```
//////////////////////////////////////////////////////////////////
"class" examples:

Name:                 "wrap" up an ASCIIZ char[ ] buffer in a class.

CheckingAccount:     Use Name as a member of CkAcct object.

//////////////////////////////////////////////////////////////////
```

Occasionally, a header file gets "#included" more than once within a source code file's compilation unit (for example, if a project contains multiple interrelated .h header include files)
- causes "duplicate declaration" errors

Code header files so that this error never arises in practice.
Specify conditional-compilation #ifndef specifications that permit a section of code to be processed the first time it is encountered, but ignored if encountered again within the same compilation unit.

.h header file

```
//////////////////////////////////////////////////////////////////
// names.h
// Type for representing person's name with 49 characters or less.

#ifndef  _NAMES_H

#define  _NAMES_H

class Name {
public:
   Name(const char n[] = "");   // Creates a name from n, using at
                                //   most the first 50 characters
                                //   from n, up to and including
                                //   the terminating '\0'.

   void copyToString(char target[], const int max);
                                // Copies the characters of the
                                //   name to target.

   void print() const;          // Prints the name to cout.

private:
   enum { MAXNAME_ = 50 };   // A totally-local named-constant
                             // Optional technique for local const names

   char name_[ MAXNAME_ ];   // Represent name_ as a cstring value
};

#endif
```

Simple convention to create unique symbol string that will not be defined anywhere else within a project:
Use the header filename, all capital letters, replace the . dot with _, and prefix with _ underscore.

End of the #ifndef section

.cpp implementation file

```
/////////////////////////////////////////////////////////////////////
// names.cpp  Implementation file

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

#include "names.h"


Name :: Name(const char n[])       // constructor
{
   // Copy cstring value n into name_ member variable
   strncpy_s(name_, MAXNAME_, n, MAXNAME_ - 1 /* or _TRUNCATE */ );
}

void Name :: copyToString(char target[], const int max)
{
   strncpy_s(target, max, name_, _TRUNCATE); // always appends NULL
                                             //    at end of target
}

void Name :: print() const
{
   cout << name_;
}
```

Include whatever header files are needed for the class implementation

Always include the corresponding .h header

```
// Some examples of Name usage

Name  patrick(“Pat”);

Name  myfriend(“Jenny”);

Name  anonymous;

anonymous = myfriend;

anonymous = Name(“Fred”);


///////////////////////////////////////////////////
// Try to output the Name “name_” cstring...

cout << anonymous;    // Error - cout cannot handle it.
                      // Does not have built-in conversion
                      //  for user-defined class datatypes

cout << anonymous.name_;  // Error - Unable to reference
                          //   private data member


///////////////////////////////////////////////////
// Instead, extract out a local copy of the cstring value.
char  who[100];

anonymous.copyToString(who, 100);

cout << "anonymous is " << who << endl; // works OK


cout << “patrick’s name is “;
patrick.print();                       // works OK
cout << endl;
```

```
/////////////////////////////////////////////////////////////////
// CkAcct.h -  CheckingAccount Class

#ifndef _CKACCT_H
#define _CKACCT_H

#include "names.h"

// A simple checking account class
class CheckingAccount {
public:

   // Create checking account with owner named n and
   //    beginning balance b.
   // ASSUME: length of n < 50; b >= 0.
   //
   // Basic "convert" constructor, with default parameter values
   CheckingAccount(const char n[] = "", const float b = 0.0);

   // Constructor with existing "Name" object
   //   Almost always pass objects "by reference"
   CheckingAccount(const Name &n, const float b = 0.0);

                                    // "getter" methods
   float theBalance() const;        //   The account balance
   Name theOwner() const;           //   Name of the account owner

   void deposit(const float amt);   // Credits balance with amt.
                                    // ASSUME: amt >= 0.

   void writeCheck(const float amt); // If current bal. is >= amt,
                                    //   amt is debited from
                                    //   the balance,
                                    // else nothing is changed.
                                    // ASSUME: amt >= 0.
private:
   float balance_;
   Name  owner_;         // Embedded Name object
};
#endif
```

```
/////////////////////////////////////////////////////////
// Some examples of Checking Account usage
#include "names.h"
#include "CkAcct.h"

// ----------------------------------------------------
CheckingAccount   mike("Mikey");

mike.deposit(50.00);
mike.deposit(100);
mike.writeCheck(500);


// ----------------------------------------------------
CheckingAccount   herAcct("Sally", 1000);

if(herAcct.theBalance()  >= 75.00)
   herAcct.writeCheck(75.00);


cout << setiosflags(ios::fixed) << setprecision(2);
cout << "Account of ";

herAcct.theOwner().print();

cout << " has a balance of $"
     << herAcct.theBalance()
     << endl;

// ----------------------------------------------------

Name  patrick("Pat");

CheckingAccount   PatsAcct(patrick, 250);

// Print account balance
PatsAcct.theOwner().print();
//  or ...
//  patrick.print();

cout << " has a balance of $" << PatsAcct.theBalance()
     << endl;
```

```
////////////////////////////////////////////////////////////////////
// ckAcct.cpp   Implementation File

#include "ckAcct.h"

Name CheckingAccount :: theOwner() const
{
   return owner_;
}

float CheckingAccount :: theBalance() const  { return balance_; }

void CheckingAccount :: deposit(const float amt)
{
   balance_ += amt;
}

void CheckingAccount :: writeCheck(const float amt)
{
   if (balance_ >= amt)
      balance_ -= amt;
}
```

Constructors, using "header initialization" syntax:

```
// Constructor: header initialization form
CheckingAccount :: CheckingAccount(const char n[], const float b)
     :  balance_(b), owner_(n)
{ }

CheckingAccount :: CheckingAccount(const Name &n, const float b)
     :  balance_(b), owner_(n)
{ }
```

Alternative, using normal method implementation:

```
// Constructor: Body definition form
CheckingAccount :: CheckingAccount(const char n[], const float b)
{
   owner_   = Name(n);
   balance_ = b;
}

CheckingAccount :: CheckingAccount(const Name &n, const float b)
{
   owner_   = n;
   balance_ = b;
}
```