C++ Namespaces

Problem:
As programs grow in size and complexity, it becomes more difficult to keep the "names" used in different modules from overlapping and conflicting.

→ many data structures have an "insert" operation

A C++ namespace is a mechanism that allows a collection of related variables and functions to be grouped together in a unique declarative region. Then the scope resolution operator :: is used to refer to the items in a region.

→ a namespace declaration region is similar to a "class" declaration, except that it simply is a grouping of related items.

Example:

```
namespace  stringPkgNamespace
{
     int  stringLength(const char s[]);
     void stringCopy(char dest[], const char src[]);
     void stringConcatenate(char dest[], const char t[]);
     char stringGetchar(const char s[], const int position);
     int  stringFindchar(const char s[], const char ch);
     void stringSubstring(
          char resultString[],char s[],
          const int start, const int len = -1);
     int  stringCompare(const char s[], const char t[]);
}

// Implement - use :: scope resolution operator
int stringPkgNamespace::stringLength(const char s[])
{
     int len = 0;
     while(s[len] != 0)  ++len;

     return(len);
}

. . . // etc etc etc
```

```
// Reference using scope resolution :: operator

int main( )
{
     char buf[50] = “Hi There”;
     char dest[50];
     int  len;

     len = stringPkgNamespace::stringLength(buf);

     stringPkgNamespace::stringCopy(dest, buf);
  . . .
}
```

```
// USING declaration to import a specific name directly
// into the current (default program) namespace

using namespace stringPkgNamespace::stringLength;

int main( )
{
     char buf[50] = “Hi There”;
     char dest[50];
     int  len;

     len = stringLength(buf);

     stringPkgNamespace::stringCopy(dest, buf);
  . . .
}
```

```
// USING declaration to import all names directly
// into the current (default program) namespace

using namespace stringPkgNamespace;

int main( )
{
     char buf[50] = “Hi There”;
     int  len;

     len = stringLength(buf);

     stringCopy(dest, buf);
  . . .
}
```

C++ Standard Libraries

Current versions of C++ support both an old-style collection of library functions and a newer namespace-oriented collection of similar library functions.

Older style

```
#include <iostream.h>
#include <string.h>

     int numHits;
     cin >> numHits;
     cout << "Number of hits: " << numHits << endl;

     char buf[50] = "Hi There";
     char dest[50];
     int  len;

     len = strlen(buf);
     strcpy(dest, buf);
```

Newer **std** namespace style: <iostream>

<cstring> C-strings equivalency
Not part of **std** namespace

```
#include <iostream>
#include <cstring>

     int numHits;
     std::cin >> numHits;
     std::cout << "Number of hits: " << numHits << std::endl;

     char buf[50] = "Hi There";
     char dest[50];
     int  len;

     len = strlen(buf);       // cstring imports are NOT part
     strcpy(dest, buf);       //   of the std::  namespace
```

using namespace directive:
imports **all prototypes** from the **std** namespace into the current (default program) namespace.

```
#include <iostream>
#include <cstring>

using namespace std;

     int numHits;
     cin >> numHits;
     cout << "Number of hits: " << numHits << endl;

     char buf[50] = "Hi There";
     char dest[50];
     int  len;

     len = strlen(buf);
     strcpy(dest, buf);
```

```
#include <iostream>
#include <cstring>

using namespace std::cin;
using namespace std::cout;
using namespace std::endl;

     int numHits;
     cin >> numHits;
     cout << "Number of hits: " << numHits << endl;

     char buf[50] = "Hi There";
     char dest[50];
     int  len;

     len = strlen(buf);
     strcpy(dest, buf);
```

using namespace directives:
imports **specific prototypes** from the **std** namespace into the current (default program) namespace.

STL   Standard Template Library

```
<cassert>      Equivalent to older C-style .h functionality
<cmath>
<cstdlib>           Included names are NOT part of std:: namespace
<cstring>
<ctime>

<iostream>     Equivalent to older C-style .h functionality
<istream>      except
<ostream>           Included names ARE part of std:: namespace
<fstream>
<strstream>

<string>       =, size, compare ops, + concatenate, <<, >>,
                    getline(), c_str()
<list>         =, empty, size, front, back, push_front, pop_front,
                    push_back, pop_back, insert, erase, remove,
                    sort, merge, swap, reverse, iterate
<stack>        =, size, empty, push, pop, top&
<queue>        =, size, empty, push, pop, front&, back&
<deque>        =, size, empty, push_front, pop_front, push_back,
                    pop_back, insert, erase
<vector>       =, size, empty, insert, erase, at, [ ]

<exception>    Exception Handlers
<stdexcept>
```

C++ Exception Handling
Throwing Exceptions, Try – Catch Blocks

Problem:
Most "code segments" in a program could potentially fail to execute properly if certain data values or other exceptional circumstances were to occur. Some of these situations are the result of illegal or unanticipated data values. Some are the result of exceeding the storage capacity of a data structure. Some are caused by program coding errors.

- → runtime computation errors, e.g., divide by zero
- → incorrect input data, e.g., reading an 'int', but receiving a string value
- → attempt to insert a new value into a "full" list
- → attempt to access a position in a list that does not exist

Some of these situations can be effectively handled by coding protocols. For example, a list "insert" method can return a "success" status code, indicating whether or not the operation completed properly.

However, some situations cannot be easily handled with simple coding protocols. For example, handling user-data input errors is usually complex and difficult.

Additionally, some situations will rarely actually occur during execution (for example, a computational error, or overflowing the capacity of a data structure).

C++ Exception Throwing and Catching:

C++ provides an "exception handling" mechanism that allows a systematic and straightforward coding style to deal with runtime errors that might occur in a program. The general form of a "try – throw – catch" block is:

```
try
{
   statement-list;       // where some stmts might throw
                         // an exception

           → throw  ExceptionClass(stringArgument);
}
catch (ExceptionClass identifier)
{
   statements;      // code to handle the exception
                    //    if it us thrown
}
```

Try-Catch example

```
#include <iostream.h>
#include <exception>

int main()
{
     int a, b, c;

     try
     {
          cout << “Enter 3 sides of a triangle: “;
          cin >> a >> b >> c;

          if( (a <= 0) || (b <= 0) || (c <= 0) )
               throw  0;                              throws  int  exception

          if( (a+b<=c) || (a+c<=b) || (b+c<=a) )
               throw “Sides do not form triangle”;    throws  char [ ]  exception

     }
     catch ( int e )           catches  int  exception
     {
          cout << “Exception raised: Illegal side length“ << endl;
          cout << “Defaulting to (3, 4, 5)” << endl;
          a = 3;  b = 4;  c = 5;
     }
     catch ( char msg[ ] )     catches  char [ ]  exception
     {
          cout << “Exception raised: “ << msg << endl;
          exit(1);       // Exit the program
     }
     catch ( exception e )     catches any  other  exception,   e.g., cin I/O exception
     {
          cout << “Base exception raised: “ << e.what() << endl;
          exit(1);       // Exit the program
     }

     // Rest of program continues, with valid triangle sides
     //. . . .
}
```

- If the Try block does not detect an exception, all Catch clauses are skipped, and execution just flows to the statements following the try-catch construct.
- Code in this Try block may throw either an int exception or a char[ ] exception, which will be caught and handled in the corresponding Catch clause.
- If an exception is thrown at runtime, C++ matches the data type of the “throw” with the data type of the “catch” parameter, and enters the corresponding catch clause.
- When a catch clause is finished, the other catch clauses are skipped and execution just flows to the statements following the try-catch construct (unless the program exits( ) in the catch clause).