << "\n";
45:       Counter a = ++c;
46:       std::cout << "The value of a: " << a.getValue();
47:       std::cout << " and c: " << c.getValue() << "\n";
48:       a = c++;
49:       std::cout << "The value of a: " << a.getValue();
50:       std::cout << " and c: " << c.getValue() << "\n";
51:       return 0;
52: }
```

7. Overloading the Addition Operator

Counter4.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:        public:
6:                  Counter();
7:                  Counter(int initialValue);
8:                  ~Counter(){}
9:                  int getValue() const { return value; }
10:                 void setValue(int x) { value = x; }
11:                 Counter operator+(const Counter&);
12:
13:       private:
14:                 int value;
15: };
16:
17: Counter::Counter(int initialValue):
18: value(initialValue)
19: {}
20:
```

...

```
21: Counter::Counter():
22: value(0)
23: {}
24:
25: Counter Counter::operator+(const Counter &rhs)
26: {
27:       return Counter(value + rhs.getValue());
28: }
29:
30: int main()
31: {
32:       Counter alpha(4), beta(13), gamma;
33:       gamma = alpha + beta;
34:       std::cout << "alpha: " << alpha.getValue() << "\n";
35:       std::cout << "beta: " << beta.getValue() << "\n";
36:       std::cout << "gamma: " << gamma.getValue()
37:       << "\n";
38:       return 0;
39: }
```

8. operator=

Assignment.cpp

```
1: #include <iostream>
2:
3: class Tricycle
4: {
5:         public:
6:                   Tricycle();
7:                   // copy constructor and destructor use default
8:                   int getSpeed() const { return *speed; }
9:                   void setSpeed(int newSpeed) { *speed = newSpeed; }
10:                  Tricycle operator=(const Tricycle&);
11:
12:        private:
13:                  int *speed;
14: };
15:
16: Tricycle::Tricycle()
17: {
18:        speed = new int;
19:        *speed = 5;
20: }
21:
```

...

```
22: Tricycle Tricycle::operator=(const Tricycle& rhs)
23: {
24:       if (this == &rhs)
25:       return *this;
26:       delete speed;
27:       speed = new int;
28:       *speed = rhs.getSpeed();
29:       return *this;
30: }
31:
32: int main()
33: {
34:       Tricycle wichita;
35:       std::cout << "Wichita's speed: " << wichita.getSpeed()
36:       << "\n";
37:       std::cout << "Setting Wichita's speed to 6 ...\n";
38:       wichita.setSpeed(6);
39:       Tricycle dallas;
40:       std::cout << "Dallas' speed: " << dallas.getSpeed()
41:       << "\n";
42:       std::cout << "Copying Wichita to Dallas ...\n";
43:       wichita = dallas;
44:       std::cout << "Dallas' speed: " << dallas.getSpeed()
45:       << "\n";
46:       return 0;
47: }
```

9. Conversion Operators

Counter5.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:         public:
6:                   Counter();
7:                   ~Counter() {}
8:                   int getValue() const { return value; }
9:                   void setValue(int newValue) { value = newValue; }
10:        private:
11:                  int value;
12: };
13:
14: Counter::Counter():
15: value(0)
16: {}
17:
18: int main()
19: {
20:        int beta = 5;
21:        Counter alpha = beta;
22:        std::cout << "alpha: " << alpha.getValue() << "\n";
23:        return 0;
24: }
```

Counter6.cpp

```
1: #include <iostream>
2:
3: class Counter
4: {
5:       public:
6:                  Counter();
7:                  ~Counter() {}
8:                  Counter(int newValue);
9:                  int getValue() const { return value; }
10:                 void setValue(int newValue) { value = newValue; }
11:      private:
12:      int value;
13: };
14:
15: Counter::Counter():
16: value(0)
17: {}
18:
19: Counter::Counter(int newValue):
20: value(newValue)
21: {}
22:
23: int main()
24: {
25:      int beta = 5;
26:      Counter alpha = beta;
27:      std::cout << "alpha: " << alpha.getValue() << "\n";
28:      return 0;
29: }
```

10. The **int()** Operator

…

```
1: #include <iostream>
2:
3: class Counter
4: {
5:        public:
6:                  Counter();
7:                  ~Counter() {}
8:                  Counter(int newValue);
9:                  int getValue() const { return value; }
10:                 void setValue(int newValue) { value = newValue; }
11:                 operator unsigned int();
12:       private:
13:       int value;
14: };
15:
16: Counter::Counter():
17: value(0)
18: {}
19:
```

Counter7.cpp

```
20: Counter::Counter(int newValue):
21: value(newValue)
22: {}
23:
24: Counter::operator unsigned int()
25: {
26:       return (value);
27: }
28:
29: int main()
30: {
31:       Counter epsilon(19);
32:       int zeta = epsilon;
33:       std::cout << "zeta: " << zeta << "\n";
34:       return 0;
35: }
```

Tugas

- Modifikasi program Assignment untuk melakukan overload operator equal "==". Gunakan operator tersebut untuk membandingkan 2 kecepatan object dari Tricycle.
- Modifikasi program Counter2 untuk melakukan overload operator minus dan gunakan untuk melakukan pengurangan sederhana.