Object Oriented Programming with C++

10. Type Conversion

By: Prof. Pandav Patel

Second Semester, 2020-21
Computer Engineering Department
Dharmsinh Desai University

```
#include<iostream>
#include<string>

using std::cout;
using std::endl;
using std::ostream;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }

    friend ostream &operator<<(ostream &strm, Number &n);
};

ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    n = 20;
    cout << n << endl;
    return 0;
}
```

```
constructor called
num is: 10
constructor called
num is: 20
```

```
#include<iostream>
#include<string>

using std::cout;
using std::endl;
using std::ostream;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }

    friend ostream &operator<<(ostream &strm, Number &n);
};

ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    int i;
    // error: cannot convert ‘Number’ to ‘int’ in assignment
    i = n;
    cout << i << endl;
    return 0;
}
```

Conversation Function

- Enables conversion from a class type to another type.
- Conversion function is declared like member function with no parameters,
  - no explicit return type
- **operator** conversion-type-id() {}
- When such member function is declared in class X, it performs conversion from X to conversion-type-id

```
#include<iostream>
#include<string>

using std::cout;
using std::endl;
using std::ostream;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }

    operator int() {
        cout << "conversion function called\n";
        return num;
    }

    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    int i;
    i = n;
    cout << i << endl;
    return 0;
}
```

```
constructor called
num is: 10
conversion function called
10
```

```
class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }
    operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
        constructor called
        num is: 10
        Fnum constructor called
        Fnum is: 7.7
        conversion function 2 called
        constructor called
        num is: 7
        Fnum constructor 2 called
        Fnum is: 7
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    }

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }

    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << “Fnum is: " << fn.fnum;
    return strm;
}
```

Working Perfect.... Let’s try to answer following Questions:

- **Why fn = n working without conversion function?**
- **Can we create Conversion function for it ?**

```
class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }
    operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
constructor called
num is: 10
Fnum constructor called
Fnum is: 7.7
conversion function 2 called
constructor called
num is: 7
Fnum constructor 2 called
Destructor Called
Fnum is: 7
Destructor Called
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    }

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }
    ~Fnumber(){
        cout<<"Destructor Called"<<endl;
    }
    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << "num is: " << fn.fnum;
    return strm;
}
```

For fn = n statement compiler perform followings steps:
- Compiler creates temporary object by calling Fnum constructor 2
- Copies temporary to fn and destroys temporary

```
class A{
public:
    int a1;
    A(){
        a1=10;
    }
};

class B{
    float b1;
public:
    B(){
        b1=5;
        cout<<"B"<<endl;
    }
  B(A a){
      cout<<"Constructor2 Called"<<endl;
    b1=(int)a.a1;
  }
  float get_b1(){
      return b1;
  }
  ~B(){
      cout<<"Destructor Called"<<endl;
  }
};
```

```
int main(){
    B o1;
    A o2;
    cout<<o1.get_b1()<<endl;
    o1=o2;
    cout<<o1.get_b1()<<endl;
}
```

```
B
5
Constructor2 Called
Destructor Called
10
Destructor Called
```

Just another Example

Here also compiler creates temporary object of type B from o2 by calling constructor2 (conversion constructor) while we write o1=o2

Then it copies temporary to o1 and destroys the temporary object

```
class Fnumber;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    }

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }

    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << "num is: " << fn.fnum;
    return strm;
}
```

```
operator Fnumber(){
        cout << “Conversion function from Number is called\n";
       return Fnumber(float(num));
}
    operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

error: return type ‘class Fnumber’ is incomplete c++

```
class Fnumber;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }


Operator Fnumber();
operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    }

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }

    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << "num is: " << fn.fnum;
    return strm;
}
Number:: operator Fnumber(){
        cout << “Conversion function from Number is called\n";
       return Fnumber(float(num));
}
```

Output ??

```
class Fnumber;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }


Operator Fnumber();
operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    }

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }

    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << "num is: " << fn.fnum;
    return strm;
}
```

```
Number:: operator Fnumber(){
        cout << “Conversion function from Number is called\n";
       return Fnumber(float(num));
}
```

error: conversion from ‘Number’ to ‘Fnumber’ is ambiguous

```
class Fnumber;

class Number {
    int num;
public:
    Number(int num) {
        cout << "constructor called\n";
        this->num = num;
    }
Operator Fnumber();
operator int() {
        cout << "conversion function called\n";
        return num;
    }
    int get_num() {
        return num;
    }
    friend ostream &operator<<(ostream &strm, Number &n);
};
ostream &operator<<(ostream &strm, Number &n) {
    strm << "num is: " << n.num;
    return strm;
}
```

```
int main() {
    Number n = 10;
    cout << n << endl;;
    Fnumber fn = 7.7f;
    cout << fn << endl;
    n = fn;
    cout << n << endl;
    fn = n;
    cout << fn << endl;
    return 0;
}
```

```
constructor called
num is: 10
Fnum constructor called
num is: 7.7
conversion function 2 called
constructor called
num is: 7
conversion function from Number is called
Fnum constructor called
num is: 7
```

```
class Fnumber {
    float fnum;
public:
    Fnumber(float fnum) {
        cout << "Fnum constructor called\n";
        this->fnum = fnum;
    }

    /*
    Fnumber(Number n) {
        cout << "Fnum constructor 2 called\n";
        this->fnum = n.get_num();
    } */

    operator Number() {
        cout << "conversion function 2 called\n";
        return Number(int(fnum));
    }

    friend ostream &operator<<(ostream &strm, Fnumber &fn);
};
ostream &operator<<(ostream &strm, Fnumber &fn) {
    strm << "num is: " << fn.fnum;
    return strm;
}
Number:: operator Fnumber(){
        cout << “Conversion function from Number is called\n";
       return Fnumber(float(num));
}
```

Interesting reads

- Conversion constructor vs. conversion operator: precedence
  - https://stackoverflow.com/questions/1384007/conversion-constructor-vs-conversion-operator-precedence