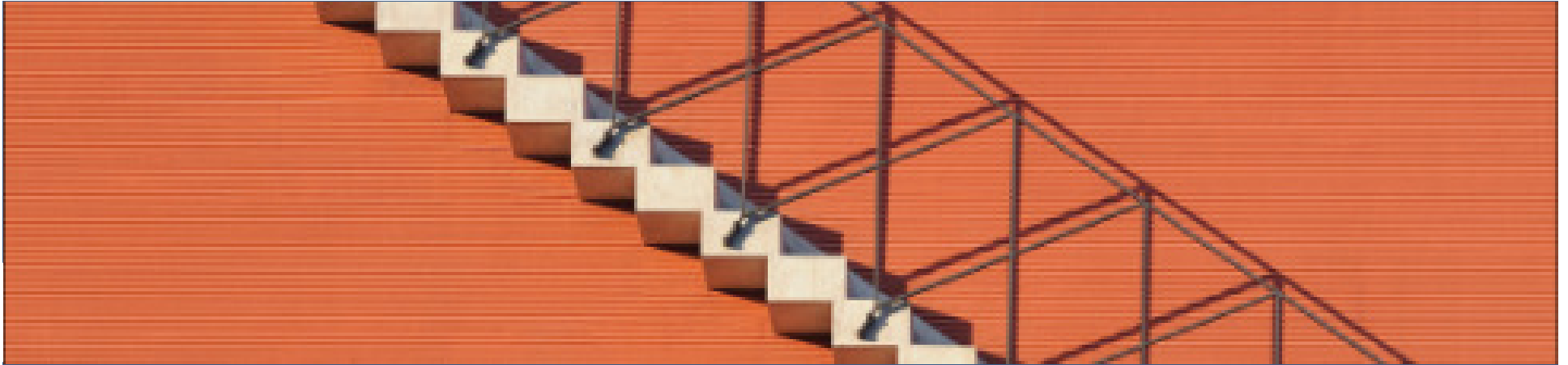




Module 7 b

- Namespaces
- Exceptions handling

C++ Namespace

- Often, a solution to a problem will have **groups of related classes and other declarations**, such as functions, types, and constants.
- C++ provides a mechanism for logically **grouping these declarations and definitions into a common declarative region** known as **namespace**.
- What we have seen so far: **std** namespace

using namespace std;

C++ Namespace (cont)

- We can define our own namespace and include a “using” directive in order to use it. The general syntax is:

```
namespace name_space_name {  
    // place declaration here  
}
```

- The scope of a using directive is the block in which it appears.

C++ Namespace (Example 1)

```
#include <iostream>
#include <string>
```

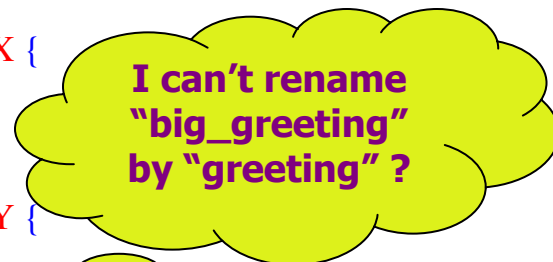
```
using namespace std;
```

```
namespace CS1102CX {
    void greeting ();
}
```

```
namespace CS1102CY {
    void greeting ();
}
```

```
void big_greeting ();
```

```
void greeting (string);
```



```
main () {
```

```
{
    using namespace CS1102CX;
    greeting ();
}
```

```
{
    using namespace CS1102CY;
    greeting ();
}
```

```
big_greeting (); // using namespace global and std
```

```
greeting ("CS1102C"); // using namespace global and std
```

```
}
```

C++ Namespace (Example 1, cont)

```
namespace CS1102CX {  
    void greeting () {  
        cout << "Group X students welcome you" << endl;  
    }  
}
```

```
namespace CS1102CY {  
    void greeting () {  
        cout << "Group Y students welcome you" << endl;  
    }  
}
```

```
void big_greeting () {  
    cout << "We all welcome you" << endl;  
}
```

```
void greeting (string s) {  
    cout << s << " welcome you" << endl;  
}
```

Group X students welcome you
Group Y students welcome you
We all welcome you
CS1102C welcome you

C++ Namespace (Example 1, cont)

// etc etc

```
using namespace CS1102CX;  
using namespace CS1102CY;
```

Another way of calling

```
main () {
```

```
    CS1102CX::greeting ();
```

```
    CS1102CY::greeting ();
```

```
    big_greeting (); // using namespace global and std
```

```
    greeting ("CS1102C"); // using namespace global and std
```

```
}
```

C++ Namespace (Example 2)

```
/* -----  
   A header file.  
  
   File name: Circle.h  
   -----*/  
  
// using namespace std;   → I don't have to put it here  
  
namespace usingCircle {  
    const double PI = 3.14159;  
  
    class Circle {  
        private:  
            int radius;  
  
        public:  
            Circle (int);  
            int getRadius ();  
            double area ();  
    };  
}
```

C++ Namespace (Example 2, cont)

```
/*-----  
    Implement Circle.h header file  
    File name: Circle.cpp  
-----*/
```

```
#include "Circle.h"
```

```
using namespace usingCircle;
```

```
Circle::Circle (int r) {  
    radius = r;  
}
```

```
int Circle::getRadius () {  
    return radius;  
}
```

```
double Circle::area () {  
    return PI * radius * radius;  
}
```

C++ Namespace (Example 2, cont)

```
/* -----  
   File name: testCircle.cpp  
   ----- */  
  
#include <iostream>  
#include "Circle.h"  
using namespace std;  
using namespace usingCircle;  
  
main () {  
    int r;  
  
    cout << "Enter a value ";  
    cin >> r;  
  
    Circle c (r);  
  
    cout << "The area is " << c.area() << endl;  
}
```

C++ Exceptions

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

main () {
    int a, b;

    cout << "Enter a and b: ";
    cin >> a >> b;

    int c = a / b;
    cout << "c is " << c << endl;
}
```



What will happen
if you compile and
run the program?

Enter a and b: **3 0**

Arithmetic Exception

C++ Exceptions (cont)

- An **exception** is a mechanism for handling an error during execution!!
- If we detect an error during an execution, we can **throw an exception**. The code that deals with exception is said **to catch** or handle it.

C++ Exceptions (cont)

- To catch an exception, C++ provides *try-catch* blocks.
- Use a *throw* statement to throw an exception.
- You place a statement that might cause an exception within a *try* block.
- The *try* block must be followed by one or more *catch* blocks.
- Each *catch* block indicates a type of exception you want to handle.
- A *try* block can have many *catch* blocks associated with it.

C++ Exceptions (cont)

General Syntax for try – catch - throw:

```
try {  
    statement (s);  
}
```

```
catch (ExceptionType identifier) {  
    statement (s);  
}
```

```
throw ExceptionType (stringArgument);
```

C++ Exceptions (Example)

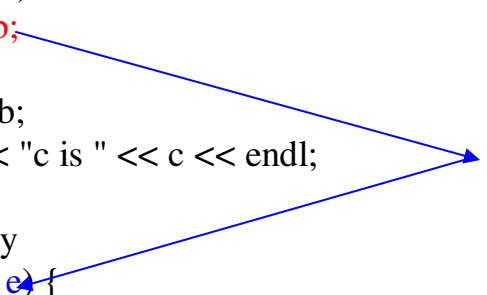
```
#include <iostream>
using namespace std;

main () {
    int a, b, c;

    try {
        cout << "Enter a and b: ";
        cin >> a >> b;

        if (b == 0)
            throw b;
        else {
            c = a / b;
            cout << "c is " << c << endl;
        }
    } // end try
    catch (int e) {
        cout << "Explicitly throw an exception" << endl;
    } // end catch

    cout << "End of program" << endl;
}
```



Enter a and b: 3 0

Explicitly throw an exception

End of program

C++ Exceptions (Another example)

We can define our own exception class:

```
/* -----
```

Header file: `notHappyException.h`

```
#include <string> ----- */
```

```
using namespace std;
```

```
class notHappyException {  
public:  
    notHappyException () : message ("Very sad leh") { }
```

```
    string what () const { return message; }
```

```
private:
```

```
    string message;  
};
```

```
#include "notHappyException.h"  
#include <iostream>
```

```
int main () {  
    bool failedCS1102 = true;
```

```
    try {  
        if (failedCS1102)  
            throw notHappyException ();
```

```
    else  
        cout << "Happy" << endl;  
    }
```

```
    catch (notHappyException ex) {  
        cout << "Exception occurred: " << ex.what () << endl;  
    }  
}
```

Exception occurred: Very sad leh