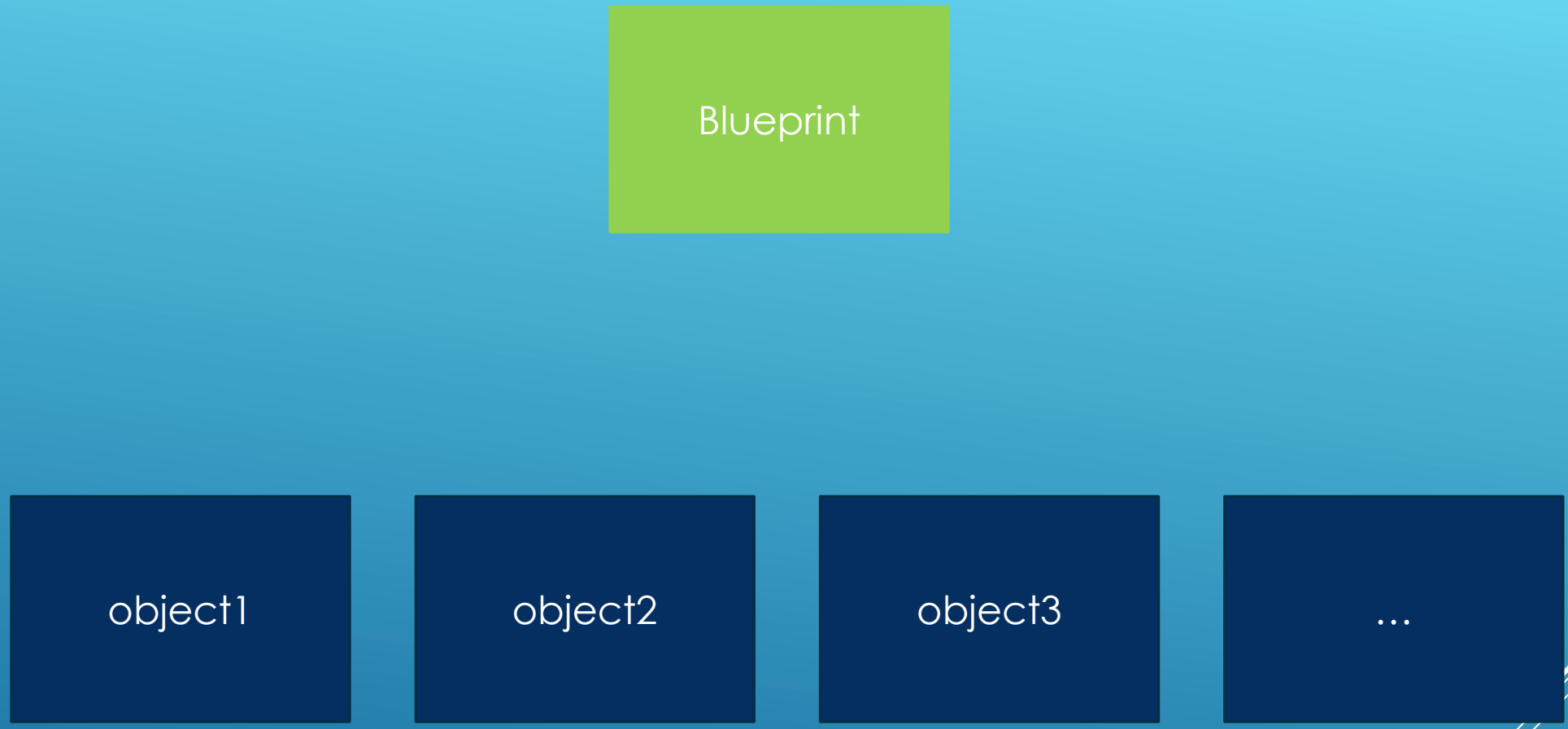
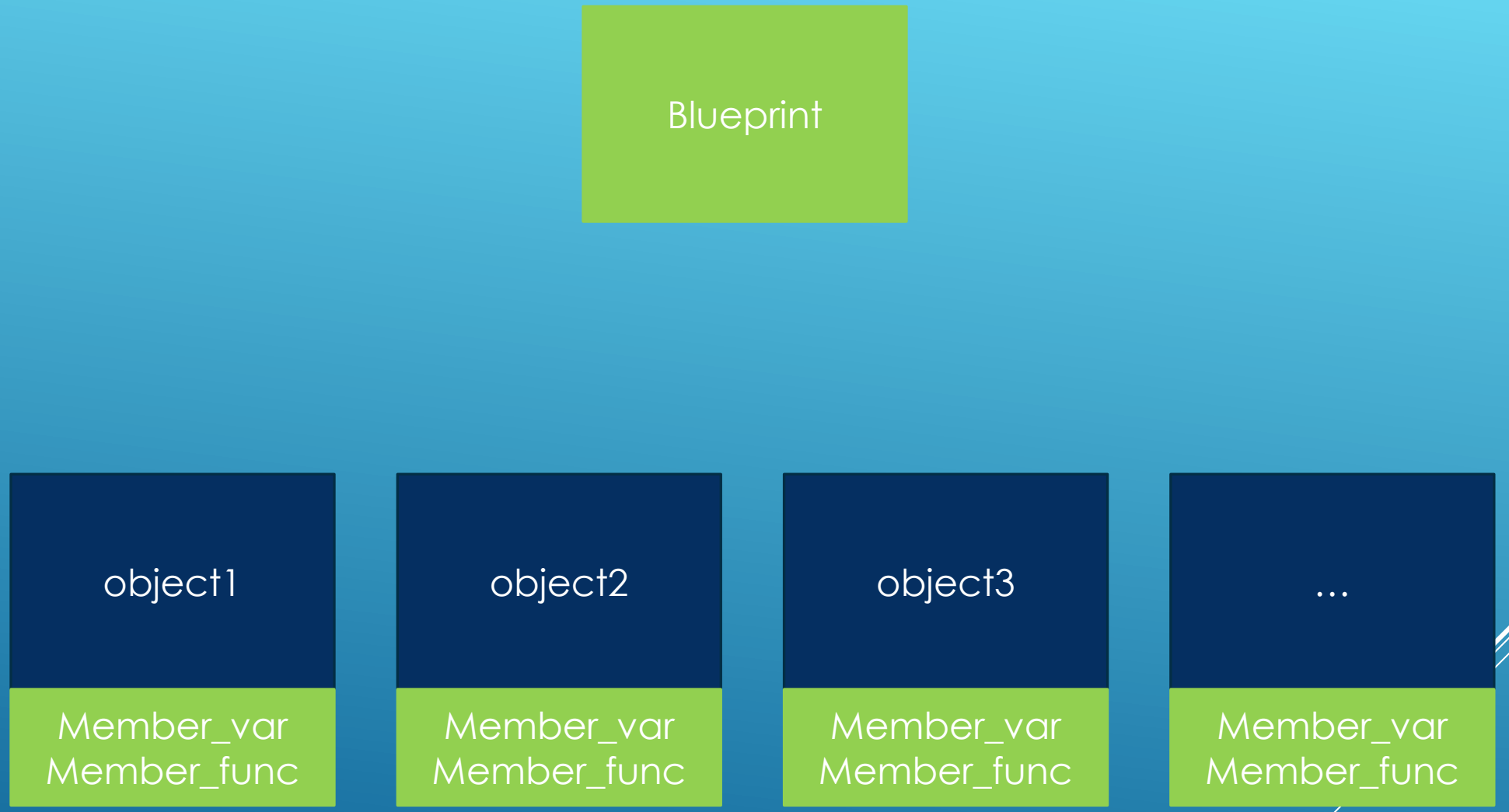
Slides

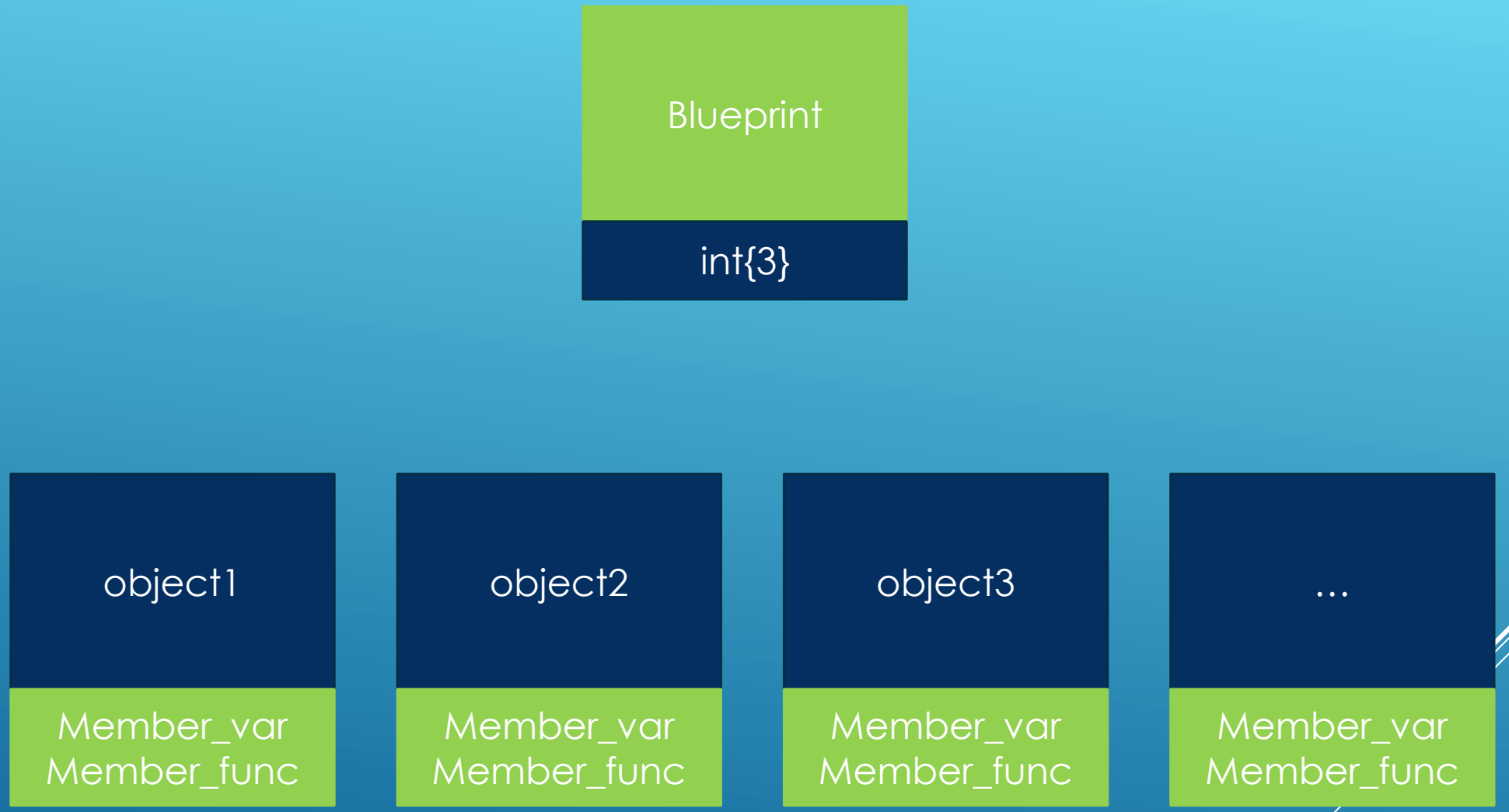
Section : Static Members

Slide intentionally left empty

Static and const members : Introduction


Blueprint
object1
object2
object3
...


Blueprint
object1
Member_var
Member_func
object2
Member_var
Member_func
object3
Member_var
Member_func
...
Member_var
Member_func


Blueprint
int{3}
object1
Member_var
Member_func
object2
Member_var
Member_func
object3
Member_var
Member_func
...
Member_var
Member_func

Slide intentionally left empty

Static Members

```
class Point {
public :

private :
        double x;
        double y;
};
```

```
class Point {
public :

private :
        double x;
        double y;
};
```

Point p1;

Point p2;

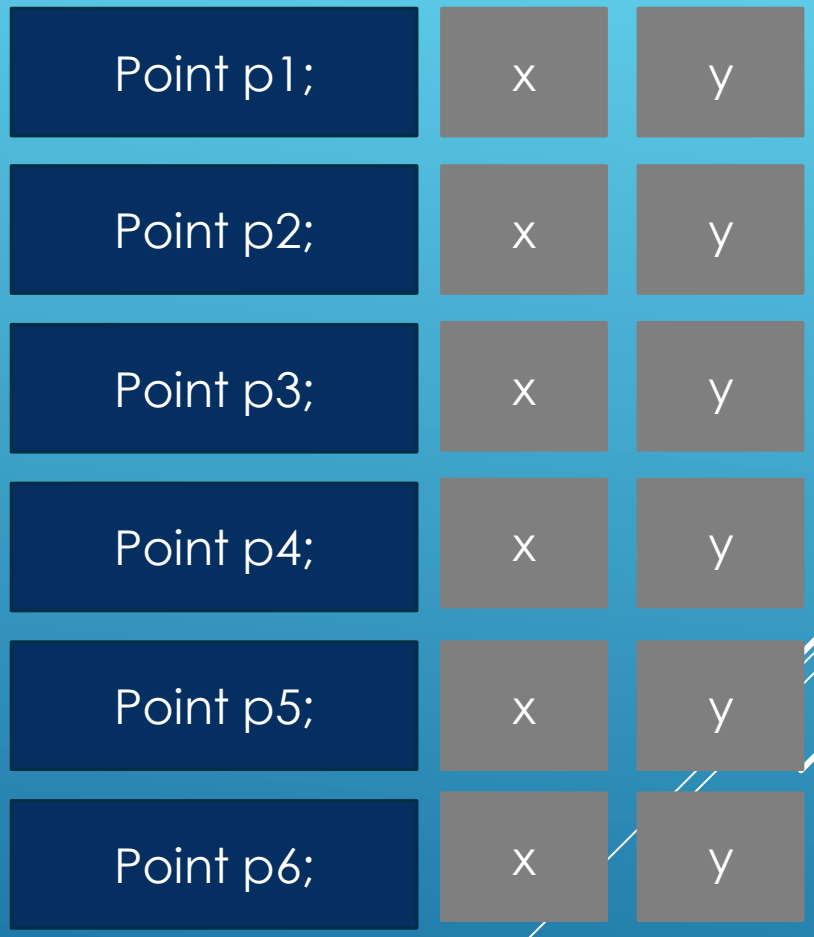
Point p3;

Point p4;

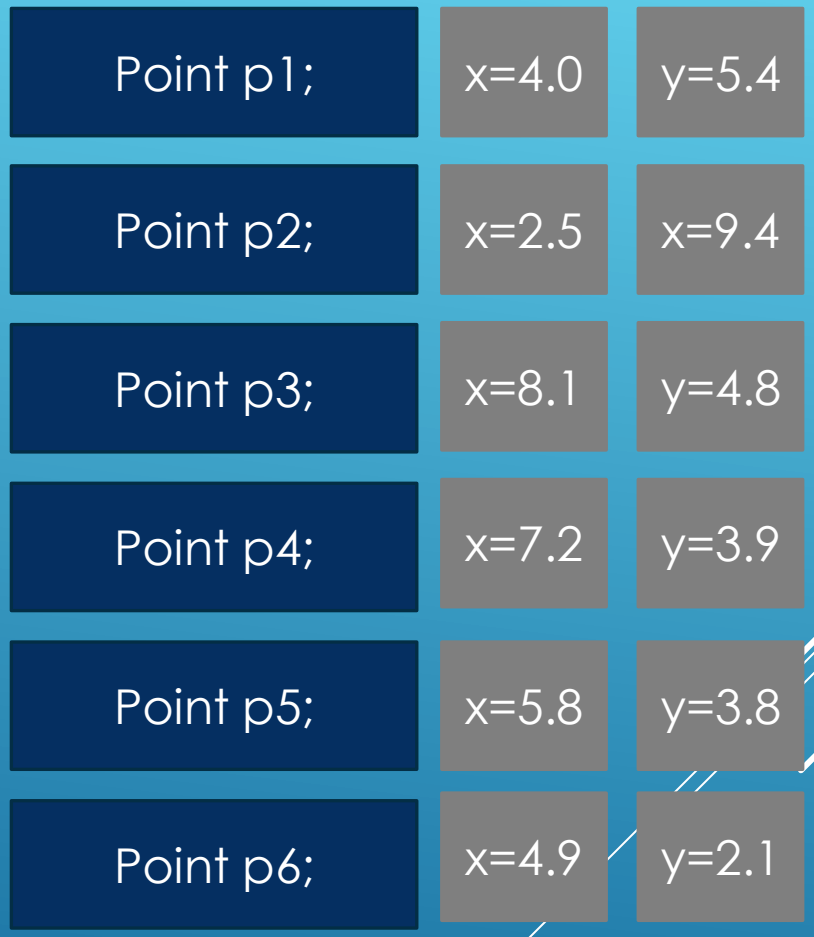
Point p5;

Point p6;

```
class Point {
public :

private :
        double x;
        double y;
};
```



```
class Point {
public :

private :
        double x;
        double y;
};
```



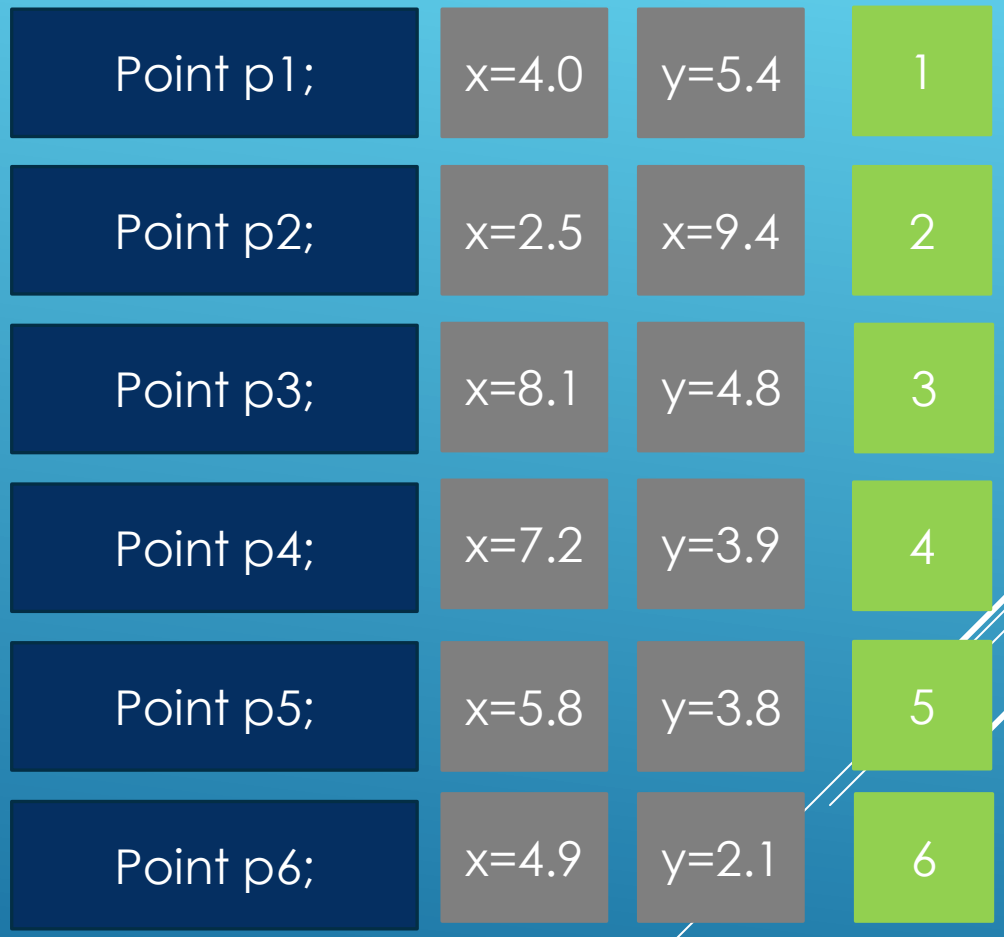
Regular member variables are associated with objects. They belong to class objects.

What if we need member variables that are not tied to any object, but the class blueprint itself

```
class Point {
public :
size_t point_count{};

private :
      double x;
      double y;
};
```

Point p1;	x=4.0	y=5.4
Point p2;	x=2.5	x=9.4
Point p3;	x=8.1	y=4.8
Point p4;	x=7.2	y=3.9
Point p5;	x=5.8	y=3.8
Point p6;	x=4.9	y=2.1

```
class Point {
public :
size_t point_count{};

private :
        double x;
        double y;
};
```



Slide intentionally left empty

Static member variables

Static member variables

Member variables not tied to any object of the class. They live in the context of object blueprints. The are created even before a single class object has been created.

Declare a static member variable

```
class Point
{
public:
  //Constructors
  Point(double x, double y);
  Point(double xy_coord);        // Point Constructor
  Point();                     // Default constructor
  Point(const Point& point);     // Point Copy Constructor
  ~Point();                      // Point Destructor
  double length() const;   // Function to calculate distance from the point(0,0)

  size_t get_point_count() const{
     return m_point_count;
  }

private:
  double m_x;
  double m_y;

public:
 static size_t m_point_count;

};
```

Initialization of static variable

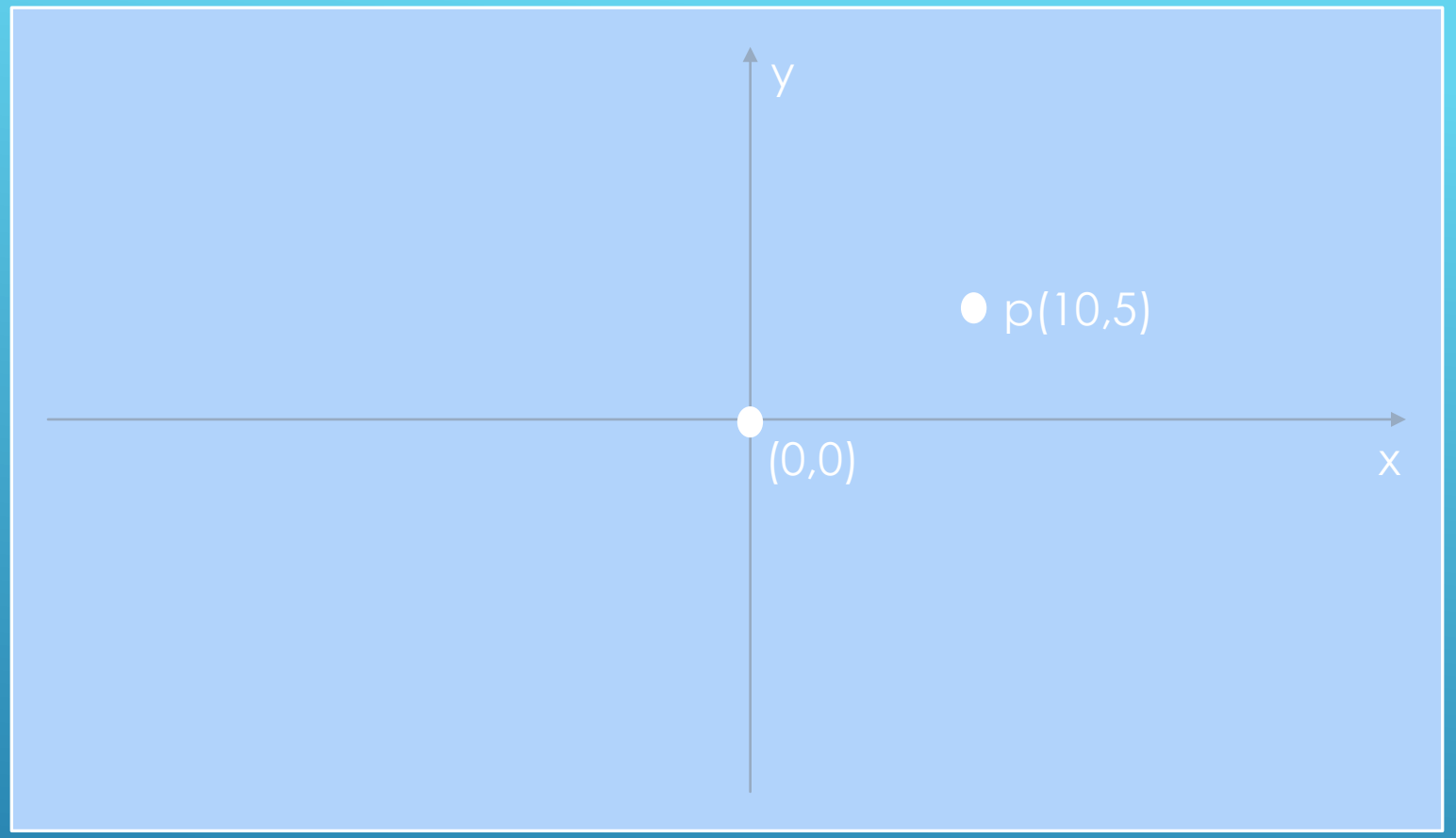
```
#include "point.h"
#include <iostream>
#include <cmath>

size_t Point::m_point_count {}; // Initialize static member of Point class to 0

/* ...
```

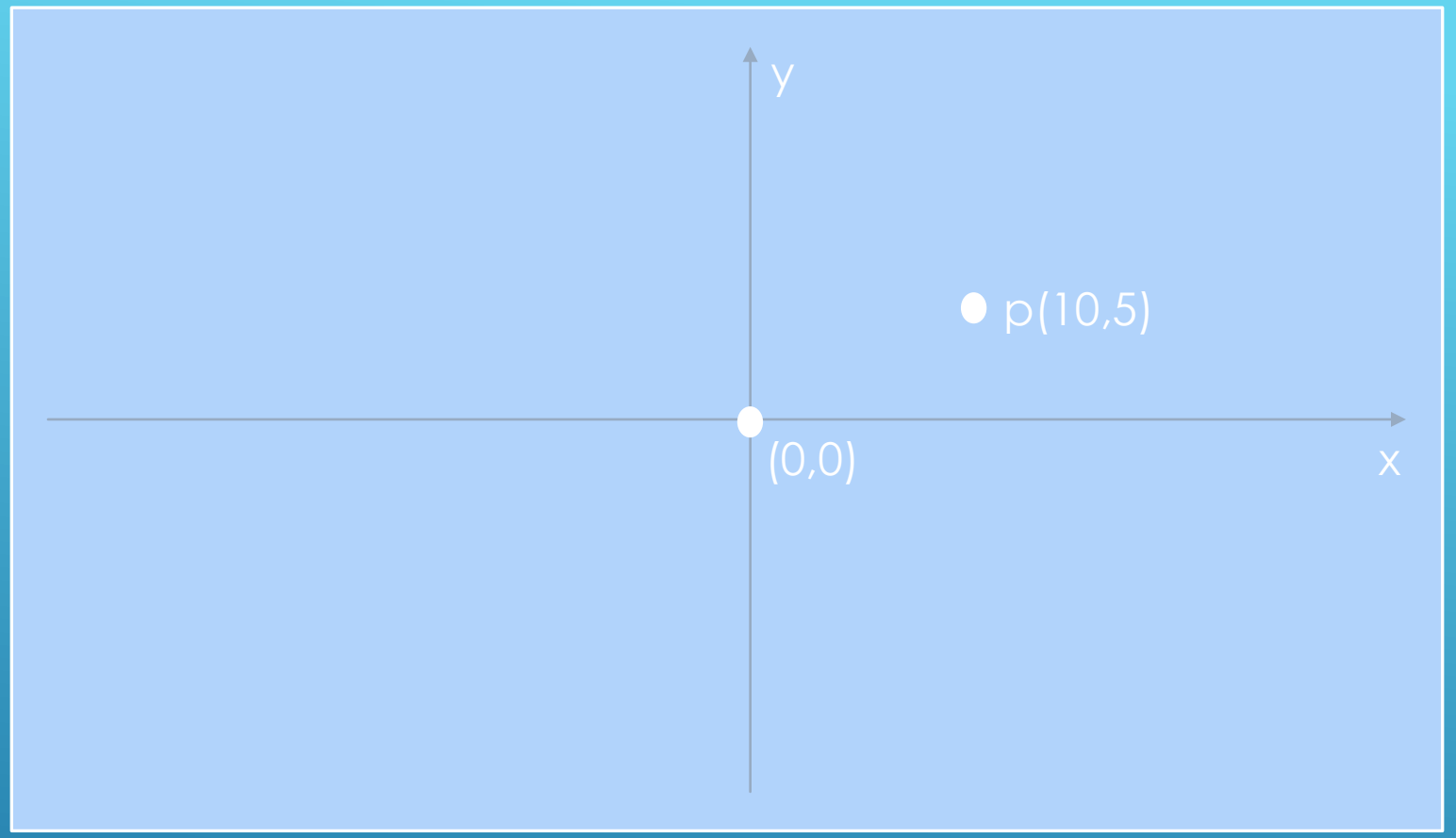
Length : Distance from Point(0.0,0.0)

```
double Point::length() const{
    return sqrt(pow(m_x - 0, 2) +  pow(m_y - 0, 2) * 1.0);
}
```

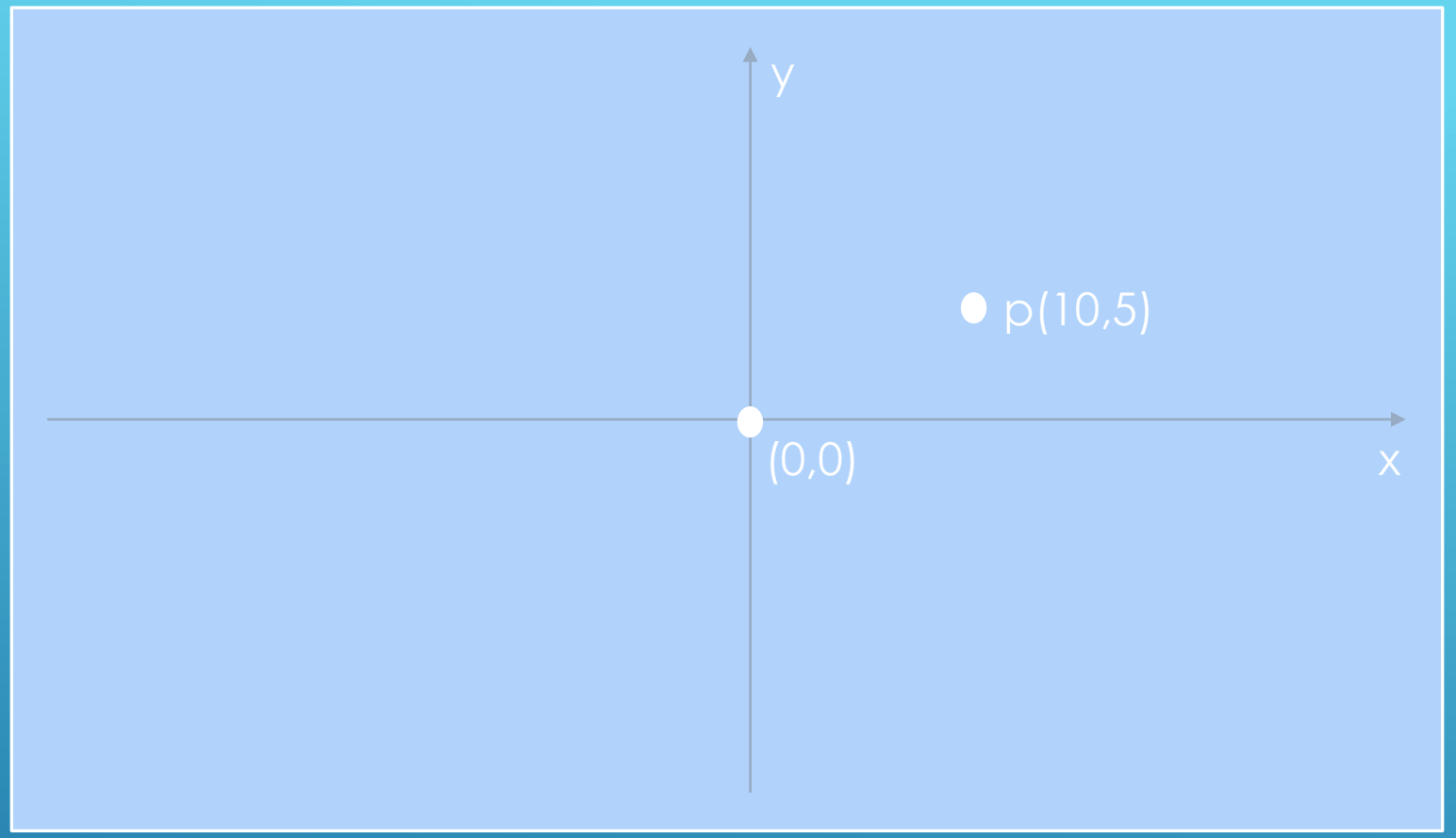
Distance

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$


y
p(10,5)
(0,0)
x



Distance $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$




```
double Point::length() const{
    return sqrt(pow(m_x - 0, 2) +  pow(m_y - 0, 2) * 1.0);
}
```

Slide intentionally left empty

Inline static member variables

Inline static member variables

```
class Point
{
public:
  //Constructors
  Point(double x, double y);
  Point(double xy_coord);        // Point Constructor
  Point();                     // Default constructor
  Point(const Point& point);     // Point Copy Constructor
  ~Point();                      // Point Destructor
  double length() const;   // Function to calculate distance from the point(0,0)

  size_t get_point_count() const{
      return m_point_count;
  }
private:
  double m_x;
  double m_y;
//public:
 inline static size_t m_point_count{};
};
```

Slide intentionally left empty

Static constants

```
class Cylinder {

public :
    //Static constants : public
    static inline const std::string default_color{"Red"};

    //Constructors
    Cylinder()= default;
    Cylinder(double radius_param , double height_param );
private :
    double base_radius{1.0};
    double height{1.0};
    static inline const double PI {3.14159265358979323846264338327795};
};
```

Slide intentionally left empty

Static constants(Pre C++17)

Only integral types int and enum are initializable from the class declaration, others have to be initialized in some cpp file.

Different static constants

```
class Cylinder {
public :
    //int and enum can be initialized in header.
    static const int    INT_CONSTANT = 5;
    enum Color{ Red = 0, Green, Blue };
    static const Color COLOR_CONSTANT= Color::Green;

    //Others have to be done in the cpp file.
    static const std::string default_color;
    static const char*  CHAR_PTR_CONSTANT;
    static const int    INT_ARRAY_CONSTANT[5];
    static const float  FLOAT_CONSTANT;
    static const std::string STRING_ARRAY_CONSTANT[2];

    //Not static
    const float WEIRD_FLOAT;
    const char * WEIRD_C_STRING;
    const int    WEIRD_INT_ARRAY_CONSTANT[5];
    /* ...
private :
    double base_radius{1.0};
    double height{1.0};
    static  const double PI ;
};
```

Different static constants

```
const std::string Cylinder::default_color {"Red"};
const double Cylinder::PI {3.14159265358979323846264338327795};
const char*  Cylinder::CHAR_PTR_CONSTANT    = "CString here";
const int    Cylinder::INT_ARRAY_CONSTANT[] = {10,20,30,40,50};
const float  Cylinder::FLOAT_CONSTANT    = 6.98f;
const std::string Cylinder::STRING_ARRAY_CONSTANT[] = {"String1","String2"};



Cylinder::Cylinder(double radius_param , double height_param )
    :   WEIRD_FLOAT(33.3),
        WEIRD_C_STRING("Weird"),
        WEIRD_INT_ARRAY_CONSTANT{10,20,30,40,50}


{
    //WEIRD_FLOAT = 33.3;// Not allowed

    base_radius = radius_param;
    height = height_param;
}
```

WARNING : Static initialization Order

Static initialization order is not guaranteed. If have static variables that depend on other static variables, you may get crashes for your app if the initialization order doesn't work in your favor.

Slide intentionally left empty

Member variables of type self

Members of type self

```
class Point
{
public:
  //Constructors
  Point(double x, double y);
private:
  double m_x;
  double m_y;
    static size_t m_point_count;
public:
    Point m_origin_point1; // Incomplete type compiler error
    const Point m_origin_point2; // // Incomplete type compiler error
    static inline const Point m_origin_point3; //Incomplete type compiler error
};
```

Member of pointer to self

```
class Point
{
public:
  //Constructors
  Point(double x, double y);

private:
  double m_x;
  double m_y;
    static size_t m_point_count;
public:
    Point * p_m_origin_point4;
};
```

Member of pointer to self

```
Point::Point(double x, double y)
    :    m_x(x) , m_y(y),p_m_origin_point4(nullptr)
{
    std::cout << "Constructing Point [ m_x : " << m_x << ", m_y : " << m_y << "]" << std::endl;
    ++m_point_count;
}
```

WATCH OUT! Infinite recursive calls to constructor

```
Point::Point(double x, double y)
    :    m_x(x) , m_y(y),p_m_origin_point4(new Point())
{
    std::cout << "Constructing Point [ m_x : " << m_x << ", m_y : " << m_y << "]" << std::endl;
    ++m_point_count;
}
```

Old school static variables

```
class Point
{
public:

  //Constructors
  Point(double x, double y);
private:
  double m_x;
  double m_y;
    static size_t m_point_count;
public:
    static Point m_origin_point5;
    static const Point m_origin_point6;
};
```

```
Point Point::m_origin_point5 {1.0,1.0};

const Point Point::m_origin_point6 {2.0, 2.0};

Point::Point(double x, double y)
    :    m_x(x) , m_y(y),p_m_origin_point4(nullptr)
{
    std::cout << "Constructing Point [ m_x : " << m_x << ", m_y : " << m_y << "]" << std::endl;
    ++m_point_count;
}
```

Slide intentionally left empty

Member variables of other types(Object)

```
class Integer
{
public:
    explicit Integer(int value);
    Integer()  = default;
    ~Integer();

    int get_value () const{
        return inner_int;
    }

    void set_value(int new_val){
        inner_int = new_val;
    }

private :
    int inner_int{0};

};
```

Other objects as member variables

```
class Point
{
public :
  //Constructors
  Point(double x, double y);
  /* ...
    // Members of other objects
    Integer i1{1};
    const Integer i2{2};
    static inline Integer i3{3};
    static inline const Integer i4{4};

    Integer* p_i5{nullptr};
    static Integer i6;
    static const Integer i7;
private:
    double m_x;
    double m_y;
    static size_t m_point_count;

 };
```

```
Integer Point::i6 {6};
const Integer Point::i7 {7};


Point::Point(double x, double y)
    :   p_i5(new Integer(5)),
        m_x(x) , m_y(y)
{
    std::cout << "Constructing Point [ m_x : " << m_x << ", m_y : " << m_y << "]" << std::endl;
    ++m_point_count;
}
```

Slide intentionally left empty

Static methods

Static member functions

```
class Point
{
public :
    //Constructors
    Point(double x, double y);
      double length() const;
      static size_t get_point_count(){
          return m_point_count;
      }
      static void print_point_info(const Point& p){
        std::cout << "Point : [ m_x : " << p.m_x << ", m_y : " << p.m_y << "]"
             << std::endl;
      }
private:
    double m_x;
    double m_y;
    static size_t m_point_count;
};
```

Slide intentionally left empty

Nested classes

Inner is nested in Outer

```
class Outer
{
public:
    Outer(int int_param, double double_param);
    Outer();
    ~Outer();
    void do_something();

private :
    int m_var1;
    double m_var2;
    inline static int static_int{45};
//public :
    class Inner{
        public:
            Inner(double double_param);
    private :
            double inner_var;
    };
};
```

Some facts about nested classes

- When Inner is private, its objects can't be created from the outside, like in main
- Outer doesn't have access to private section of Inner
- Inner has access to private section of Outer
- Inner can directly access static members of Outer, but can't access member variables without going through an object

Slide intentionally left empty

In class member initialization

In class member initialization

```
private :
    //Non initialized : will constain junk
    /*
    double m_x ; // Wold cause an error if compiled in pre C++11 mode
    double m_y ;
    */

    //Assignment
    /*
    double m_x =0.0; // Wold cause an error if compiled in pre C++11 mode
    double m_y = 0.0;
    */

    //Empty brace initialization
    /*
    double m_x {}; // Wold cause an error if compiled in pre C++11 mode
    double m_y {};
    */

    //Explicit brace initialization
    double m_x {0.0}; // Wold cause an error if compiled in pre C++11 mode
    double m_y {0.0};

    //Can even in class initialize custom user defined types, like Integer
    Integer i{}; // Will be default initialized.
```

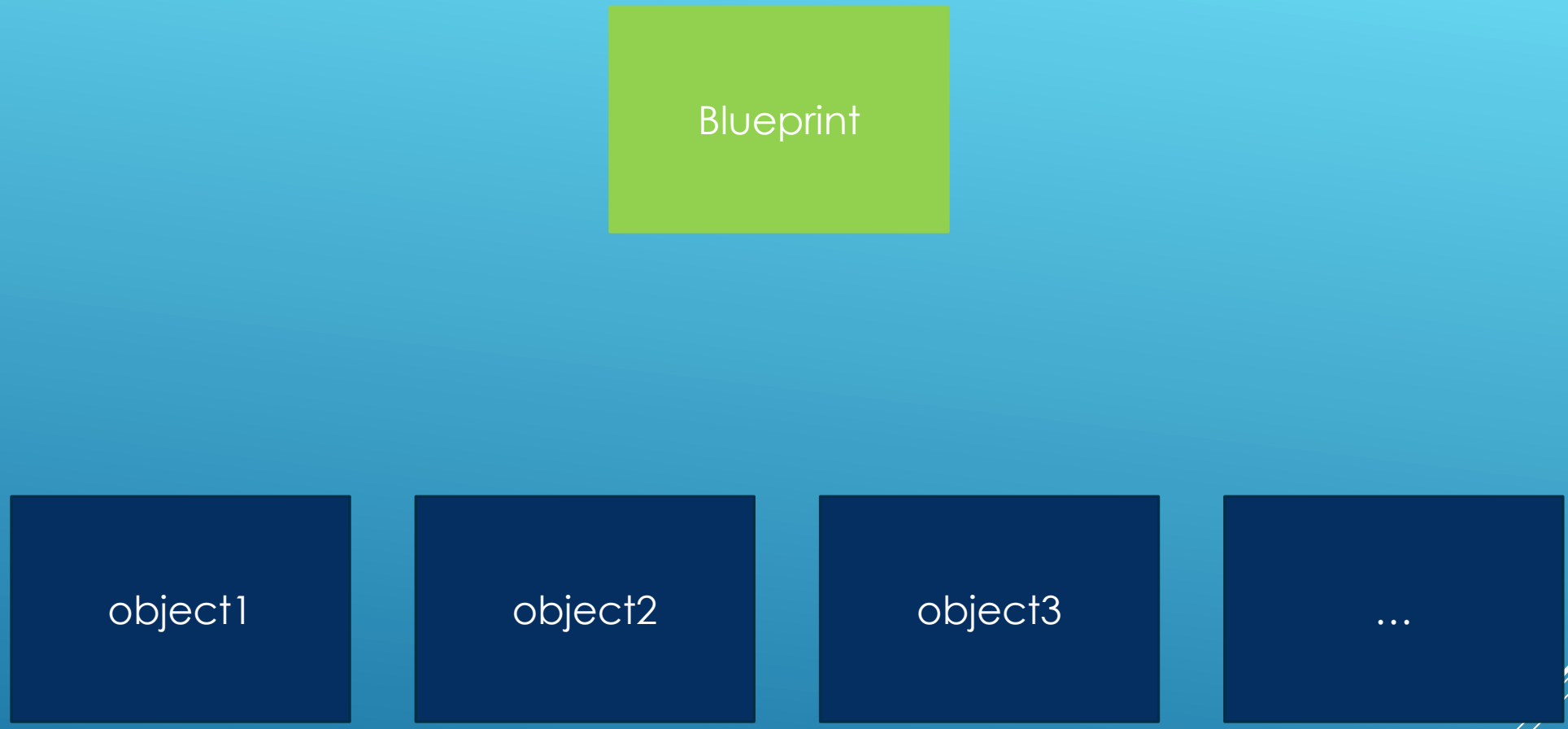
In class member initialization : only widely supported since C++11

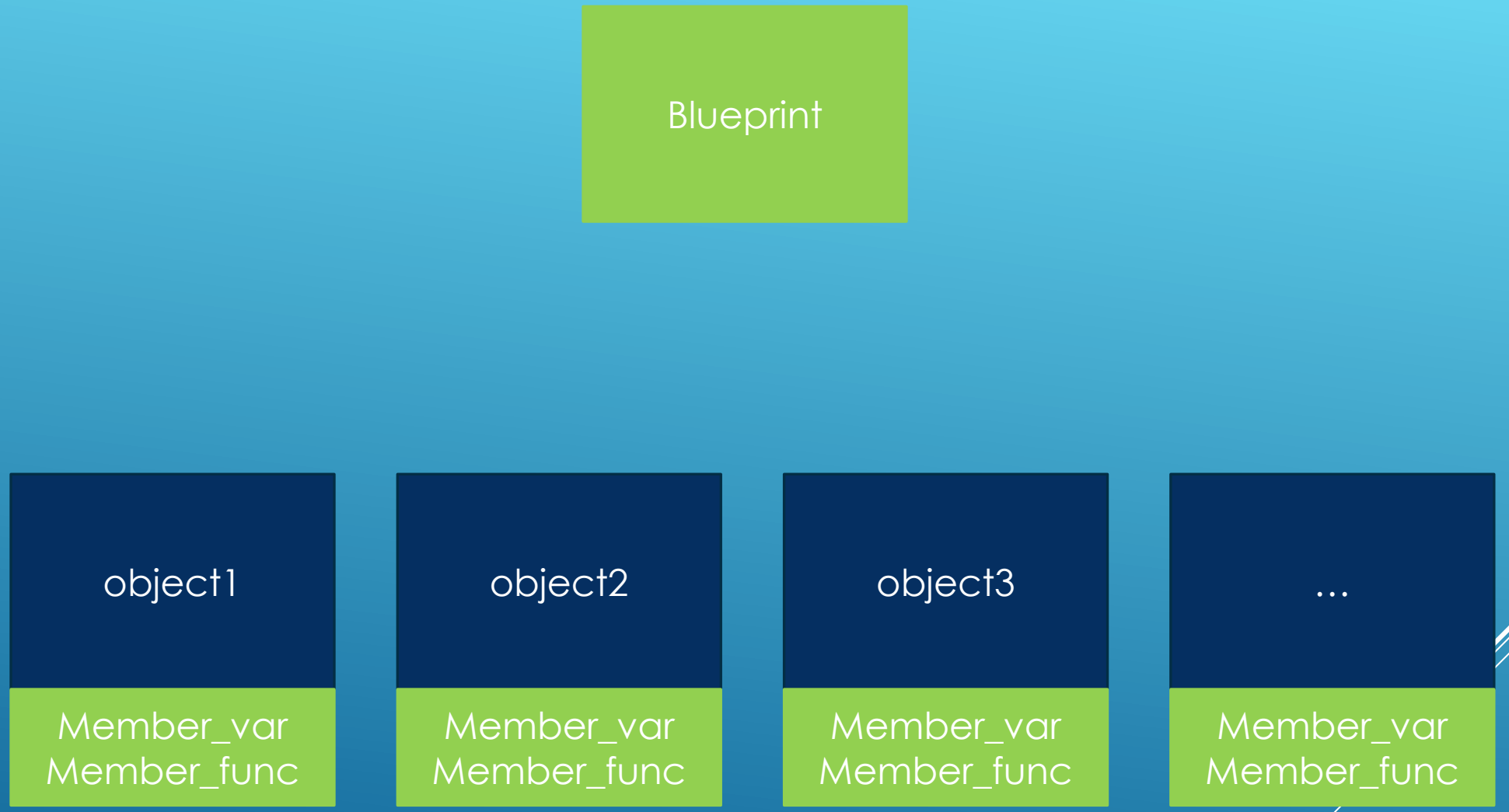
Before C++11, in class member initialization was only possible for

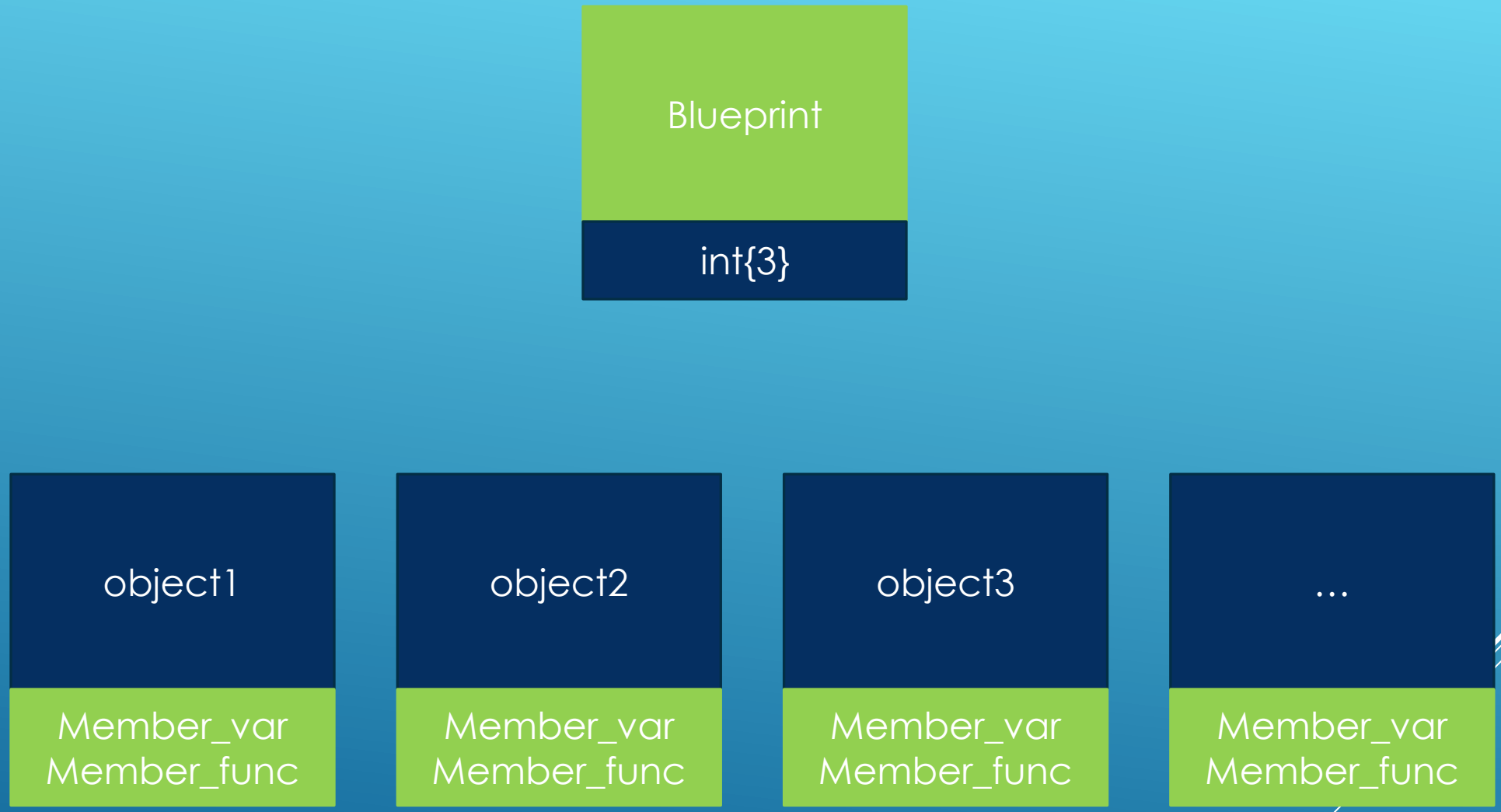
- Static constants of integral types or
- Static constants of enum type

Slide intentionally left empty

Static and const members : Summary


Blueprint
object1
object2
object3
...


Blueprint
object1
Member_var
Member_func
object2
Member_var
Member_func
object3
Member_var
Member_func
...
Member_var
Member_func


Blueprint
int{3}
object1
Member_var
Member_func
object2
Member_var
Member_func
object3
Member_var
Member_func
...
Member_var
Member_func

Declare a static member variable

```
class Point
{
public:
  //Constructors
  Point(double x, double y);
  Point(double xy_coord);        // Point Constructor
  Point();                     // Default constructor
  Point(const Point& point);     // Point Copy Constructor
  ~Point();                      // Point Destructor
  double length() const;   // Function to calculate distance from the point(0,0)

  size_t get_point_count() const{
     return m_point_count;
  }

private:
  double m_x;
  double m_y;

public:
 static size_t m_point_count;

};
```

Initialization of static variable

```
#include "point.h"
#include <iostream>
#include <cmath>

size_t Point::m_point_count {}; // Initialize static member of Point class to 0

/* ...
```

Inline static member variables

```
class Point
{
public:
  //Constructors
  Point(double x, double y);
  Point(double xy_coord);        // Point Constructor
  Point();                     // Default constructor
  Point(const Point& point);     // Point Copy Constructor
  ~Point();                      // Point Destructor
  double length() const;   // Function to calculate distance from the point(0,0)

  size_t get_point_count() const{
      return m_point_count;
  }
private:
  double m_x;
  double m_y;
//public:
 inline static size_t m_point_count{};
};
```

Static constants

```
class Cylinder {

public :
    //Static constants : public
    static inline const std::string default_color{"Red"};

    //Constructors
    Cylinder()= default;
    Cylinder(double radius_param , double height_param );
private :
    double base_radius{1.0};
    double height{1.0};
    static inline const double PI {3.14159265358979323846264338327950};
};
```

Different static constants before C++17

```
class Cylinder {
public :
    //int and enum can be initialized in header.
    static const int    INT_CONSTANT = 5;
    enum Color{ Red = 0, Green, Blue };
    static const Color COLOR_CONSTANT= Color::Green;

    //Others have to be done in the cpp file.
    static const std::string default_color;
    static const char*  CHAR_PTR_CONSTANT;
    static const int    INT_ARRAY_CONSTANT[5];
    static const float  FLOAT_CONSTANT;
    static const std::string STRING_ARRAY_CONSTANT[2];

    //Not static
    const float WEIRD_FLOAT;
    const char * WEIRD_C_STRING;
    const int    WEIRD_INT_ARRAY_CONSTANT[5];
    /* ...
private :
    double base_radius{1.0};
    double height{1.0};
    static  const double PI ;
};
```

Static constants before C++17

```
const std::string Cylinder::default_color {"Red"};
const double Cylinder::PI {3.14159265358979323846264338327795};
const char*  Cylinder::CHAR_PTR_CONSTANT    = "CString here";
const int    Cylinder::INT_ARRAY_CONSTANT[] = {10,20,30,40,50};
const float  Cylinder::FLOAT_CONSTANT    = 6.98f;
const std::string Cylinder::STRING_ARRAY_CONSTANT[] = {"String1","String2"};



Cylinder::Cylinder(double radius_param , double height_param )
    :   WEIRD_FLOAT(33.3),
        WEIRD_C_STRING("Weird"),
        WEIRD_INT_ARRAY_CONSTANT{10,20,30,40,50}


{
    //WEIRD_FLOAT = 33.3;// Not allowed

    base_radius = radius_param;
    height = height_param;
}
```

Member of pointer to self

```
class Point
{
public:
  //Constructors
  Point(double x, double y);

private:
  double m_x;
  double m_y;
    static size_t m_point_count;
public:
    Point * p_m_origin_point4;
};
```

Other objects as member variables

```
class Point
{
public :
  //Constructors
  Point(double x, double y);
  /* ...
    // Members of other objects
    Integer i1{1};
    const Integer i2{2};
    static inline Integer i3{3};
    static inline const Integer i4{4};

    Integer* p_i5{nullptr};
    static Integer i6;
    static const Integer i7;
private:
    double m_x;
    double m_y;
    static size_t m_point_count;

 };
```

Static member functions

```
class Point
{
public :
    //Constructors
    Point(double x, double y);
      double length() const;
      static size_t get_point_count(){
          return m_point_count;
      }
      static void print_point_info(const Point& p){
        std::cout << "Point : [ m_x : " << p.m_x << ", m_y : " << p.m_y << "]"
             << std::endl;
      }
private:
    double m_x;
    double m_y;
    static size_t m_point_count;
};
```

Inner is nested in Outer

```
class Outer
{
public:
    Outer(int int_param, double double_param);
    Outer();
    ~Outer();
    void do_something();

private :
    int m_var1;
    double m_var2;
    inline static int static_int{45};
//public :
    class Inner{
        public:
            Inner(double double_param);
    private :
            double inner_var;
    };
};
```

In class member initialization

```
private :
    //Non initialized : will constain junk
    /*
    double m_x ; // Wold cause an error if compiled in pre C++11 mode
    double m_y ;
    */

    //Assignment
    /*
    double m_x =0.0; // Wold cause an error if compiled in pre C++11 mode
    double m_y = 0.0;
    */

    //Empty brace initialization
    /*
    double m_x {}; // Wold cause an error if compiled in pre C++11 mode
    double m_y {};
    */

    //Explicit brace initialization
    double m_x {0.0}; // Wold cause an error if compiled in pre C++11 mode
    double m_y {0.0};

    //Can even in class initialize custom user defined types, like Integer
    Integer i{}; // Will be default initialized.
```

Head to the IDE and show all this off