

...

Namespaces

- **What is a namespace?**
  - Mechanism for expressing logical grouping
- **Why do we need them?**
- **How should they be used?**
  - The simple way
    - using namespace std;
  - More sophisticated use

.....

---

---

---

---

---

---

---

---

...

What's the Problem?

```
// alib.h - Ada.
...
#include <iostream>
...
void f(ostream& os);
...
```

```
// blib.h - Bjarne.
...
#include <iostream>
...
void f(ostream& os);
...
```

```
// namspc.cpp
...
#include "alib.h"
#include "blib.h"

void main ()
{
  ...
  f(cout);
}
```

.....

---

---

---

---

---

---

---

---

...

Namespace Solution

- **Define separate namespaces**
  - "Ada" and "Bjarne"
- **Define separate libraries**
  - Wrapped in corresponding namespaces
- **Specify desired elements**
  - Using qualified names

.....

---

---

---

---

---

---

---

---

...

Library Declaration

```

// alib.h - "Ada" library.
#ifndef ALIB_H
#define ALIB_H

#include <iostream>

namespace Ada
{
    void f(std::ostream& os);
}
#endif // ALIB_H

```

.....

---

---

---

---

---

---

---

---

...

Library Definition

```

// alib.cpp - "Ada" library.
#include <iostream>

#include "alib.h"

using std::ostream;
using std::endl;

void Ada::f(ostream& os)
{
    os << "Ada:f" << endl;
}

```

Using declarations →

Qualified name →

.....

---

---

---

---

---

---

---

---

...

Using Directives/Declarations

- **Using directive**
  - Makes all names in namespace accessible in the current scope
    - using namespace std;
- **Using declaration**
  - Declares synonym for element in current scope
    - using std::ofstream;

.....

---

---

---

---

---

---

---

---

...

Namespace Advice

- **Create separate namespaces**
- **Global using directive (e.g., std)**
  - A little bit sloppy
  - Transitional/introductory technique
- **Qualified names**
  - Unambiguous, a little verbose(?)
- **Using declarations**
  - Use sparingly, in local scopes(?)

---

---

---

---

---

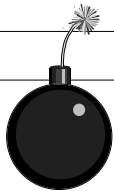
---

---

---

...

Exception Handling



- **Exceptions can't happen!?**
  - Don't worry, be happy
- **Laissez-faire**
  - Ignore; break someone else's code!
- **Exceptions are normal?**
  - Exception testing everywhere?
- **Exceptions are exceptional?**
  - Optimize normal; cope with exceptions

---

---

---

---

---

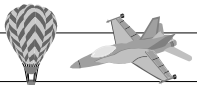
---

---

---

...

Bailing Out



- **If exceptions are exceptional**
  - Don't occur in "normal" operation
  - No recovery at point of exception
- **Stop what we're doing**
  - Clean up (destructors) & give up
- **Ask for higher-level help**

---

---

---

---

---

---

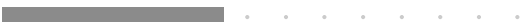
---

---

...

C++ Exceptions

- **At point of exception**
  - Create object representing exception
    - Usually of an exception class
- **Throw exception**
  - Leave current context (local destructors)
  - Passing exception object to ... ???



...

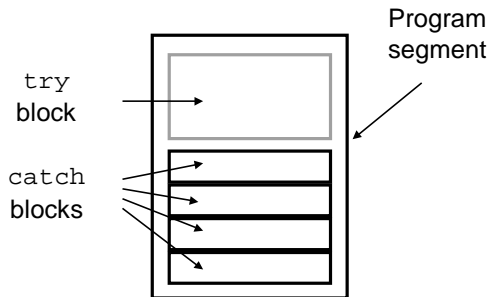
Exception Handler

- **Defined area of responsibility**
  - Section of code
    - Including any called routines
  - Unless lower-level handler intervenes
- **Catches thrown exceptions**
  - By type of exception object



...

Exception Structure



Handwritten notes area with horizontal lines.

Handwritten notes area with horizontal lines.

Handwritten notes area with horizontal lines.

⋮

Exception Example (1)

```
try
{...
  int x = divide (a,b);
}
catch (exception& ex)
{
  cout << ex.what() << endl;
}
```

.....

---

---

---

---

---

---

---

---

⋮

Exception Example (2)

```
int divide (int num, int div)
{
  if (num < 0)
    throw invalid_argument("neg div");

  if (div == 0)
    throw domain_error("div zero!!");

  if (num == div)
    throw string ("Hello");

  return (num/div);
}
```

.....

---

---

---

---

---

---

---

---

⋮

Catch Options

Catch blocks are tried in order until a type match (including derived) is found.

```
try { . . . }
catch (invalid_argument& ex)
{
  cout<<"Invalid arg: "<<ex.what()<<endl;
}
catch (...)
{
  cout<<"Other exception"<<endl;
}
```

.....

---

---

---

---

---

---

---

---

⋮

Exception Specs

- **Part of function declaration**
  - void f() throw (domain\_error);
  - void g() throw(); // none
  - void h(); // any
- **Specifies possible exceptions**
  - Others, if any, are “unexpected”

.....

⋮

Standard Exception Classes

```
exception
  logic_error
    domain_error    invalid_argument
    length_error    out_of_range
  runtime_error
    range_error     overflow_error
    underflow_error
  bad_alloc
  bad_exception
  ios_base::failure
  bad_cast
  bad_typeid
```

.....

⋮

Cautionary Note



- **Exception handling is complex**
  - Have only touched on the basics
- **Some other issues**
  - Exceptions from constructors
  - Exceptions from destructors
  - Dynamic objects tied to pointers
    - Are they deallocated?
  - Exception catching details

.....

Horizontal lines for notes.

Horizontal lines for notes.

Horizontal lines for notes.

```
// namspc.cpp - Demonstration program for namespaces.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#include <iostream>
#include <fstream>

#include "alib.h"    // "Ada" library.
#include "blib.h"    // "Bjarne" library

// Function prototype.
void f(std::ostream& os);

void main ()
{
    std::ofstream outfile("namspc.txt");

    // Call the function defined in this module.
    f(outfile);

    // Explicitly call the library functions.
    Ada::f(outfile);
    Bjarne::f(outfile);

    // Define a synonym for one of the namespaces.
    namespace B = Bjarne;

    // Use the synonym to specify a function.
    B::f(outfile);

    // Use "using" directive to make a synonym for the function name
    // in the current scope.
    using Ada::f;

    // Invoke the function using the synonym.
    f(outfile);

    {
        // Use "using" directive to make a synonym for the function
        // name in a nested scope.
        using Bjarne::f;

        // Invoke the function using the synonym.
        f(outfile);

        // Invoke the global namespace function.
        ::f(outfile);
    }

    // Invoke the function using the synonym (back in the outer
    // scope).
    f(outfile);
}

void f(std::ostream& os)
{
    os << "<Global>::f" << std::endl;
}
```

```

// alib.h - Declaration of "Ada" library.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#ifndef ALIB_H
#define ALIB_H

#include <iostream>

namespace Ada
{
    // Declare a function in the namespace.
    // (Could also have classes or other things.)
    void f(std::ostream& os);
}

#endif // ALIB_H
=====
// blib.h - Declaration of "Bjarne" library.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#ifndef BLIB_H
#define BLIB_H

#include <iostream>

namespace Bjarne
{
    // Declare a function in the namespace.
    // (Could also have classes or other things.)
    void f(std::ostream& os);
}

#endif // BLIB_H
=====
// alib.cpp - Definition of "Ada" library.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#include <iostream>

#include "alib.h"

using std::ostream;
using std::endl;

void Ada::f(ostream& os)
{
    os << "Ada::f" << endl;
}

=====
// blib.cpp - Definition of "Bjarne" library.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#include <iostream>

#include "blib.h"

void Bjarne::f(std::ostream& os)
{
    os << "Bjarne::f" << std::endl;
}

```

<Global>::f  
Ada::f  
Bjarne::f  
Bjarne::f  
Ada::f  
Bjarne::f  
<Global>::f  
Ada::f

```
// except.cpp - Demo program for exception handling.
// Mark Sebern, MSOE
// Version 1.0
// 9 May 1999

#include <iostream>
#include <string>
#include <exception>
using namespace std;

// Function prototype:
int divide (int num, int div);

void main ()
{
    // Execute code where we want to handle exceptions.
    try
    {
        int a = 0;
        int b = 0;
        cout << "Enter a, b: ";
        cin >> a >> b;

        int x = divide (a,b);
        cout << "x = " << x << endl;
    }
    // Catch invalid argument errors.
    catch (invalid_argument& ex)
    {
        cout << "Invalid arg: " << ex.what() << endl;
    }
    // Catch any other errors derived from "exception".
    catch (exception& ex)
    {
        cout << "Exception: " << ex.what() << endl;
    }
    // Catch any kind of error we haven't handled.
    catch (...)
    {
        cout << "Other exception" << endl;
    }

    cout << "Done with try/catch block." << endl;
}

// Note: no explicit "throw" declaration on this function, so
// it can throw anything.

int divide (int num, int div)
{
    if (num < 0) throw invalid_argument("negative dividend not allowed");

    if (div == 0) throw domain_error("divide by zero!!");

    // Try throwing class not derived from exception.
    if (num == div) throw string ("Hello");

    return (num/div);
}
```