

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

CS100 Recitation 8

GKxx

April 11, 2022

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

1 Copy Control

- Copying an Object
 - The Copy Constructor
 - The Copy-Assignment Operator
 - Synthesized Copy Operations
 - Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- auto
- decltype

When Copy Happens

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Point2d {  
    double x, y;  
public:  
    Point2d() : Point2d(0, 0) {}  
    Point2d(double a, double b) : x(a), y(b) {}  
};
```

Question

What member functions have been synthesized by the compiler?

When Copy Happens

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Point2d {  
    double x, y;  
public:  
    Point2d() : Point2d(0, 0) {}  
    Point2d(double a, double b) : x(a), y(b) {}  
};
```

When we have the class definition above, there are 3 ways of constructing an object:

```
Point2d p0;  
Point2d p1(3.14, 6.28);  
Point2d p2(p1); // same as Point2d p2 = p1;
```

When Copy Happens

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Point2d {  
    double x, y;  
public:  
    Point2d() : Point2d(0, 0) {}  
    Point2d(double a, double b) : x(a), y(b) {}  
};
```

We can also pass Point2d objects as arguments, or as return-values:

```
// BAD!! You should use reference-to-const.  
Point2d less_in_x(Point2d lhs, Point2d rhs) {  
    return lhs.get_x() < rhs.get_x() ? lhs : rhs;  
}
```

When Copy Happens

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

How many copies are created?

```
Point2d p3 = less_in_x(p0, p1);
```

When Copy Happens

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

How many copies are created?

```
Point2d p3 = less_in_x(p0, p1);
```

- Copy-initialize the parameters lhs and rhs.
- Copy-initialize a temporary object generated by the calling expression with the return value. (?)
- Copy-initialize p3. (not assignment!)

When Copy Happens

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction
auto
decltype

How many copies are created?

```
Point2d p3 = less_in_x(p0, p1);
```

- Copy-initialize the parameters lhs and rhs.
- Copy-initialize a temporary object generated by the calling expression with the return value. (?)
- Copy-initialize p3. (not assignment!)

Copying a return-value?

- Many compilers avoid such copying by **Return Value Optimization (RVO)**.
- Since C++11, a local object will be returned by **moving** instead of **copying**, if it is **move-constructible**.
- C++17 **guarantees** that such copying won't happen, even when the object is not move-constructible.

Contents

CS100
Recitation 8
GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- auto
- decltype

The Copy Constructor

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

The **copy ctor** defines the behavior of copy initializing an object.

```
class Point2d {  
public:  
    Point2d(const Point2d &other)  
        : x(other.x), y(other.y) {}  
    // other members  
};
```

The Copy Constructor

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The

Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

The **copy ctor** defines the behavior of copy initializing an object.

```
class Point2d {  
public:  
    Point2d(const Point2d &other)  
        : x(other.x), y(other.y) {}  
    // other members  
};
```

- Can we define the parameter type as `Point2d` instead of reference-to-**const**?
- Can we define the parameter type as `Point2d &`?

The Copy Constructor

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Vector {
    std::size_t m_size, m_capacity;
    int *m_data;
public:
    Vector(const Vector &other)
        : m_size(other.m_size),
          m_capacity(other.m_capacity),
          m_data(new int[m_capacity]{})) {
        for (std::size_t i = 0; i < m_size; ++i)
            m_data[i] = other.m_data[i];
    }
    // other members
};
```

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

- 1 Copy Control
 - Copying an Object
 - The Copy Constructor
 - The Copy-Assignment Operator
 - Synthesized Copy Operations
 - Prevent Copying
- 2 More about Class
 - Constructors and Type-casting
 - Friends
 - Constant Member Functions Revisited
- 3 Type Deduction
 - auto
 - decltype

The Copy-Assignment Operator

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

The behavior of assignment like

```
Point2d p1, p2;
```

```
p1 = p2;
```

is defined by the [copy-assignment operator](#). It is defined by [overloading](#) the assignment operator.

```
class Point2d {  
    public:  
        Point2d &operator=(const Point2d &other) {  
            x = other.x;  
            y = other.y;  
            return *this;  
        }  
};
```

The Copy-Assignment Operator

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction
auto
decltype

```
class Point2d {  
public:  
    Point2d &operator=(const Point2d &other) {  
        x = other.x;  
        y = other.y;  
        // return reference to the object itself  
        return *this;  
    }  
};
```

Notice

You should make the overloaded operator behave in consistence with the built-in one.

The Copy-Assignment Operator

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

When an assignment happens, the left-hand operand is bound to the implicit `this`. The right-hand operand is passed as the parameter.

```
Point2d p1, p2;  
p1 = p2;  
p1.operator=(p2); // equivalent way
```


The Copy-Assignment Operator

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

When an assignment happens, the left-hand operand is bound to the implicit `this`. The right-hand operand is passed as the parameter.

```
Point2d p1, p2;  
p1 = p2;  
p1.operator=(p2); // equivalent way
```

Since the assignment operator returns the object on the left-hand side, we can chain assignments together:

```
p1 = p2 = p3;
```

Copy-Assignment of Vector

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

```
class Vector {  
public:  
    Vector &operator=(const Vector &other) {  
        delete[] m_data;  
        m_size = other.m_size;  
        m_capacity = other.m_capacity;  
        m_data = new int[m_capacity];  
        for (std::size_t i = 0; i < m_size; ++i)  
            m_data[i] = other.m_data[i];  
        return *this;  
    }  
    // other members  
};
```

Copy-Assignment of Vector

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

```
class Vector {  
public:  
    Vector &operator=(const Vector &other) {  
        delete[] m_data;  
        m_size = other.m_size;  
        m_capacity = other.m_capacity;  
        m_data = new int[m_capacity];  
        for (std::size_t i = 0; i < m_size; ++i)  
            m_data[i] = other.m_data[i];  
        return *this;  
    }  
    // other members  
};
```

- Anything wrong with this assignment operator?

Self-assignment Safety

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

- Should self-assignment happen, the data is **deleted** at first! It becomes a disaster.
- Exception-safety issue.

Self-assignment Safety

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

- Should self-assignment happen, the data is **deleted** at first! It becomes a disaster.
- Exception-safety issue.

Self-assignment may happen unnoticed and without a warning:

```
Vector v = some_value();  
Vector &rv = some_function(v);  
v = rv;
```

Notice

Assignment operator should always be **self-assignment-safe**.

The Correct Way

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Vector {  
public:  
    Vector &operator=(const Vector &other) {  
        int *new_data = new int[other.m_capacity];  
        for (std::size_t i = 0; i < other.m_size; ++i)  
            new_data[i] = other.m_data[i];  
        m_size = other.m_size;  
        m_capacity = other.m_capacity;  
        delete[] m_data;  
        m_data = new_data;  
        return *this;  
    }  
};
```

Still Problematic Way

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

```
Vector &Vector::operator=(const Vector &other) {  
    // test self-assignment directly  
    if (this != &other) {  
        delete[] m_data;  
        m_size = other.m_size;  
        m_capacity = other.m_capacity;  
        m_data = new int[m_capacity];  
        for (std::size_t i = 0; i < m_size; ++i)  
            m_data[i] = other.m_data[i];  
    }  
    return *this;  
}
```

- This handles self-assignment correctly, but **still has exception-safety issue**.

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- **Synthesized Copy Operations**
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- auto
- decltype

Synthesized Copy Ctor

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

The compiler will synthesize a copy ctor if

- the copy ctor is not defined, and
- every member is **copy-constructible**.

The synthesized copy ctor will copy-initialize the members one-by-one, and has an empty function body.

Question

Is it ok for Point2d to use the synthesized copy ctor? What about Vector?

Synthesized Copy-Assignment Operator

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

The compiler will synthesize a copy-assignment operator if

- the copy-assignment operator is not defined, and
- every member is **copy-assignable**.

The synthesized copy-assignment operator will copy-assign the members one-by-one, and of course return ***this**.

Synthesized Copy-Assignment Operator

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

The compiler will synthesize a copy-assignment operator if

- the copy-assignment operator is not defined, and
- every member is **copy-assignable**.

The synthesized copy-assignment operator will copy-assign the members one-by-one, and of course return ***this**.

Question

Is it ok for `Point2d` to use the synthesized copy-assignment operator? What about `Vector`?

Copying Array Members

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

It is not allowed to copy arrays directly like

```
int a[100], b[100];  
a = b;
```

But if there's an array member, the synthesized copy operations will copy the elements in the array one-by-one. Don't worry!

Use =default

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

For a **default ctor**, a **copy ctor**, a **copy-assignment operator** or a **destructor**, we can explicitly require the compiler to synthesize one with defaulted behavior by **=default**:

```
class Point2d {  
    double x, y;  
public:  
    Point2d() = default;  
    Point2d(const Point2d &) = default;  
    Point2d &operator=(const Point2d &) = default;  
    ~Point2d() = default;  
    // other members  
};
```

The Rule of Three

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

Theorem (The Rule of Three)

If a class needs one of the three copy-controlling operations (copy-ctor, copy-assignment operator and destructor), it is highly possible that all of them are needed.

- Such idea was not so widely acknowledged when C++98 came out. Therefore, the compiler will still generate the others if you only define one or two of them.
- We will see changes in C++11 when we talk about **moving**.

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- auto
- decltype

Prevent Copying

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

Some class, like `std::istream`, should avoid copying. (Why?)

Question

Can we prevent copying by simply not defining the copy operations?

Prevent Copying

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

Some class, like `std::istream`, should avoid copying. (Why?)

Question

Can we prevent copying by simply not defining the copy operations?

Before C++11, people prevent copying by **declaring the copying operations as private, and not defining them**.

- Attempts to copy such an object outside the class and out of a friend will cause an error in access-level.
- Attempts to copy inside the class or in a friend will cause a linking error.

Deleted Functions

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

Since C++11, we can define a function as **deleted** by defining it to be `=delete`.

```
class Uncopyable {  
    public:  
        Uncopyable(const Uncopyable &) = delete;  
        Uncopyable &operator=(const Uncopyable &) = delete;  
};
```

Deleted Functions

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

Since C++11, we can define a function as **deleted** by defining it to be `=delete`.

```
class Uncopyable {  
    public:  
        Uncopyable(const Uncopyable &) = delete;  
        Uncopyable &operator=(const Uncopyable &) = delete;  
};
```

Use `=delete` to avoid copying in modern C++!

Deleted Functions

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

Since C++11, we can define a function as **deleted** by defining it to be `=delete`.

```
class Uncopyable {  
    public:  
        Uncopyable(const Uncopyable &) = delete;  
        Uncopyable &operator=(const Uncopyable &) = delete;  
};
```

Use `=delete` to avoid copying in modern C++!

Notice

If we define a special member function to be `=default` but the compiler cannot synthesize it, it is implicitly **deleted** and will not cause an error (but will generate a warning).

Recommended Reading Materials

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

- *Effective C++*, Item 5: Know what functions C++ silently writes and calls.
- *Effective C++*, Item 6: Explicitly disallow the use of compiler-generated functions you do not want.

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- auto
- decltype

Constructors and Type-casting

CS100
Recitation 8
GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

A constructor also defines a type-casting:

```
void fun(std::string s) {  
    // do something  
}  
  
int main() {  
    fun("Hello world");  
    return 0;  
}
```

- `std::string` has a constructor that accepts a `const char *` parameter.
- When calling `fun("Hello")`, the initialization of the parameter could be seen as a **conversion** from `const char *` to `std::string`.

Constructors and Type-casting

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

Sometimes this will be confusing.

```
class Vector {  
public:  
    Vector(std::size_t n)  
        : m_size(n), m_capacity(n),  
          m_data(new int[n]()) {}  
};  
int main() {  
    Vector v = 10; // What??  
    return 0;  
}
```


explicit Constructors

By defining a constructor as `explicit`, we disallow such conversion from happening implicitly.

```
class Vector {  
    public:  
        explicit Vector(std::size_t n)  
            : m_size(n), m_capacity(n),  
              m_data(new int[n]()) {}  
};  
  
int main() {  
    Vector v1 = 10; // Error  
    Vector v2(10);  // ok  
    return 0;  
}
```

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

`auto`

`decltype`

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- **Friends**
- Constant Member Functions Revisited

3 Type Deduction

- `auto`
- `decltype`

Friends

Code inside a **friend** of a class can access the private members of that class.

```
class Vector {
    friend bool equal_to(const Vector &, const Vector &);
    friend class SomeOtherClass;
    // other members
};

inline bool equal_to
    (const Vector &lhs, const Vector &rhs) {
    if (lhs.m_size != rhs.m_size)
        return false;
    for (std::size_t i = 0; i < lhs.m_size; ++i)
        if (lhs.m_data[i] != rhs.m_data[i])
            return false;
    return true;
}
```

Friends

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

- A `friend` declaration is not a member of the class.
- Access-modifiers do not apply to `friend` declarations.
- `friends` are often declared together at the beginning or end of the class.

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

- 1 Copy Control
 - Copying an Object
 - The Copy Constructor
 - The Copy-Assignment Operator
 - Synthesized Copy Operations
 - Prevent Copying
- 2 More about Class
 - Constructors and Type-casting
 - Friends
 - Constant Member Functions Revisited
- 3 Type Deduction
 - auto
 - decltype

Access Elements of Vector

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

```
class Vector {  
public:  
    int &at(std::size_t n) {  
        return m_data[n];  
    }  
    // other members  
};
```

What will happen on a `const` object?

```
void print_vector(const Vector &v) {  
    for (std::size_t i = 0; i < v.size(); ++i)  
        std::cout << v.at(i) << " "; // Error!  
}
```

Access Elements of Vector

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

```
class Vector {  
public:  
    int &at(std::size_t n) const {  
        return m_data[n];  
    }  
    // other members  
};
```

Access Elements of Vector

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction
auto
decltype

```
class Vector {  
public:  
    int &at(std::size_t n) const {  
        return m_data[n];  
    }  
    // other members  
};
```

Still problematic:

```
const Vector v = some_value();  
v.at(10) = 42;
```

Compilers may fail to detect such modification, but it is undefined behavior!

Correct Way

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction
auto
decltype

Const overloading.

```
class Vector {  
public:  
    int &at(std::size_t n) {  
        return m_data[n];  
    }  
    const int &at(std::size_t n) const {  
        return m_data[n];  
    }  
    // other members  
};
```

Calling a `const` member function is actually **adding low-level `const`** to the `this` pointer.

Bitwise `const` vs Logical `const`

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

`auto`

`decltype`

- A member function is bitwise-`const` if it does not modify any data member.
- A member function is logical-`const` if it makes the object *appear* unchanged to users.
 - A logical-`const` member function should prevent potential modification.
 - A logical-`const` member function may modify some data member, but the object seems unchanged to users.

The compiler can only check bitwise constness.

Bitwise `const` vs Logical `const`

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

`auto`

`decltype`

Bitwise-`const` but not logical-`const` :

```
class Vector {  
    public:  
        int &at(std::size_t n) const {  
            return m_data[n];  
        }  
};
```

Directly returning a non-const reference to a data member is not allowed, but compilers may fail to detect this one.

mutable Member

What if we want to count how many times the function is called?

```
class Vector {
    int access_cnt;
public:
    int &at(std::size_t n) {
        ++access_cnt;
        return m_data[n];
    }
    const int &at(std::size_t n) const {
        ++access_cnt;    // Oops! It is not bitwise-const!
        return m_data[n];
    }
};
```

CS100
Recitation 8
GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto
decltype

mutable Member

Define a member to be `mutable`, so that it is modifiable even in a `const` member function.

```
class Vector {  
    mutable int access_cnt;  
public:  
    int &at(std::size_t n) {  
        ++access_cnt;  
        return m_data[n];  
    }  
    const int &at(std::size_t n) const {  
        ++access_cnt;  
        return m_data[n];  
    }  
};
```

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

auto

decltype

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

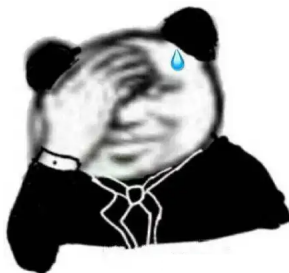
- **auto**
- **decltype**

Before C++11...

In C and before C++11, the `auto` specifier is used like this:

```
auto int x = 42;
```

It indicates 'automatic storage duration': the variable `x` should be destroyed at the end of its scope...



CS100
Recitation 8
GKxx

Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

Type Deduction

- `auto`
- `decltype`

Since C++11

Since C++11, the `auto` specifier is used to let the compiler deduce the type:

```
auto x = 42;    // x is int
auto y = 3.14;  // y is double
```

The variable declared by `auto` must be explicitly initialized, so that the compiler can know the type.

```
auto z;        // ???
```

If more than one variables are declared in one statement, all of them should be explicitly initialized, and the initializers should have a same type.

```
auto a = 1, b;  // Error.
auto c = 42, d = 3.14; // Error.
```


The `auto` Type Specifier

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

`auto`

`decltype`

Compound types are allowed:

```
std::string s = "Hello";  
const auto &sr = s; // const std::string &  
auto *p = &s;      // std::string *
```

The `auto` Type Specifier

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type

Deduction

`auto`

`decltype`

Compound types are allowed:

```
std::string s = "Hello";  
const auto &sr = s; // const std::string &  
auto *p = &s;      // std::string *
```

Low-level `const` will be preserved, but reference and top-level `const` will be ignored (Why?):

```
const int ci = 42;  
auto i2 = ci;      // int  
auto *pi = &ci;    // const int *  
auto &ri = ci;     // const int &  
auto i3 = ri;      // int; & and const are ignored.
```

Why we Need `auto` ?

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

`auto`
`decltype`

The C++ type system is much more complex than C.

- Sometimes the name of the type is too long, e.g.

```
std::shared_ptr<std::map<std::string, std::vector<  
    std::string>::size_type>>
```

- Sometimes we have no idea what type it is:

```
// a lambda expression  
auto f = [](int a, int b) -> int { return a + b; };
```

The compiler knows everything! (C++ is statically-typed!)

Contents

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

`auto`

`decltype`

1 Copy Control

- Copying an Object
- The Copy Constructor
- The Copy-Assignment Operator
- Synthesized Copy Operations
- Prevent Copying

2 More about Class

- Constructors and Type-casting
- Friends
- Constant Member Functions Revisited

3 Type Deduction

- `auto`
- `decltype`

The `decltype` Type Specifier

CS100

Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

`auto`

`decltype`

Sometimes we want to use the **type** of an expression to declare an object, but we don't want to evaluate that expression.

```
int fun() {  
    // do something  
}  
  
decltype(f()) x;    // x is int, but f is not called!
```

The compiler can know the type of the expression without evaluating it. (statically-typed!)

decltype is Honest

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

`decltype` does not ignore top-level `const` or reference. It just says the exact type of the expression.

```
const int i = 0;    // decltype(i) is const int
int f(const Widget &w);
// decltype(w) is const Widget &
// decltype(f) is int(const Widget &)
const std::string s = "Hello";
// decltype(s) is const std::string
// decltype(s[0]) is const char &
```

decltype is Honest

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about
Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type
Deduction

auto

decltype

When `decltype` is used on an expression that yields an `lvalue`, the result is a reference (Why?).

```
int i = 42, *p = &i;  
int j = 50;  
decltype(*p) r = j;    // r is int & bound to j.
```

An object itself can be used as an expression (which yields an `lvalue!`), but you should place it in a pair of parentheses to show this explicitly.

```
decltype(i) k = 65;    // k is int  
decltype((i)) l = k;  // l is int & bound to k.
```

Reading Materials

CS100
Recitation 8

GKxx

Copy Control

Copying an Object

The Copy
Constructor

The
Copy-Assignment
Operator

Synthesized Copy
Operations

Prevent Copying

More about Class

Constructors and
Type-casting

Friends

Constant Member
Functions Revisited

Type Deduction

`auto`

`decltype`

C++ has a very complex type system and type deduction rules. These rules are explained thoroughly in *Effective Modern C++* [Item 1](#), [2](#), [3](#), [4](#), [5](#), [6](#). But it is too early to understand them for you now.