



Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary

Module 33: Programming in C++

Type Casting & Cast Operators: Part 2

Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

sourangshu@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in C++

by **Prof. Partha Pratim Das**



Module Objectives

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary

- Understand casting in C and C++



Module Outline

Module 33

Sourangshu
Bhattacharya

Objectives & Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary

- Casting: C-Style: RECAP
 - Upcast & Downcast
- Cast Operators in C++
 - `const_cast` Operator
 - `static_cast` Operator
 - `reinterpret_cast` Operator
 - `dynamic_cast` Operator
- `typeid` Operator



Casting in C and C++

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary

- Casting in C
 - Implicit cast
 - Explicit C-Style cast
 - **Loses type information in several contexts**
 - **Lacks clarity of semantics**
- Casting in C++
 - Performs fresh inference of types **without change of value**
 - Performs fresh inference of types **with change of value**
 - Using **implicit computation**
 - Using **explicit (user-defined) computation**
 - **Preserves type information** in all contexts
 - Provides **clear semantics** through **cast operators**:
 - `const_cast`
 - `static_cast`
 - `reinterpret_cast`
 - `dynamic_cast`
 - Cast operators can be **grep**-ed in source
 - **C-Style cast must be avoided in C++**



static_cast Operator

- `static_cast` performs all conversions allowed implicitly (not only those with pointers to classes), and also the opposite of these. It can:
 - Convert from `void*` to any pointer type
 - Convert integers, floating-point values and `enum` types to `enum` types
- `static_cast` can perform conversions between pointers to related classes:
 - Not only up-casts, but also down-casts
 - No checks are performed during run-time to guarantee that the object being converted is in fact a full object of the destination type
- Additionally, `static_cast` can also perform the following:
 - Explicitly call a single-argument constructor or a conversion operator
 - The User-Defined Cast
 - Convert to rvalue references
 - Convert `enum` class values into integers or floating-point values
 - Convert any type to `void`, evaluating and discarding the value

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary



static_cast Operator: Built-in Types

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators
static_cast
reinterpret_cast

Summary

```
#include <iostream>
using namespace std;
```

```
// Built-in Types
```

```
int main() {
    int i = 2;
    double d = 3.7;
    double *pd = &d;
```

```
    i = d;                // implicit -- warning
    i = static_cast<int>(d); // static_cast -- okay
    i = (int)d;            // C-style -- okay
```

```
    d = i;                // implicit -- okay
    d = static_cast<double>(i); // static_cast -- okay
    d = (double)i;         // C-style -- okay
```

```
    i = pd;                // implicit -- error
    i = static_cast<int>(pd); // static_cast -- error
    i = (int)pd;            // C-style -- okay:    RISKY: Should use reinterpret_cast
```

```
    return 0;
}
```



static_cast Operator: Class Hierarchy

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

```
#include <iostream>
using namespace std;

// Class Hierarchy
class A { };
class B: public A { };

int main() {
    A a;
    B b;

    // UPCAST
    A *p = &b;           // implicit -- okay
    p = static_cast<A*>(&b); // static_cast -- okay
    p = (A*)&b;           // C-style -- okay

    // DOWNCAST
    q = &a;               // implicit -- error
    q = static_cast<B*>(&a); // static_cast -- okay: RISKY: Should use dynamic_cast
    q = (B*)&a;           // C-style -- okay

    return 0;
}
```



static_cast Operator: Pitfall

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

```
class Window { public: virtual void onResize(); ... }
class SpecialWindow: public Window { // derived class
public:
    virtual void onResize() { // derived onResize impl;
        static_cast<Window>(*this).onResize(); // cast *this to Window,
        // then call its onResize;
        // this doesn't work!

        ... // do SpecialWindow-specific stuff
    }
    ...
};
```

Slices the object, creates a temporary and calls the method!

```
class SpecialWindow: public Window { // derived class
public:
    virtual void onResize() { // derived onResize impl;
        Window::onResize(); // Direct call works

        ... // do SpecialWindow-specific stuff
    }
    ...
};
```




static_cast Operator: Unrelated Classes

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

```
#include <iostream>
using namespace std;
```

```
// Un-related Types
```

```
class B;
class A {
public:
```

```
};
```

```
class B { };
```

```
int main() {
```

```
    A a;
```

```
    B b;
```

```
    int i = 5;
```

```
    // B ==> A
```

```
    a = b;
```

```
    // error
```

```
    a = static_cast<A>(b);
```

```
    // error
```

```
    a = (A)b;
```

```
    // error
```

```
    // int ==> A
```

```
    a = i;
```

```
    // error
```

```
    a = static_cast<A>(i);
```

```
    // error
```

```
    a = (A)i;
```

```
    // error
```

```
    return 0;
```

```
}
```

```
#include <iostream>
using namespace std;
```

```
// Un-related Types
```

```
class B;
class A {
public:
```

```
    A(int i = 0) { cout << "A::A(i)\n"; }
```

```
    A(const B&) { cout << "A::A(B&)\n"; }
```

```
};
```

```
class B { };
```

```
int main() {
```

```
    A a;
```

```
    B b;
```

```
    int i = 5;
```

```
    // B ==> A
```

```
    a = b;
```

```
    // Uses A::A(B&)
```

```
    a = static_cast<A>(b);
```

```
    // Uses A::A(B&)
```

```
    a = (A)b;
```

```
    // Uses A::A(B&)
```

```
    // int ==> A
```

```
    a = i;
```

```
    // Uses A::A(int)
```

```
    a = static_cast<A>(i);
```

```
    // Uses A::A(int)
```

```
    a = (A)i;
```

```
    // Uses A::A(int)
```

```
    return 0;
```

```
}
```



static_cast Operator: Unrelated Classes

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

```
#include <iostream>
using namespace std;

// Un-related Types
class B;
class A { int i_; public:

};
class B { public:

};

int main() {
    A a; B b; int i = 5;

    // B ==> A
    a = b;                // error
    a = static_cast<A>(b); // error
    a = (A)b;             // error

    // A ==> int
    i = a;                // error
    i = static_cast<int>(a); // error
    i = (int)a;           // error

    return 0;
}
```

```
#include <iostream>
using namespace std;

// Un-related Types
class B;
class A { int i_; public:
    A(int i = 0) : i_(i)
    { cout << "A::A(i)\n"; }
    operator int()
    { cout << "A::operator int()\n"; return i_; }
};
class B { public:
    operator A()
    { cout << "B::operator A()\n"; return A(); }
};

int main() {
    A a; B b; int i = 5;

    // B ==> A
    a = b;                // B::operator A()
    a = static_cast<A>(b); // B::operator A()
    a = (A)b;             // B::operator A()

    // A ==> int
    i = a;                // A::operator int()
    i = static_cast<int>(a); // A::operator int()
    i = (int)a;           // A::operator int()

    return 0;
}
```



reinterpret_cast Operator

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

- `reinterpret_cast` converts any pointer type to any other pointer type, even of unrelated classes
- The operation result is a simple binary copy of the value from one pointer to the other
- All pointer conversions are allowed: neither the content pointed nor the pointer type itself is checked
- It can also cast pointers to or from integer types
- The format in which this integer value represents a pointer is platform-specific
- The only guarantee is that a pointer cast to an integer type large enough to fully contain it (such as `intptr_t`), is guaranteed to be able to be cast back to a valid pointer
- The conversions that can be performed by `reinterpret_cast` but not by `static_cast` are low-level operations based on reinterpreting the binary representations of the types, which on most cases results in code which is system-specific, and thus non-portable



reinterpret_cast Operator

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

static_cast
reinterpret_cast

Summary

```
#include <iostream>
using namespace std;

class A {};
class B {};

int main() {
    int i = 2;
    double d = 3.7;
    double *pd = &d;

    i = pd; // implicit -- error
    i = reinterpret_cast<int>(pd); // reinterpret_cast -- okay
    i = (int)pd; // C-style -- okay
    cout << pd << " " << i << endl;

    A *pA;
    B *pB;

    pA = pB; // implicit -- error
    pA = reinterpret_cast<A*>(pB); // reinterpret_cast -- okay
    pA = (A*)pB; // C-style -- okay

    return 0;
}
```



Module Summary

Module 33

Sourangshu
Bhattacharya

Objectives &
Outline

Cast
Operators

`static_cast`
`reinterpret_cast`

Summary

- Studied `static_cast`, and `reinterpret_cast` with examples