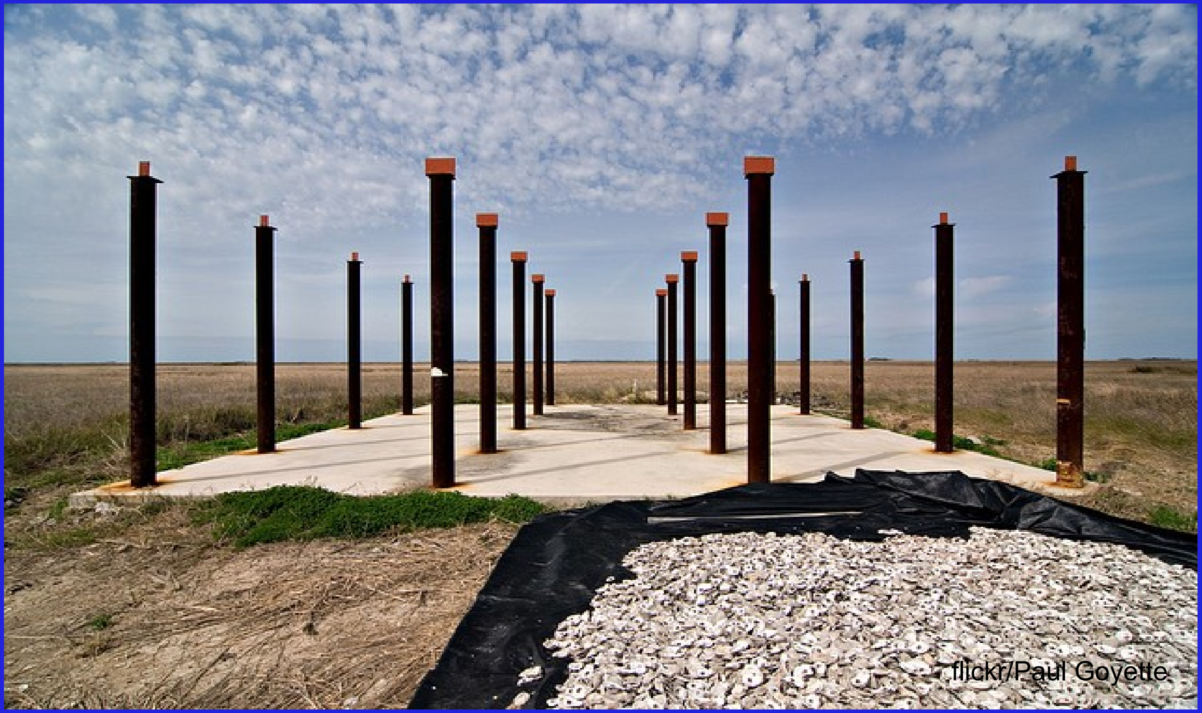
# C++ Foundation



flickr/Paul Goyette

## A Brief Tour of C++

# A Brief Tour of C++

- By example...

Your task is to create an LCD string representation of an integer value using a 3x3 grid of space, underscore, and pipe characters for each digit. Each digit is shown below (using a dot instead of a space)

```
._.   ...   ._.   ._.   ...   ._.   ._.   ._.   ._.   ._.
|.|   ..|   ._|   ._|   |_|   |_.   |_.   ..|   |_|   |_|
|_|   ..|   |_.   ._|   ..|   ._|   |_|   ..|   |_|   ..|
```

Example: `910`

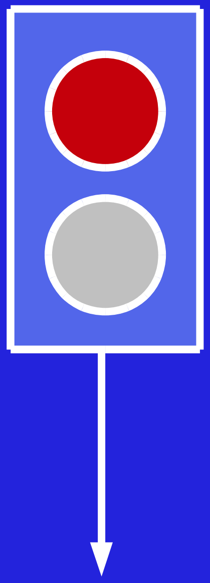
```
._.  ...  ._.
|_|  ..|  |.|
..|  ..|  |_|
```

# Test First Development

- Think of a test as an executable specification

```
...
int main()
{
    lcd_spec(0, lcd(
        “ _ ”,
        “| |”,
        “|_|”
    ));

    std::cout << “All passed”
              << std::endl;
}
```

lcd_tests.cpp



A test that doesn't compile yet certainly counts as a failing test

# Test First Development

- Our tests use this helper function

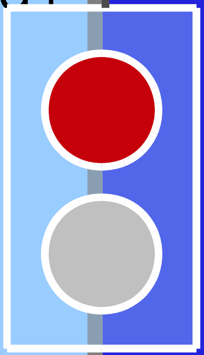
```cpp
#include <string>
#include <vector>

typedef std::vector<std::string> lcd_grid;

lcd_grid lcd(string s1, string s2, string s3)
{
    lcd_grid result;
    result.push_back(s1);
    result.push_back(s2);
    result.push_back(s3);
    return result;
}
```

lcd.cpp

# Test First Development

lcd_tests.cpp

```
#include “lcd.hpp”
#include <iostream>
...
void lcd_spec(int value, lcd_grid grid)
{
    std::string expected = to_string(grid),
                actual = to_string(lcd(value));
    if (expected != actual)
    {
        std::cerr
            << “lcd(“ value << “)” << std::endl
            << “expected== << std::endl
            << expected << std::endl
            << “actual==” << std::endl
            << actual << std::endl;
        std::exit(EXIT_FAILURE);
    }
}
```



# Test First Development

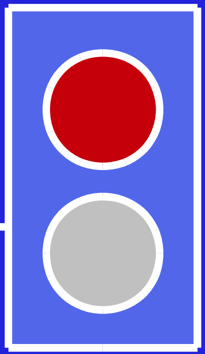
- Get the tests to compile and link

lcd.hpp

```
#ifndef LCD_INCLUDED
#define LCD_INCLUDED

#include <string>
#include <vector>

typedef std::vector<std::string> lcd_grid;

lcd_grid lcd(int value);

#endif
```

lcd.cpp

```
#include “lcd.hpp”

lcd_grid lcd(int value)
{
    throw “to do”;
}
```

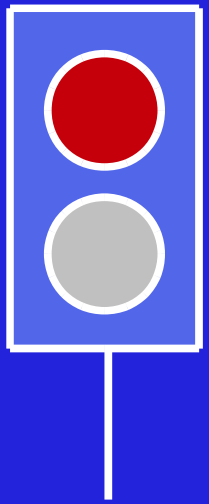


# Test First Development

- The tests now run, but fail, the perfect start :-)

```
...
int main()
{
    lcd_spec(0, lcd(
        “ _ “,
        “| |”,
        “|_|”
    ));
}
```

lcd_tests.cpp



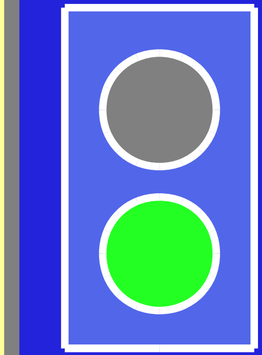
```
$g++ -Wall -Wextra lcd*.cpp && ./a.out
terminate called after thowing an instance
of char const *
```

# Test First Development

- Make the tests pass

```
const lcd_grid digits[] =
{
    lcd(“ _ “,
        “| |”,
        “|_|”
    ),
};

lcd_grid lcd(int value)
{
    if (value == 0)
        return digits[0];
    else
        throw “to do”;
}
```

lcd.cpp

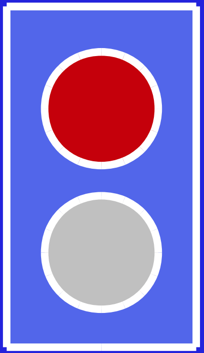


# Test First Development

- Write another failing test

```
...
int main()
{
    ...
    #define WS “ “

    lcd_spec(12, lcd(
        “   “ WS “ _ “,
        “  |” WS “ _|”,
        “  |” WS “|_ ”
    ));

    #undef WS
}
```

lcd_tests.cpp

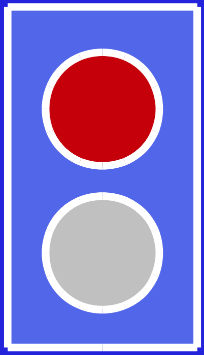


# Test First Development

- Make the tests pass

```
const lcd_grid digits[] =
{
    lcd(“ _ “,
        “| |”,
        “|_|”
    ),
    lcd(“   “,
        “  |”,
        “  |”
    ),
    lcd(“ _ “,
        “ _|”,
        “|_ “
    ),
};
```

lcd.cpp

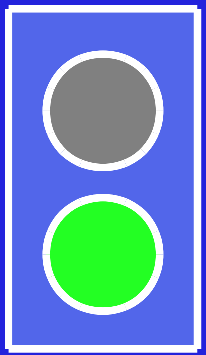


# Test First Development

- Get the test to pass

```
lcd_grid lcd::grid(int value)
{
    if (value < 10)
        return digits[value];
    else
    {
        lcd_grid lhs = lcd(value / 10);
        lcd_grid rhs = digits[value % 10];
        return lcd(
            lhs[0] + “ “ + rhs[0],
            lhs[1] + “ “ + rhs[1],
            lhs[2] + “ “ + rhs[2]);
    }
}
```

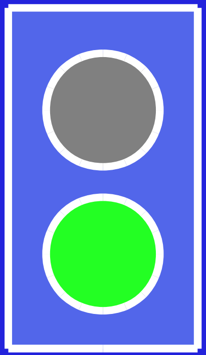
lcd.cpp



# Test First Development

- Refactor when at green

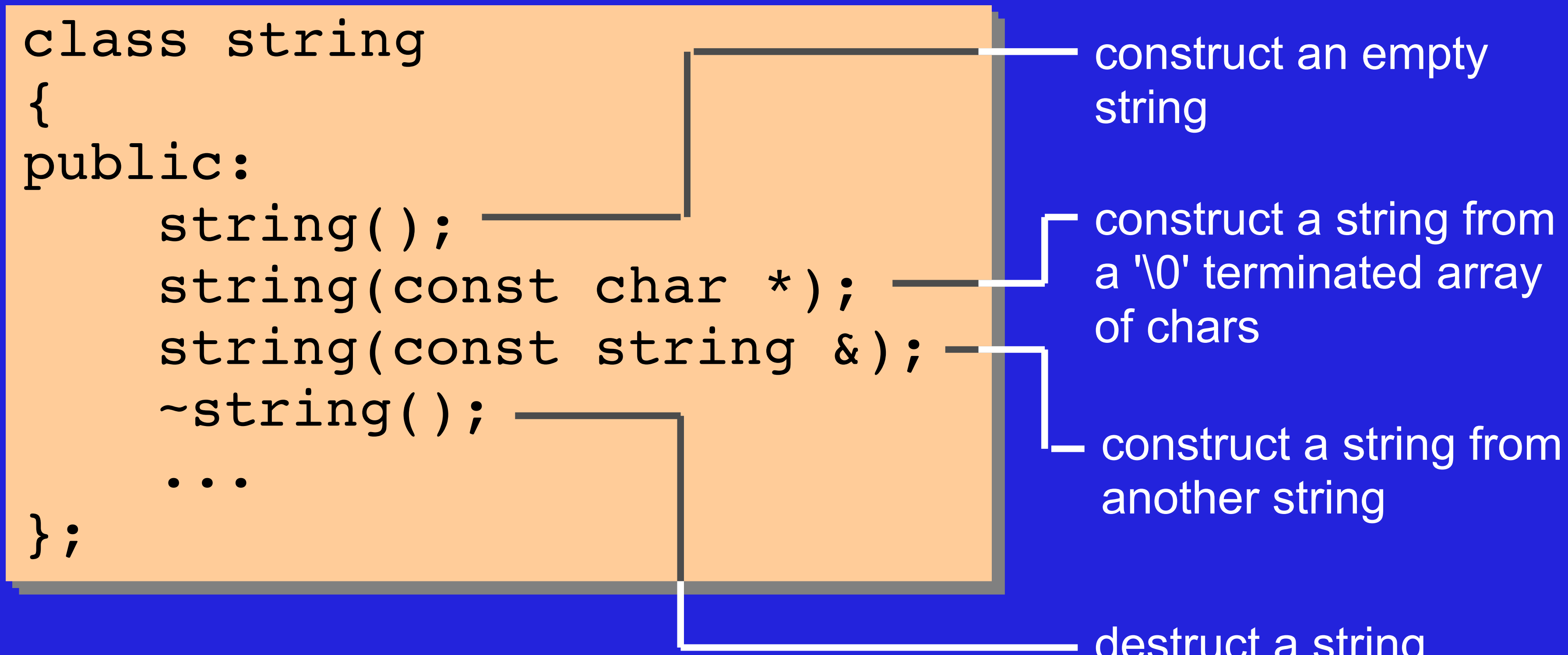
```
lcd_grid lcd::grid(int value)
{
    if (value < 10)
        return digits[value];
    else
    {
        lcd_grid lhs = lcd(value / 10),
                 rhs = digits[value % 10];
        const std::string ws = “ “;
        return lcd(
            lhs[0] + ws + rhs[0],
            lhs[1] + ws + rhs[1],
            lhs[2] + ws + rhs[2]);
    }
}
```



lcd.cpp

# What Did We Use?

- std::string - to abstract away char* horribleness



```
class string
{
public:
    string();
    string(const char *);
    string(const string &);
    ~string();
    ...
};
```

construct an empty string

construct a string from a '\0' terminated array of chars

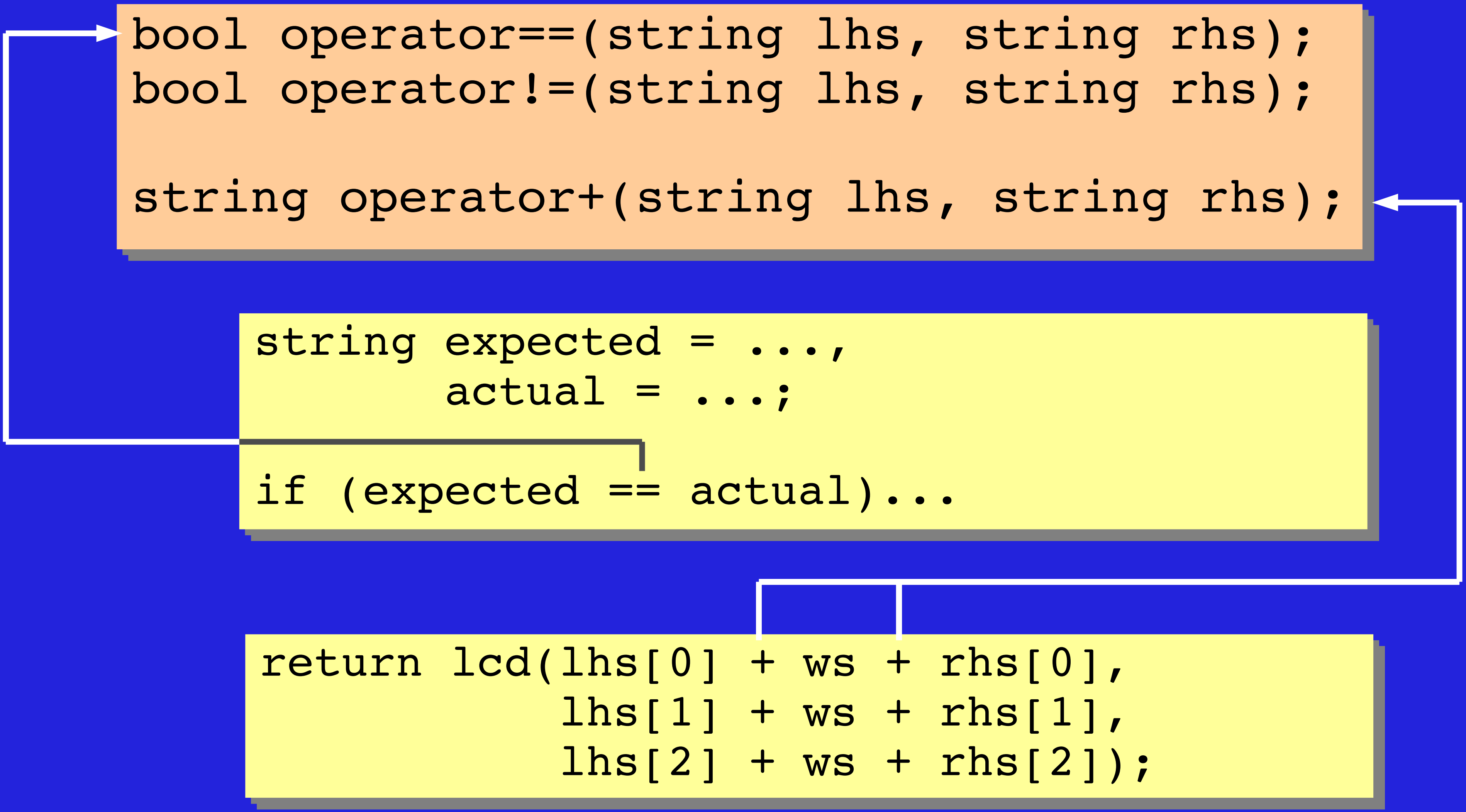
construct a string from another string

destruct a string

simplified (string is actually a typedef)

# What Did We Use?

- std::string - to abstract away char* horribleness



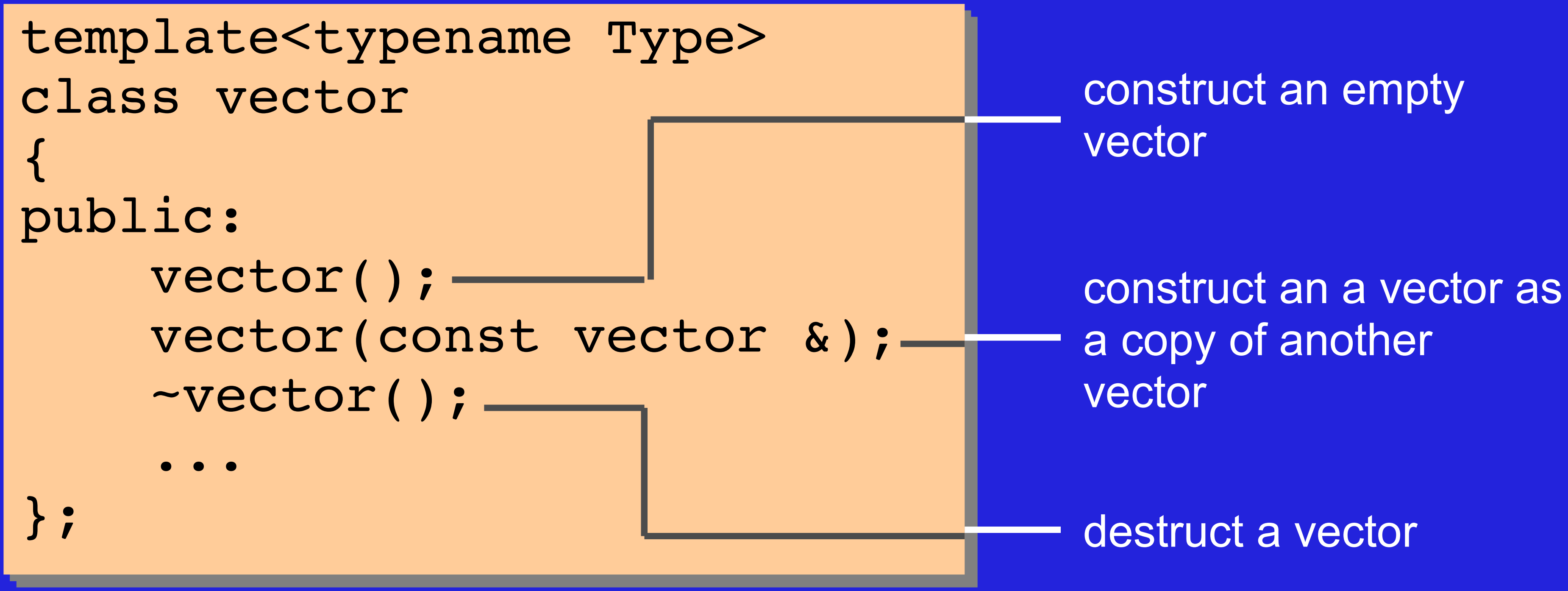
```
bool operator==(string lhs, string rhs);
bool operator!=(string lhs, string rhs);

string operator+(string lhs, string rhs);
```

```
string expected = ...,
       actual = ...;

if (expected == actual)...
```

```
return lcd(lhs[0] + ws + rhs[0],
           lhs[1] + ws + rhs[1],
           lhs[2] + ws + rhs[2]);
```

# What Did We Use?

- std::vector<> - a resizeable array



```
template<typename Type>
class vector
{
public:
    vector();
    vector(const vector &);
    ~vector();
    ...
};
```

construct an empty vector

construct an a vector as a copy of another vector

destruct a vector

simplified

# What Did We Use?

- std::vector<> - a resizeable array

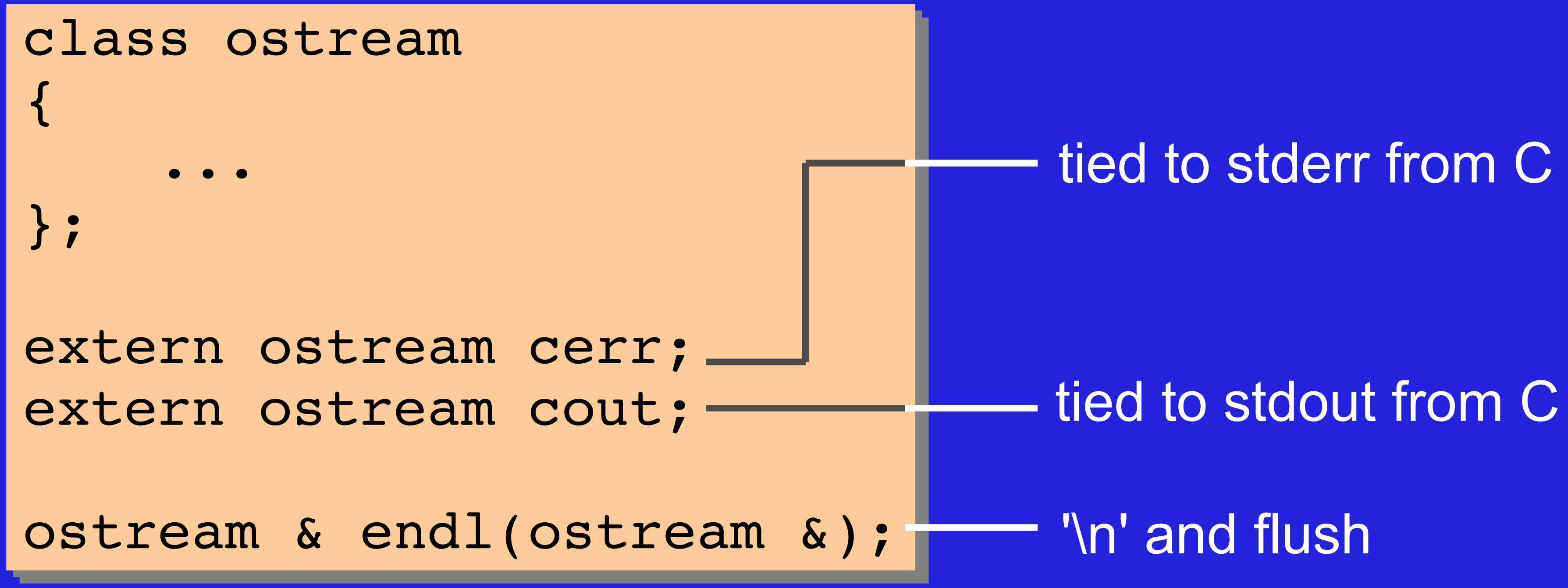
```
template<typename Type>
class vector
{
public:
    void push_back(Type pushed);
    ...
    Type & operator[](size_t at);
    ...
};
```

```
std::vector<std::string> result;
result.push_back(s1);
```

```
std::vector<std::string> lhs = ...;
std::vector<std::string> rhs = ...;
    lhs[0] ...  rhs[0]
    lhs[1] ...  rhs[1]
```

# What Did We Use?

- std::ostream - an output stream



```
class ostream
{
      ...
};

extern ostream cerr;
extern ostream cout;

ostream & endl(ostream &);
```

tied to stderr from C

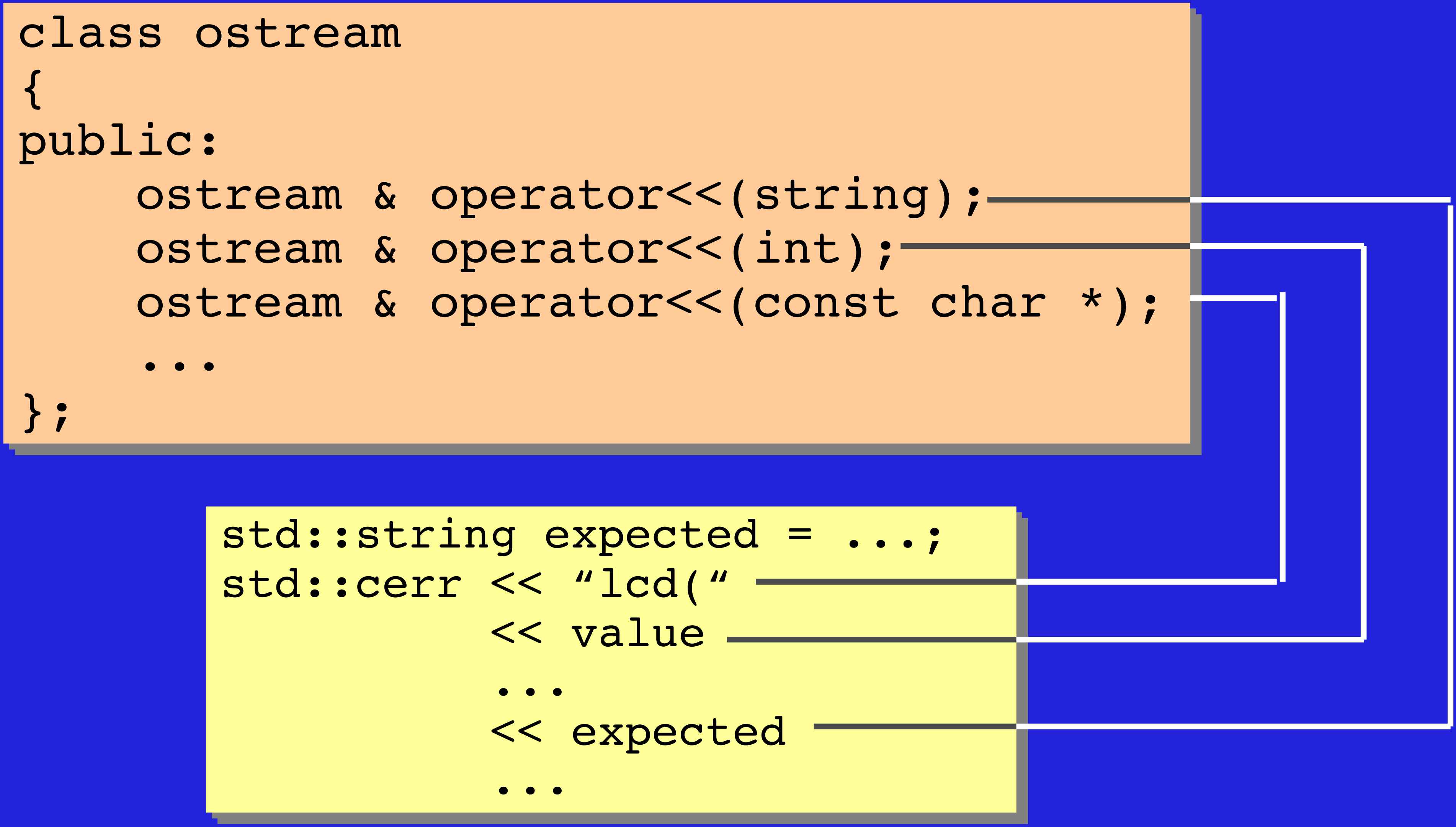
tied to stdout from C

'\n' and flush

simplified

# What Did We Use?

- std::ostream - an output stream



```
class ostream
{
public:
    ostream & operator<<(string);
    ostream & operator<<(int);
    ostream & operator<<(const char *);
    ...
};
```

```
std::string expected = ...;
std::cerr << “lcd(“
          << value
          ...
          << expected
          ...
```