Ugly Realities

The Dark Side of C++
Chapter 12

Ugly Realities

- In C++ we can pass objects to functions, use an object to initialize another object, and set an object equal to another object of the same type (assignment).

- All of these actions involve *copying* an object.
  - This can get ugly when an object includes pointers as member variables.

Example

- We will look at the StringBad example at the beginning of Chapter 12.
  - Listing 12.1, 12.2, 12.3

- Example of a simple string class that has some bad properties.
  - Shows the pitfalls of making copies of an object that includes a pointer as a member variable.
  - And how you might make a copy without knowing it.

Example

- Create a new C++ console project.
- Download from the class web site:
- http://www.csee.usf.edu/~turnerr/Object_Oriented_Design/Downloads/Chapter_12/
- Three files:
  - strngbad.h
  - strngbad.cpp
  - vegnews.cpp (Main Program)
- Add files to the project.

strngbad.h -- flawed string class definition

```
#include <iostream>
#ifndef STRNGBAD_H_
#define STRNGBAD_H_

class StringBad
{
private:
    char * str;                  // pointer to string
    int len;                     // length of string
    static int num_strings;      // number of objects
public:
    StringBad(const char * s); // constructor
    StringBad();                 // default constructor
    ~StringBad();                // destructor
// friend function
   friend std::ostream & operator<<(std::ostream & os,
                    const StringBad & st);
};
#endif
```

strngbad.cpp -- StringBad class methods

```
#include "strngbad.h"
using std::cout;

// initializing static class member
int StringBad::num_strings = 0;

// class methods

// construct StringBad from C string
StringBad::StringBad(const char * s)
{
    len = std::strlen(s);            // set size
    str = new char[len + 1];         // allocate storage
    std::strcpy(str, s);             // initialize pointer
    num_strings++;                   // set object count

    cout << num_strings << ": \"" << str
         << "\" object created\n";  // FYI
}
```

Static variable must be initialized in the .cpp file

Default constructor

```
StringBad::StringBad()
{
    len = 4;
    str = new char[4];
    std::strcpy(str, "C++");                    // default string
    num_strings++;
    cout << num_strings << ": \"" << str
         << "\" default object created\n";  // FYI
}
```

Destructor

```
StringBad::~StringBad()
{
    cout << "\"" << str << "\" object deleted, ";    // FYI
    --num_strings;                              // required
    cout << num_strings << " left\n"; // FYI
    delete [] str;                              // required
}
```

Note []

Required because str is an array

We *must* define a real destructor for any class that has dynamically allocated member variables. Otherwise there will be a memory leak.

Friend operator<<

```
std::ostream & operator<<(std::ostream & os, const StringBad & st)
{
    os << st.str;
    return os;
}
```

Main Program

```
// vegnews.cpp -- using new and delete with classes
// compile with strngbad.cpp
#include <iostream>
using std::cout;

#include "strngbad.h"

void callme1(StringBad &);  // pass by reference
void callme2(StringBad);    // pass by value

int main()
{
    using std::endl;
    StringBad headline1("Celery Stalks at Midnight");
    StringBad headline2("Lettuce Prey");
    StringBad sports("Spinach Leaves Bowl for Dollars");
    cout << "headline1: " << headline1 << endl;
    cout << "headline2: " << headline2 << endl;
    cout << "sports: " << sports << endl;
```

Main Program (continued)

```
    callme1(headline1);
    cout << "headline1: " << headline1 << endl;
    callme2(headline2);
    cout << "headline2: " << headline2 << endl;
    cout << "Initialize one object to another:\n";
    StringBad sailor = sports;
    cout << "sailor: " << sailor << endl;
    cout << "Assign one object to another:\n";
    StringBad knot;
    knot = headline1;
    cout << "knot: " << knot << endl;
    cout << "End of main()\n";

    std::cin.get();    // Add this

    return 0;
}
```

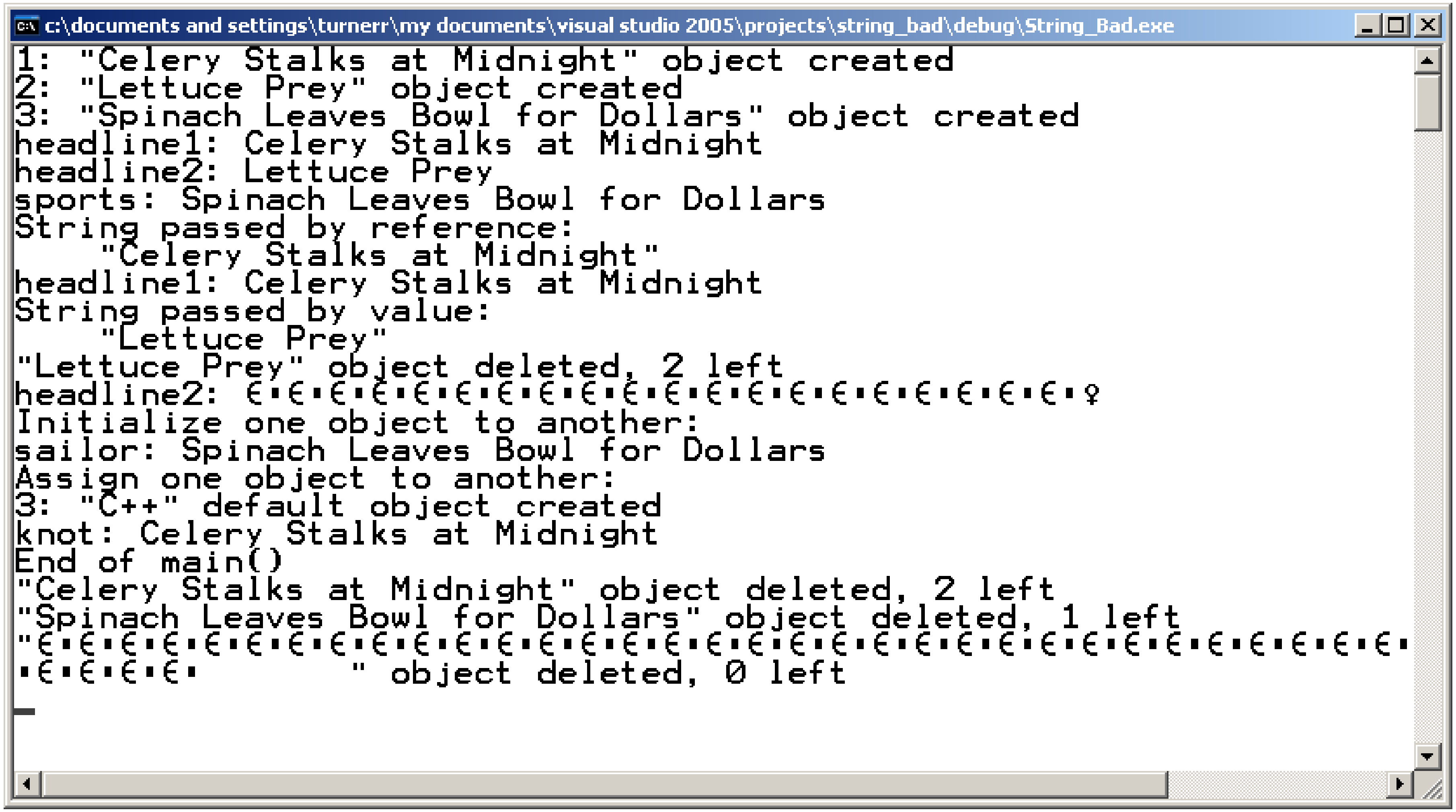
Functions callme1 and callme2

```
void callme1(StringBad & rsb)
{
    cout << "String passed by reference:\n";
    cout << "    \"" << rsb << "\"\n";
}

void callme2(StringBad sb)
{
    cout << "String passed by value:\n";
    cout << "    \"" << sb << "\"\n";
}
```

- Build and run.

Program Output



```
c:\documents and settings\turnerr\my documents\visual studio 2005\projects\string_bad\debug\String_Bad.exe
1: "Celery Stalks at Midnight" object created
2: "Lettuce Prey" object created
3: "Spinach Leaves Bowl for Dollars" object created
headline1: Celery Stalks at Midnight
headline2: Lettuce Prey
sports: Spinach Leaves Bowl for Dollars
String passed by reference:
    "Celery Stalks at Midnight"
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 2 left
headline2: €▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪♀
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
Assign one object to another:
3: "C++" default object created
knot: Celery Stalks at Midnight
End of main()
"Celery Stalks at Midnight" object deleted, 2 left
"Spinach Leaves Bowl for Dollars" object deleted, 1 left
"€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪€▪
▪€▪€▪€▪€▪       " object deleted, 0 left
```

What's Happening Here?

- Set breakpoint and step line by line.

- OK at first.
  - Declare 3 StringBad objects
  - Initialization results in call to constructor
    - Constructor uses new char[len + 1] to allocate space for the string.
    - Copies C string passed by the caller into the allocated space
    - Increments string count.
    - Outputs message

  - Output these three strings.

callme1()

- Pass StringBad object headline1 to callme1 by reference.
- It outputs the string and returns.
- After return the StringBad object (headline1) is still OK.

callme2()

- Pass StringBad object headline2 to callme2 by *value*.
- It outputs the string and returns.
- After return from callme2 the object headline2 is garbage!
- Prior to that we have an output message from the StringBad destructor.

What happened?

- There's more going on here than meets the eye!

- On call to callme2(), compiler makes a *copy* of the StringBad object passed as the argument.
- What constructor is invoked?
  - A default *copy constructor*.

Copy Constructor

- A *copy constructor* is used whenever we initialize an object from an existing object of the same type.

- Where motto is an existing StringBad object:
  - `StringBad ditto(motto);`
  - `StringBad metoo = motto;`
  - `StringBad also = StringBad(motto);`

  -
    ```
    StringBad * pStringBad =
                   new StringBad(motto);
    ```

Copy Constructors

- Call by value always passes a *copy* of the caller's argument to the function.

- A copy constructor is used to instantiate a copy the StringBad argument when callme2() is called.
  - The object seen by the function code as the function parameter is a *copy* of the object used as the argument of the call.

The Default Copy Constructor

- If we don't write a copy constructor the compiler provides one.
  - A *default* copy constructor.
  - Member by member copy
  - *Shallow* copy.
  - Pointers are replicated.
    - New object's pointers point to same thing as original object's pointers.

callme2 returns

- When callme2 returns, its local copy of the StringBad object is automatically deleted.

- The StringBad destructor is called.
- What happens?

StringBad Destructor Called

- What happens?
  - delete [] str;
- The str member in the copy points to the same char array as the original.
  - Due to the default copy constructor making a *shallow copy*.
- *The destructor for the copy deletes the char array used by the original!*

The Cure

- Define a copy constructor for the class.
- In the copy constructor, reproduce all objects pointed to by the object being copied.
  - Allocate space for the string data.
  - Copy the original data to the allocated space.
- This is called a *deep copy*.

A Deep Copy

- Add to StringBad.h:

```
StringBad(const StringBad &); // copy constructor
```

- Add to StringBad.cpp:

```
StringBad::StringBad(const StringBad & st)
{
    num_strings++;  // handle static member update
    len = st.len;                  // same length
    str = new char [len + 1];  // allot space
    std::strcpy(str, st.str);  // copy string
}
```

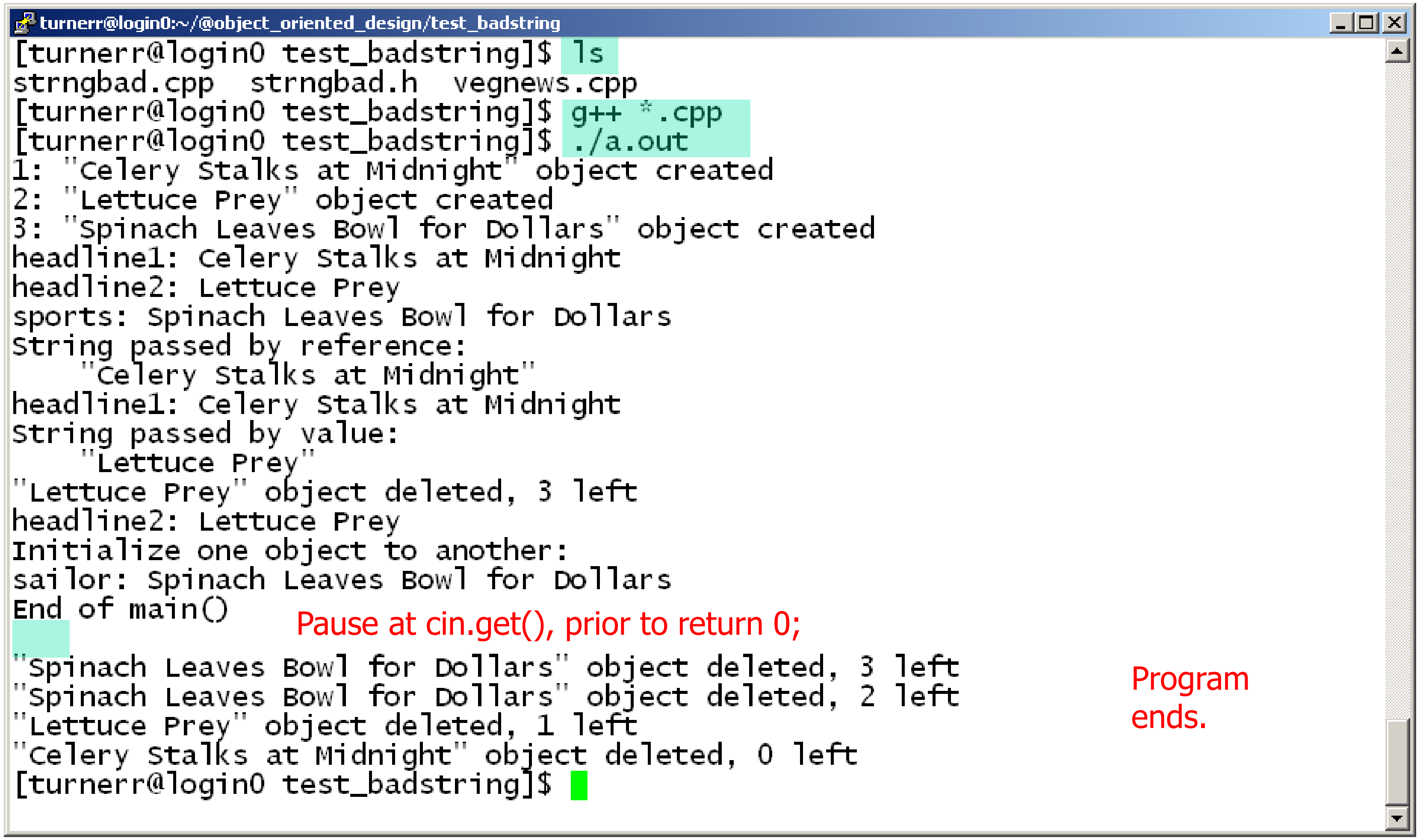
Try Again

- Program gets past callme2
  - Still has problems at the end.

- Comment out everything below

```
cout << "sailor: " << sailor << endl;
```

Running on Circe



```
[turnerr@login0 test_badstring]$ ls
strngbad.cpp   strngbad.h   vegnews.cpp
[turnerr@login0 test_badstring]$ g++ *.cpp
[turnerr@login0 test_badstring]$ ./a.out
1: "Celery Stalks at Midnight" object created
2: "Lettuce Prey" object created
3: "Spinach Leaves Bowl for Dollars" object created
headline1: Celery Stalks at Midnight
headline2: Lettuce Prey
sports: Spinach Leaves Bowl for Dollars
String passed by reference:
    "Celery Stalks at Midnight"
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 3 left
headline2: Lettuce Prey
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
End of main()

"Spinach Leaves Bowl for Dollars" object deleted, 3 left
"Spinach Leaves Bowl for Dollars" object deleted, 2 left
"Lettuce Prey" object deleted, 1 left
"Celery Stalks at Midnight" object deleted, 0 left
[turnerr@login0 test_badstring]$
```

Pause at cin.get(), prior to return 0;

Program ends.

End of Section

Assignment Operators

- Whenever we write an assignment statement for an object

```
        mystring2 = mystring2;
```

  the compiler produces code to invoke an *asignment operator* for that class.

- If we haven't defined an assignment operator, the compiler creates a default version.
  - Like the default copy constructor, it does a shallow copy.

The Problem with Assignment

```
StringBad knot;
knot = headline1;
```

The str member of StringBad object knot points to the same memory block as headline1.

When main ends, each local object is destroyed

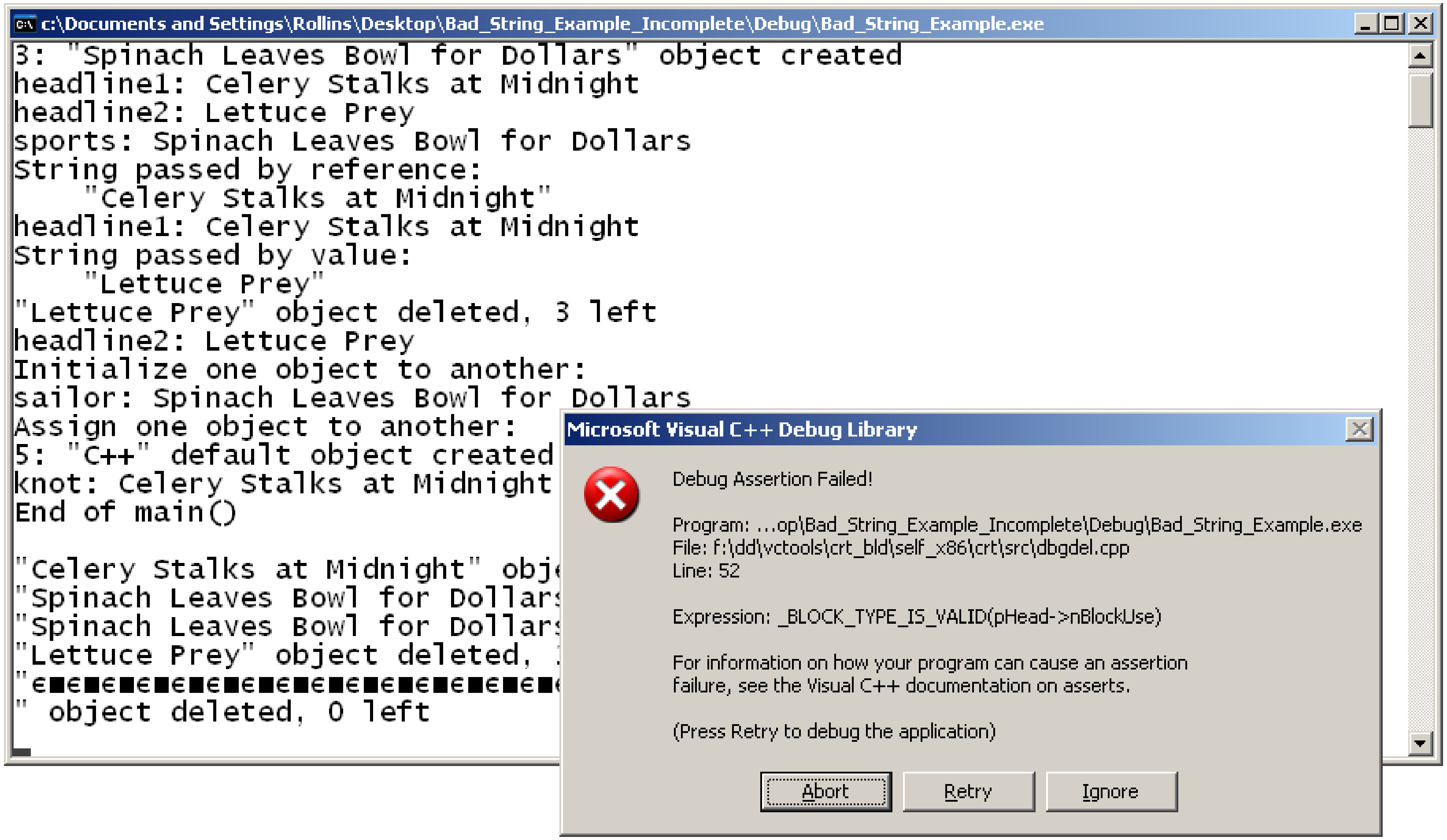
Destructor is called for each in reverse order.

Destructor for knot deallocates the space for the string.

Destructor for headline1 tries to do the same.

The results are *unpredictable*.

Running on Windows



c:\Documents and Settings\Rollins\Desktop\Bad_String_Example_Incomplete\Debug\Bad_String_Example.exe

```
3: "Spinach Leaves Bowl for Dollars" object created
headline1: Celery Stalks at Midnight
headline2: Lettuce Prey
sports: Spinach Leaves Bowl for Dollars
String passed by reference:
    "Celery Stalks at Midnight"
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 3 left
headline2: Lettuce Prey
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
Assign one object to another:
5: "C++" default object created
knot: Celery Stalks at Midnight
End of main()

"Celery Stalks at Midnight" obj
"Spinach Leaves Bowl for Dollars
"Spinach Leaves Bowl for Dollars
"Lettuce Prey" object deleted,
"ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■ε■
" object deleted, 0 left
```

Microsoft Visual C++ Debug Library

Debug Assertion Failed!

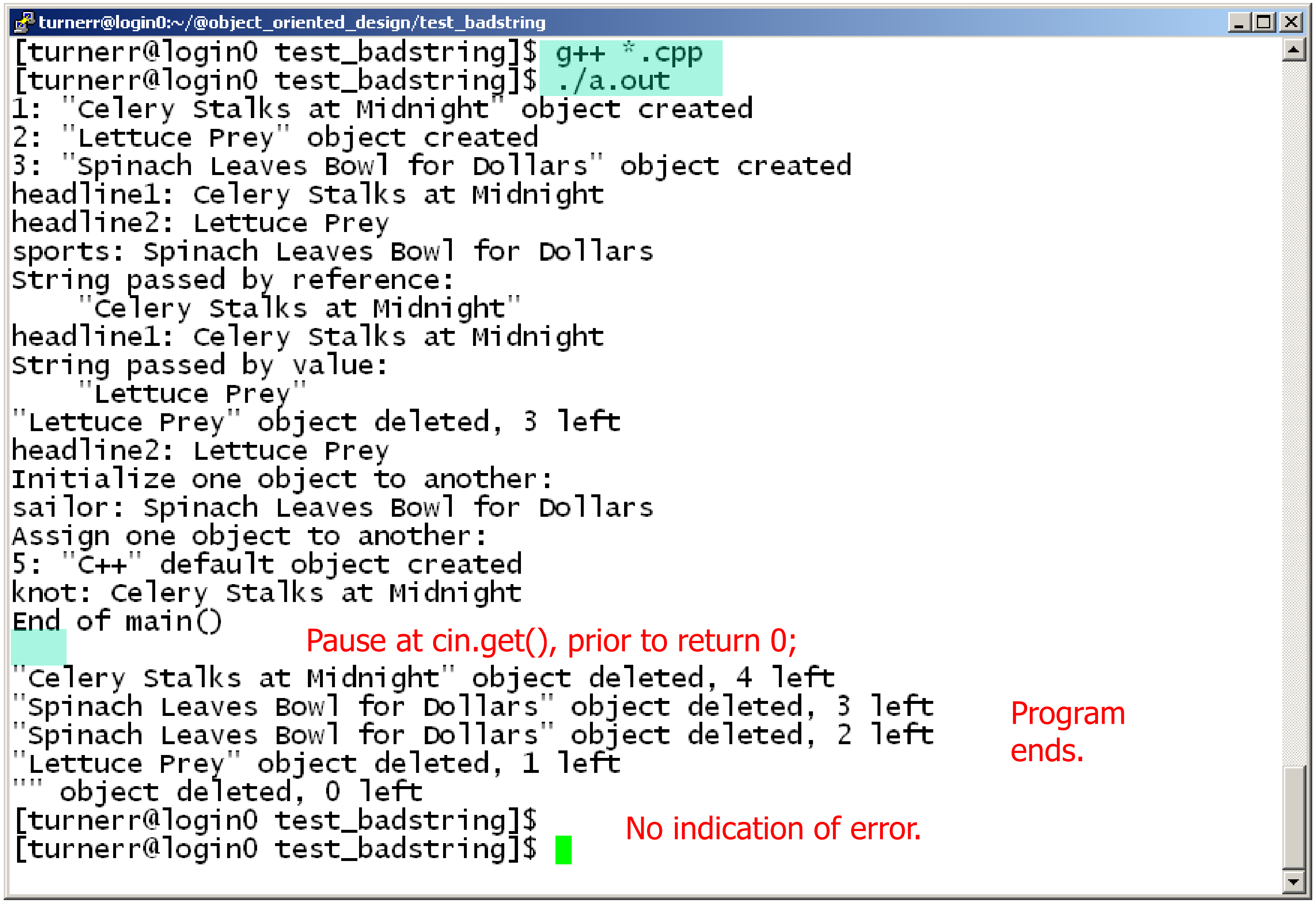
Program: ...op\Bad_String_Example_Incomplete\Debug\Bad_String_Example.exe
File: f:\dd\vctools\crt_bld\self_x86\crt\src\dbgdel.cpp
Line: 52

Expression: _BLOCK_TYPE_IS_VALID(pHead->nBlockUse)

For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.

(Press Retry to debug the application)

Running on Circe



```
[turnerr@login0 test_badstring]$ g++ *.cpp
[turnerr@login0 test_badstring]$ ./a.out
1: "Celery Stalks at Midnight" object created
2: "Lettuce Prey" object created
3: "Spinach Leaves Bowl for Dollars" object created
headline1: Celery Stalks at Midnight
headline2: Lettuce Prey
sports: Spinach Leaves Bowl for Dollars
String passed by reference:
    "Celery Stalks at Midnight"
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 3 left
headline2: Lettuce Prey
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
Assign one object to another:
5: "C++" default object created
knot: Celery Stalks at Midnight
End of main()

"Celery Stalks at Midnight" object deleted, 4 left
"Spinach Leaves Bowl for Dollars" object deleted, 3 left
"Spinach Leaves Bowl for Dollars" object deleted, 2 left
"Lettuce Prey" object deleted, 1 left
"" object deleted, 0 left
[turnerr@login0 test_badstring]$
[turnerr@login0 test_badstring]$
```

Pause at cin.get(), prior to return 0;

Program ends.

No indication of error.

The Cure

- We must provide an assignment operator that does a deep copy.

- Add to class definition in strngbad.h:

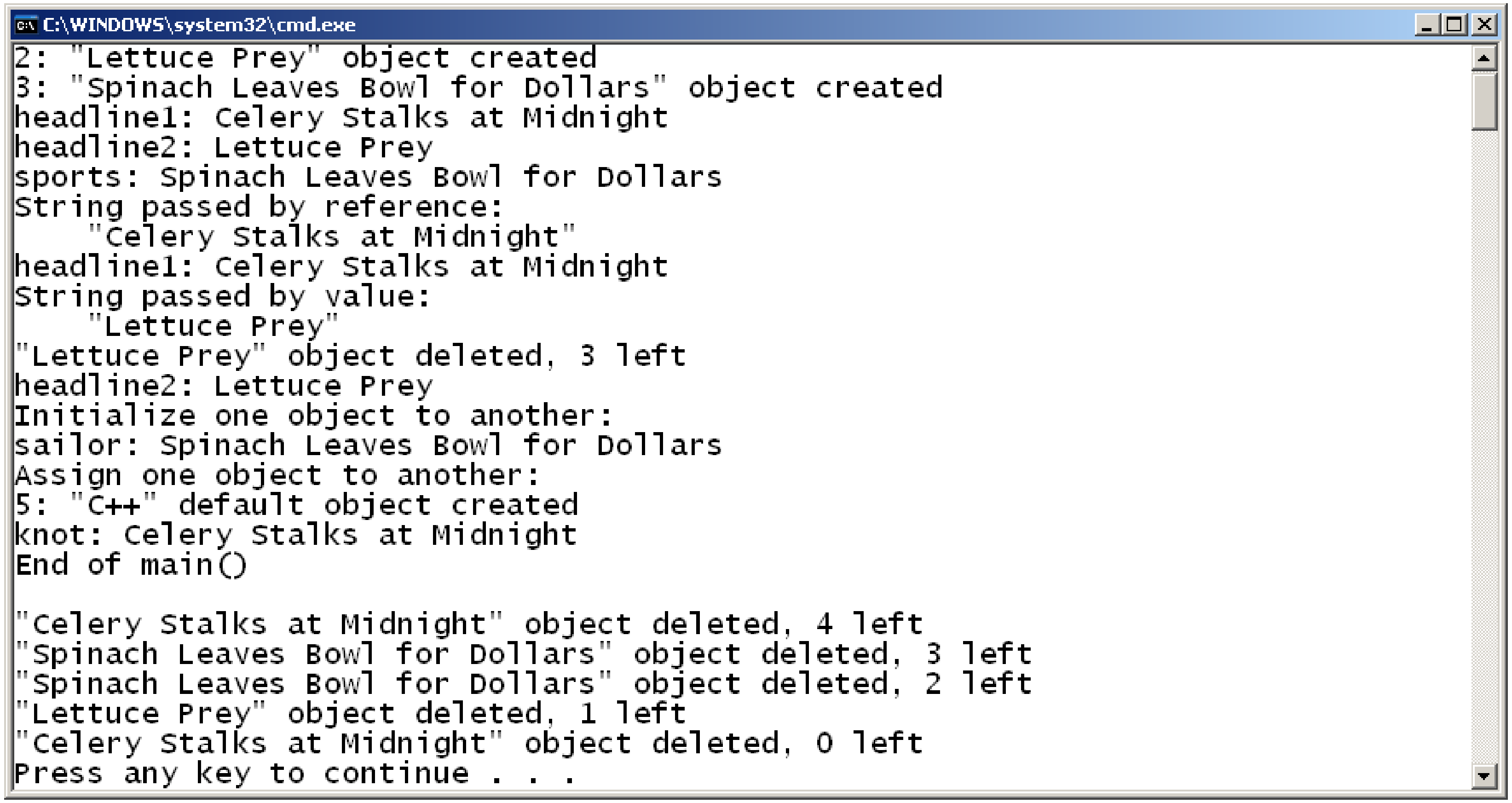
```
StringBad & operator=(const StringBad & st);
```

The Cure

- operator=( ) is called for the object on the left side of the assignment.
- Add to strngbad.cpp:

```
StringBad & StringBad::operator=(const StringBad & st)
{
    if (this == &st)        // Assignment to self?
    {
        return *this;
    }
    delete [] str;
    len = st.len;
    str = new char[len + 1];
    std::strcpy(str, st.str);
    return *this;
}
```

It Works Now!



```
2: "Lettuce Prey" object created
3: "Spinach Leaves Bowl for Dollars" object created
headline1: Celery Stalks at Midnight
headline2: Lettuce Prey
sports: Spinach Leaves Bowl for Dollars
String passed by reference:
    "Celery Stalks at Midnight"
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 3 left
headline2: Lettuce Prey
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
Assign one object to another:
5: "C++" default object created
knot: Celery Stalks at Midnight
End of main()

"Celery Stalks at Midnight" object deleted, 4 left
"Spinach Leaves Bowl for Dollars" object deleted, 3 left
"Spinach Leaves Bowl for Dollars" object deleted, 2 left
"Lettuce Prey" object deleted, 1 left
"Celery Stalks at Midnight" object deleted, 0 left
Press any key to continue . . .
```

Assignment

- Read and study carefully Chapter 12, down through page 646