Lecture 10b: Type Conversions and Overloading

CS 70: Data Structures and Program Development

Week of April 06, 2020

Overloading

Definition

Overloading means

1. There are multiple pieces of code with the same name
2. If we use the (ambiguous) name, the **compiler** uses program context to decide which one you meant.

Overloaded constructors

```
class Rectangle {
 public:
    Rectangle(int, int);
    Rectangle(const Point&, int, int);
    // ...etc...
};

Rectangle r1{100,100};
Rectangle r2{Point{}, 50, 50};
```

Overloaded member functions

```
class Cow {
 public:
    void eat();
    void eat(const std::string& foodName);
    // ...
};

Cow c;
c.eat();
c.eat("broccoli")
```

Overloaded functions

```
void foo(int x)               { cout << "1"; }
void foo(double x)            { cout << "2"; }
void foo(int x, double y)     { cout << "3"; }
void foo(int* x, double* y)   { cout << "4"; }
void foo(const char* x)       { cout << "5"; }
void foo(std::string s)       { cout << "6"; }
void foo(std::vector<int> v)  { cout << "7"; }
```

```
int a = 3;
double d = 4.0;
string s = "hi!";
vector<int> v(10);
```

```
foo(3, 4.0);
foo(3);
foo(3.0);
foo(&a, &d);
foo("hi");
foo(s);
foo(v);
```

Operator Overloading

When we write the expression

```
a + c
```

C++ will check whether either of these were defined:

```
a.operator+(c);
operator+(a, c);
```

Types: Promotions and Conversions

Which of these statements will cause a compiler error?

```
int w = "bessie";

int x = 42;
x = "bessie";

double y = 3.0;

double z = 3;
```

Promotion vs. Conversion

Promotion: value-preserving change (into larger range)

- `float` to `double`
- `int` to `long`
- `unsigned int` to `unsigned long`
- `char` to `int` (or `unsigned int`, compiler dependent)

Conversion: any other implicit translation

- `int` to `float`
- `float` to `int`
- `int` to/from `unsigned int`
- `int` to `char`

What will be printed?

```
cout << 3 + 3 << endl;
cout << 3 + 3.14 << endl;
cout << 3.14 + 3 << endl;
cout << 3.14 + 3.14 << endl;
```

What will be printed?

```
int iVal = 4.99 + 3;
cout << iVal << endl;
```

What will be printed?

```
int i = 3.49;
int j = 3.50;
int k = 3.51;
int m = -3.49;
int n = -3.50;
int o = -3.51;

cout << i << " " << j << " " << k << " " << endl;
cout << m << " " << n << " " << o << " " << endl;
```

What will be printed?

```
float fVal1 = 1 / 2;
int iVal2 = 1.75 + 1 / 2;

cout << fVal1 << endl;
cout << iVal2 << endl;
```

What will be printed?

```
int negative1 = -1;
size_t positive1 = 1;

if (negative1 > positive1) {
    cout << "-1 > 1" << endl;
} else {
    cout << "-1 <= 1" << endl;
}
```

What will be printed?

```
int negative1 = -1;
size_t positive1 = 1;

if (negative1 > positive1) {
    cout << "-1 > 1" << endl;
} else {
    cout << "-1 <= 1" << endl;
}
```

What will be printed?

```
int    p = 'a';
char   q = p + 3;
int    r = 4.25 * 100;
size_t s = -1;
size_t t = s + 1;
int    u = s - 1;

cout << p << " " << q << " " << r << " " << endl;
cout << s << " " << t << " " << u << " " << endl;
```

Overloading + Conversions

C++ Overload Resolution Rules

The compiler ranks each overloading candidate as follows:

1. Exact match of types
2. Promotion of types
3. Other conversions

Chosen function must be

- better match than all of the others in at least one argument
- no worse a match than the others in any argument

Which function (if any) will the compiler choose?

a. `foo(int, int);`
b. `foo(double, double);`
c. `foo(float, double);`

1. `foo(42, 54)`
2. `foo(3.14, 2.71)`
3. `foo(3.14f, 2.71)`
4. `foo(42, 2.71)`

Which function (if any) will the compiler choose?

a. `foo(int, int);`
b. `foo(double, double);`
c. `foo(float, double);`

1. `foo(42, 54)`
2. `foo(3.14, 2.71)`
3. `foo(3.14f, 2.71)`
4. `foo(42, 2.71)`

Conversions with User-Defined Types

Conversions to objects

```
Complex c{3.0,4.0};      // 3+4i
Complex d = c + c;
Complex e = c + 9.0;
Complex f = 9.0 + c;
Complex g = 9.0 + 9.0;
```

Conversions to objects

```
Complex c{3.0,4.0};      // 3+4i
Complex d = c + c;
Complex e = c + 9.0;
Complex f = 9.0 + c;
Complex g = 9.0 + 9.0;

class Complex {
   // ...
   Complex(double a);             //  a+0i
   Complex(double a, double b);   //  a+bi
};
Complex operator+(Complex left, Complex right) { ... }
```

Implicit conversions can be surprising

```
class Cow {
 public:
   Cow(int numLegs);
   // ...
};

void feed(Cow c) { ... }

feed(4);  // OK
```

Disabling implicit conversions

```
class Cow {
public:
    explicit Cow(int numLegs);
    // ...
};

void feed(const Cow& c) { ... }

feed(4);  // ERROR
```

C++ won't convert in front of a dot. (Why?)

```
class Complex {
public:
   Complex(double a);           //  a+0i

   Complex conjugate() const;   //  returns a-bi
};

Complex d = Complex(3.0).conjugate();  // OK
Complex c = (3.0).conjugate();         // ERROR
```

Learning Targets

1. I can define what "overloading" means and give examples.
2. I can explain the difference between "promotion" and "conversion" in C++ and give examples.