Slides

Section : Namespaces

Slide intentionally left empty

Namespaces

A facility in the C++ programming language to avoid name conflicts

```
double add(double x, double y){
    return x + y;
}

double add(double x, double y){
    return x+ y - adjustment;
}

double sub(double x, double y){
    return x - y;
}

double sub(double x, double y){
    return x- y - adjustment;
}
```

Slide intentionally left empty

Creating namespaces

```
double add(double x, double y){
    return x + y;
}

double add(double x, double y){
    return x+ y - adjustment;
}

double sub(double x, double y){
    return x - y;
}

double sub(double x, double y){
    return x- y - adjustment;
}
```

```
namespace No_Weight {
    double add(double x, double y){
        return x + y;
    }
}

namespace Weight{
    double add(double x, double y){
        return x+ y - adjustment;
    }
}

namespace No_Weight {
    double sub(double x, double y){
        return x - y;
    }
}

namespace Weight{
    double sub(double x, double y){
        return x- y - adjustment;
    }
}
```

```
namespace Weight{
    double mult(double x, double y);
    double div( double x, double y);
}

int main(){
    /* ...
}

namespace Weight{
    double mult(double x, double y){
        return x*y - adjustment;
    }
    double div( double x, double y){
        return x/y - adjustment;
    }
}
```

```
int main(){

    double result = Weight::div(4,2);
    std::cout << "result : " << result << std::endl;

    result = Weight::div(4,2);
    std::cout << "result : " << result << std::endl;

}
```

Slide intentionally left empty

Namespaces across Multiple Files


Point
Line
Cylinder

point.h

```
namespace Geom{

    class Point
    {
    public:
        Point();
        Point( double x, double y);
        void print_info()const{
            std::cout << "Point [ x : " << m_x << ", y : " << m_y << "]" << std::endl;
        }
        ~Point();
    private :
        double m_x;
        double m_y;
    };

}
```

point.cpp

```
namespace Geom{

    Point::Point(double x, double y) : m_x{x}, m_y{y}{

    }

    Point::Point() : Point(0.0,0.0)
    {
    }

    Point::~Point()
    {
    }

}
```

line.h

```
namespace Geom{

    class Line
    {
    public:
        Line(const Point& start, const Point& end);

        void print_info()const{
            std::cout << "Line from " << std::endl;
            m_start.print_info();
            std::cout << "to : " << std::endl;
            m_end.print_info();
        }
    private :
        Point m_start;
        Point m_end;
    };

}
```

line.cpp

```
namespace Geom{

    Line::Line(const Point& start, const Point& end)
        : m_start{start}, m_end{end}
    {
    }

}
```

cylinder.h

```
namespace Geom{

    class Cylinder
    {
    public:
        Cylinder(double base_rad, double height);

        double volume() const{
            return PI * m_base_rad * m_base_rad * m_height;
        }

    private :
        inline static const double PI {3.14159265358979323846264338327795};
        double m_base_rad{1};
        double m_height{1};
    };

}
```

cylinder.cpp

```
namespace Geom{

    Cylinder::Cylinder(double base_rad, double height)
        : m_base_rad{base_rad} , m_height{height}
    {
    }

}
```

```cpp
#include "point.h"
#include "line.h"
#include "cylinder.h"

int main(){

    Geom::Point p1(10.0,10.0);
    Geom::Point p2(20.1,20.1);
    p1.print_info();
    p2.print_info();

    std::cout << std::endl;

    Geom::Line l1(p1,p2);
    l1.print_info();

    Geom::Cylinder c1(10,1);
    std::cout << "volume : " << c1.volume() << std::endl;

}
```

Slide intentionally left empty

Default Global Namespace

```cpp
double add(double a, double b){
    return a + b;
}

namespace My_Thing{
    double add(double a, double b){
        return a + b - 1;
    }

    void do_something(){
        double result = ::add(5,6);
        std::cout << "result : " << result << std::endl;
    }

}

int main(){

    My_Thing::do_something();

}
```

Slide intentionally left empty

Using declarations

point.h

```
namespace Geom{

    class Point
    {
    public:
        Point();
        Point( double x, double y);
        void print_info()const{
            std::cout << "Point [ x : " << m_x << ", y : " << m_y << "]" << std::endl;
        }
        ~Point();
    private :
        double m_x;
        double m_y;
    };

}
```

point.cpp

```
namespace Geom{

    Point::Point(double x, double y) : m_x{x}, m_y{y}{

    }

    Point::Point() : Point(0.0,0.0)
    {
    }

    Point::~Point()
    {
    }

}
```

line.h

```
namespace Geom{

    class Line
    {
    public:
        Line(const Point& start, const Point& end);

        void print_info()const{
            std::cout << "Line from " << std::endl;
            m_start.print_info();
            std::cout << "to : " << std::endl;
            m_end.print_info();
        }
    private :
        Point m_start;
        Point m_end;
    };

}
```

line.cpp

```
namespace Geom{

    Line::Line(const Point& start, const Point& end)
        : m_start{start}, m_end{end}
    {
    }

}
```

cylinder.h

```
namespace Geom{

    class Cylinder
    {
    public:
        Cylinder(double base_rad, double height);

        double volume() const{
            return PI * m_base_rad * m_base_rad * m_height;
        }

    private :
        inline static const double PI {3.14159265358979323846264338327795};
        double m_base_rad{1};
        double m_height{1};
    };

}
```

cylinder.cpp

```
namespace Geom{

    Cylinder::Cylinder(double base_rad, double height)
        : m_base_rad{base_rad} , m_height{height}
    {
    }

}
```

```cpp
#include "point.h"
#include "line.h"
#include "cylinder.h"

int main(){

    Geom::Point p1(10.0,10.0);
    Geom::Point p2(20.1,20.1);
    p1.print_info();
    p2.print_info();

    std::cout << std::endl;

    Geom::Line l1(p1,p2);
    l1.print_info();

    Geom::Cylinder c1(10,1);
    std::cout << "volume : " << c1.volume() << std::endl;

}
```

Using declarations

```
//using  Geom::Point; // Brings in just the Point name from the namespace
                      // Makes the name directly usable in the global namespace
//using Geom::Line;
using namespace Geom; // Brings in the entire namespace

double add(double a, double b){
    std::cout << "::add" << std::endl;
    return a + b + 0.555;
}

//BAD : Will conflict with the add function from the global namespace
//using namespace Math;
//using Math::add;
```

```
int main(){
    Point p1(10.1,20.1);
    Point p2(20.1,30.1);
    p1.print_info();
    p2.print_info();

    std::cout << "-----" << std::endl;

    Line l1(p1,p2);
    l1.print_info();

    std::cout << "-----" << std::endl;

    double result = add(10,20);
    std::cout << "result : " << result << std::endl;

    std::cout << "-----" << std::endl;

    //using std::cout ;
    //using std::endl;
    //using namespace std; // Not recommended
    cout << "Hello" << endl;
}
```

Slide intentionally left empty

Anonymous Namespaces

```
namespace {
    double add(double a, double b);
}
int main(){
    double result = add(10,20);
    std::cout << "result : " << result << std::endl;
}

namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

- When the compiler sees an anonymous namespace declaration it will generate an internal name for the namespace
- The generated unique namespace name is not accessible to YOU, the developer
- There can only be one anonymous namespace for a single translation unit. If you set up multiple anonymous namespace blocks, they'll just be extensions to the first one
- Anonymous namespaces in different translation units are completely separate though, the compiler generates different unique namespace names for them behind the scenes
- Because we don't have access to the compiler generated namespace name for anonymous namespaces, names declared inside anonymous namespaces are only reachable/usable in the TU where they were declared

file1.cpp

```
namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

file2.cpp

```
namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

file1.cpp

```
namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

file2.cpp

```
namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

file1.cpp

```
namespace abf3l033efa {
    double add(double a, double b){
        return a + b;
    }
}
```

file2.cpp

```
namespace abf3l03ccb1 {
    double add(double a, double b){
        return a + b;
    }
}
```

Names in an anonymous namespace are only reachable/usable from the TU where they were declared/defined

Slide intentionally left empty

Nested namespaces

```
namespace Hello{
    unsigned int age{23};

    namespace World{
        int local_var{44};

        void say_something(){
            std::cout << "Hello there " << std::endl;
            std::cout << "The age is : " << age << std::endl;
        }
    }

    void do_something(){
        std::cout << "Using local_var : " << World::local_var << std::endl;
    }
}
```

```
int main(){

    //Hello::World::say_something();
    Hello::do_something();

}
```

Slide intentionally left empty

Namespace aliases

```cpp
namespace Level1{
    namespace Level2{
        namespace Level3{
            const double weight{33.33};
        }
    }
}

int main(){
    namespace Data = Level1::Level2::Level3;
    std::cout << "weight : " << Level1::Level2::Level3::weight << std::endl;
    std::cout << "weight : " << Data::weight << std::endl;
    std::cout << "weight : " << Data::weight << std::endl;
}
```

Slide intentionally left empty

Namespaces

A facility in the C++ programming language to avoid name conflicts

```
double add(double x, double y){
    return x + y;
}

double add(double x, double y){
    return x+ y - adjustment;
}

double sub(double x, double y){
    return x - y;
}

double sub(double x, double y){
    return x- y - adjustment;
}
```

Crating Namespaces

```
namespace No_Weight {
    double add(double x, double y){
        return x + y;
    }
}

namespace Weight{
    double add(double x, double y){
        return x+ y - adjustment;
    }
}

namespace No_Weight {
    double sub(double x, double y){
        return x - y;
    }
}

namespace Weight{
    double sub(double x, double y){
        return x- y - adjustment;
    }
}
```

Namespaces across Multiple Files

```
#include "point.h"
#include "line.h"
#include "cylinder.h"

int main(){

    Geom::Point p1(10.0,10.0);
    Geom::Point p2(20.1,20.1);
    p1.print_info();
    p2.print_info();

    std::cout << std::endl;

    Geom::Line l1(p1,p2);
    l1.print_info();

    Geom::Cylinder c1(10,1);
    std::cout << "volume : " << c1.volume() << std::endl;

}
```

Global namespace

```
double add(double a, double b){
    return a + b;
}

namespace My_Thing{
    double add(double a, double b){
        return a + b - 1;
    }

    void do_something(){
        double result = ::add(5,6);
        std::cout << "result : " << result << std::endl;
    }

}

int main(){

    My_Thing::do_something();

}
```

Slide intentionally left empty

Built in Namespaces

Using declarations

```
//using  Geom::Point; // Brings in just the Point name from the namespace
                      // Makes the name directly usable in the global namespace
//using Geom::Line;
using namespace Geom; // Brings in the entire namespace

double add(double a, double b){
    std::cout << "::add" << std::endl;
    return a + b + 0.555;
}

//BAD : Will conflict with the add function from the global namespace
//using namespace Math;
//using Math::add;
```

Anonymous namespaces

```
namespace {
    double add(double a, double b);
}
int main(){
    double result = add(10,20);
    std::cout << "result : " << result << std::endl;
}

namespace {
    double add(double a, double b){
        return a + b;
    }
}
```

Nested namespaces

```
namespace Hello{
    unsigned int age{23};

    namespace World{
        int local_var{44};

        void say_something(){
            std::cout << "Hello there " << std::endl;
            std::cout << "The age is : " << age << std::endl;
        }
    }

    void do_something(){
        std::cout << "Using local_var : " << World::local_var << std::endl;
    }
}
```

Namespace aliases

```
namespace Level1{
    namespace Level2{
        namespace Level3{
            const double weight{33.33};
        }
    }
}

int main(){
    namespace Data = Level1::Level2::Level3;
    std::cout << "weight : " << Level1::Level2::Level3::weight << std::endl;
    std::cout << "weight : " << Data::weight << std::endl;
    std::cout << "weight : " << Data::weight << std::endl;
}
```

Slide intentionally left empty