



Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators

`const_cast`

Summary

Module 32: Programming in C++

Type Casting & Cast Operators: Part 1

Partha Pratim Das

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

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan



Module Objectives

Module 32

Partha Pratim
Das

Objectives & Outline

Casting

Upcast &
Downcast

Cast

Operators

`const_cast`

Summary

- Understand casting in C and C++



Module Outline

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators
`const_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



Type Casting

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast

Operators

const_cast

Summary

- Why casting?
 - Casts are used to convert the type of an object, expression, function argument, or return value to that of another type
- (Silent) Implicit conversions
 - The standard C++ conversions and user-defined conversions
- Explicit conversions
 - Type is needed for an expression that cannot be obtained through an implicit conversion more than one standard conversion creates an ambiguous situation
- To perform a type cast, the compiler
 - Allocates temporary storage
 - Initializes temporary with value being cast

```
double f (int i,int j) { return (double) i / j; }
```

```
// compiler generates:
```

```
double f (int i, int j) {  
    double temp_i = i, temp_j = j; // Conversion in temporary  
    return temp_i / temp_j;  
}
```



Casting: C-Style: RECAP (Module 26)

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators

`const_cast`

Summary

- Casting is performed when a value (variable) of one type is used in place of some other type

```
int i = 3;  
double d = 2.5;
```

```
double result = d / i; // i is cast to double and used
```

- Casting can be implicit or explicit

```
int i = 3;  
double d = 2.5;
```

```
double *p = &d;
```

```
d = i;      // implicit
```

```
i = d;      // implicit -- // warning C4244: '=' : conversion from 'double' to 'int',  
           // possible loss of data
```

```
i = (int)d; // explicit
```

```
i = p;      // error C2440: '=' : cannot convert from 'double *' to 'int'  
i = (int)p; // explicit
```



Casting: C-Style: RECAP (Module 26)

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators
const_cast

Summary

- (Implicit) Casting between unrelated classes is not permitted

```
class A { int i; };  
class B { double d; };
```

```
A a;  
B b;
```

```
A *p = &a;  
B *q = &b;
```

```
a = b;      // error C2679: binary '=' : no operator found  
            // which takes a right-hand operand of type 'main::B'
```

```
a = (A)b;   // error C2440: 'type cast' : cannot convert from 'main::B' to 'main::A'
```

```
b = a;      // error C2679: binary '=' : no operator found  
            // which takes a right-hand operand of type 'main::A'
```

```
b = (B)a;   // error C2440: 'type cast' : cannot convert from 'main::A' to 'main::B'
```

```
p = q;      // error C2440: '=' : cannot convert from 'main::B *' to 'main::A *'
```

```
q = p;      // error C2440: '=' : cannot convert from 'main::A *' to 'main::B *'
```

```
p = (A*)&b;   // Forced -- Okay  
q = (B*)&a;   // Forced -- Okay
```



Casting: C-Style: RECAP (Module 26)

- Forced Casting between unrelated classes is dangerous

```
class A { public: int i; };
class B { public: double d; };

A a;
B b;

a.i = 5;
b.d = 7.2;

A *p = &a;
B *q = &b;

cout << p->i << endl; // prints 5
cout << q->d << endl; // prints 7.2

p = (A*)&b;
q = (B*)&a;

cout << p->i << endl; // prints -858993459 ----- GARBAGE
cout << q->d << endl; // prints -9.25596e+061 ----- GARBAGE
```

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast

Operators
const_cast

Summary



Casting on a Hierarchy: C-Style: RECAP (Module 26)

- Casting on a hierarchy is permitted in a limited sense

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

```
A *pa = 0;  
B *pb = 0;  
void *pv = 0;
```

```
pa = pb; // okay ----- // UPCAST
```

```
pb = pa; // error C2440: '=' : cannot convert from 'A *' to 'B *' // DOWNCAST
```

```
pv = pa; // okay ----- // Lose the type
```

```
pv = pb; // okay ----- // Lose the type
```

```
pa = pv; // error C2440: '=' : cannot convert from 'void *' to 'A *'
```

```
pb = pv; // error C2440: '=' : cannot convert from 'void *' to 'B *'
```

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast

Operators

const_cast

Summary



Casting on a Hierarchy: C-Style: RECAP (Module 26)

● Up-Casting is safe

```
class A { public: int dataA_; };  
class B : public A { public: int dataB_; };
```

```
A a;  
B b;
```

```
a.dataA_ = 2;  
b.dataA_ = 3;  
b.dataB_ = 5;
```

```
A *pa = &a;  
B *pb = &b;
```

```
cout << pa->dataA_ << endl; // prints 2  
cout << pb->dataA_ << " " << pb->dataB_ << endl; // prints 3 5
```

```
pa = &b;
```

```
cout << pa->dataA_ << endl; // prints 3  
// cout << pa->dataB_ << endl; // error C2039: 'dataB_' : is not a member of 'A'
```

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast

Operators

const_cast

Summary



Casting in C and C++

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators

`const_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++



Cast Operators

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting
Upcast &
Downcast

Cast
Operators
`const_cast`

Summary

- A **cast operator** take an expression of **source type** (implicit from the expression) and convert it to an expression of **target type** (explicit in the operator) following the **semantics of the operator**
- Use of cast operators increases robustness by generating errors in **static** or **dynamic** time



Cast Operators

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators

`const_cast`

Summary

- `const_cast` operator: `const_cast<type>(expr)`
 - Explicitly **overrides const and/or volatile** in a cast
 - Usually **does not perform computation or change value**
- `static_cast` operator: `static_cast<type>(expr)`
 - Performs a **non-polymorphic cast**
 - Usually **performs computation to change value** – **implicit** or **user-defined**
- `reinterpret_cast` operator: `reinterpret_cast<type>(expr)`
 - Casts between **unrelated pointer types** or **pointer and integer**
 - **Does not perform computation yet reinterprets value**
- `dynamic_cast` operator: `dynamic_cast<type>(expr)`
 - Performs a **run-time cast** that verifies the validity of the cast
 - **Performs pre-defined computation**, sets **null** or **throws exception**



const_cast Operator

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting
Upcast &
Downcast

Cast
Operators
`const_cast`

Summary

- `const_cast` converts between types with different cv-qualification
- Only `const_cast` may be used to cast away (remove) const-ness or volatility
- Usually **does not perform computation or change value**



const_cast Operator

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators
const_cast

Summary

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

class A { int i_;
public: A(int i) : i_(i) {}
       int get() const { return i_; }
       void set(int j) { i_ = j; }
};

void print(char * str) { cout << str; }

int main() {
    const char * c = "sample text";
    // print(c); // error: 'void print(char *)': cannot convert argument 1
                // from 'const char *' to 'char *'

    print(const_cast<char *>(c));

    const A a(1);
    a.get();

    // a.set(5); // error: 'void A::set(int)': cannot convert
                // 'this' pointer from 'const A' to 'A &'

    const_cast<A&>(a).set(5);

    // const_cast<A>(a).set(5); // error: 'const_cast': cannot convert
                                // from 'const A' to 'A'

    return 0;
}
```



const_cast Operator vis-a-vis C-Style Cast

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators
const_cast

Summary

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

class A { int i_;
public: A(int i) : i_(i) {}
      int get() const { return i_; }
      void set(int j) { i_ = j; }
};

void print(char * str) { cout << str; }

int main() {
    const char * c = "sample text";

    // print(const_cast<char *>(c));
    print((char *) (c));           // C-Style Cast

    const A a(1);

    // const_cast<A&>(a).set(5);
    ((A&)a).set(5);                 // C-Style Cast

    // const_cast<A>(a).set(5);      // error: 'const_cast': cannot convert
                                    // from 'const A' to 'A'
    ((A)a).set(5);                  // C-Style Cast

    return 0;
}
```



const_cast Operator

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting

Upcast &
Downcast

Cast
Operators
const_cast

Summary

```

#include <iostream>
using namespace std;
struct type { type() :i(3) {}
    void m1(int v) const {
        //this->i = v; // error C3490: 'i' cannot be modified because
        // it is being accessed through a const object
        const_cast<type*>(this)->i = v; // OK as long as the type object isn't const
    }
    int i;
};

int main() {
    int i = 3;                // i is not declared const
    const int& cref_i = i;
    const_cast<int&>(cref_i) = 4; // OK: modifies i
    cout << "i = " << i << '\n';

    type t; // note, if this is const type t;, then t.m1(4); is undefined behavior
    t.m1(4);
    cout << "type::i = " << t.i << '\n';

    const int j = 3; // j is declared const
    int* pj = const_cast<int*>(&j);
    *pj = 4;        // undefined behavior! Value of j and *pj may differ
    cout << j << " " << *pj << endl;

    void (type::*mfp)(int) const = &type::m1; // pointer to member function
    //const_cast<void(type::*)(int)>(mfp); // error C2440: 'const_cast' : cannot convert
    // from 'void (__thiscall type::*)(int) const' to 'void (__thiscall type::*)(int)'
    // const_cast does not work on function pointers
    return 0;
}

```

Output:

```

i = 4
type::i = 4
3 4

```




Module Summary

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting
Upcast &
Downcast

Cast
Operators
`const_cast`

Summary

- Understood casting in C and C++
- Explained cast operators in C++ and discussed the evils of C-style casting
- Studied `const_cast` with examples



Instructor and TAs

Module 32

Partha Pratim
Das

Objectives &
Outline

Casting
Upcast &
Downcast

Cast
Operators
`const_cast`

Summary

Name	Mail	Mobile
Partha Pratim Das, <i>Instructor</i>	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, <i>TA</i>	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, <i>TA</i>	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, <i>TA</i>	himadribhuyan@gmail.com	9438911655