CSE 1325: Object-Oriented Programming

Lecture 18 – Chapter 11

Custom Input / Output
Strategy Pattern + UML Activity

Mr. George F. Rice

george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Homework #8 Questions?

- Homework #7 suggested solution did NOT have a “main”, so “make” produced an error (original) or a warning (update #1)
  - “make test” produced the executable “test”
- **All grades are posted.** If you don’t see a grade, or believe a grade is incorrect, contact the grader first
  - For homework, contact the TA.
    - If no response within 2 days, contact me.
  - For pop quizzes and exams, that’s me.
    - If no response within 2 days, email again or stop by my office.

Debugging in the Twilight Zone™

- Not all bugs are your fault
  - Though *almost* all are
- Here’s an example where g++ and gtkmm interacted in a most... unfortunate... way...
  in the Twilight Zone

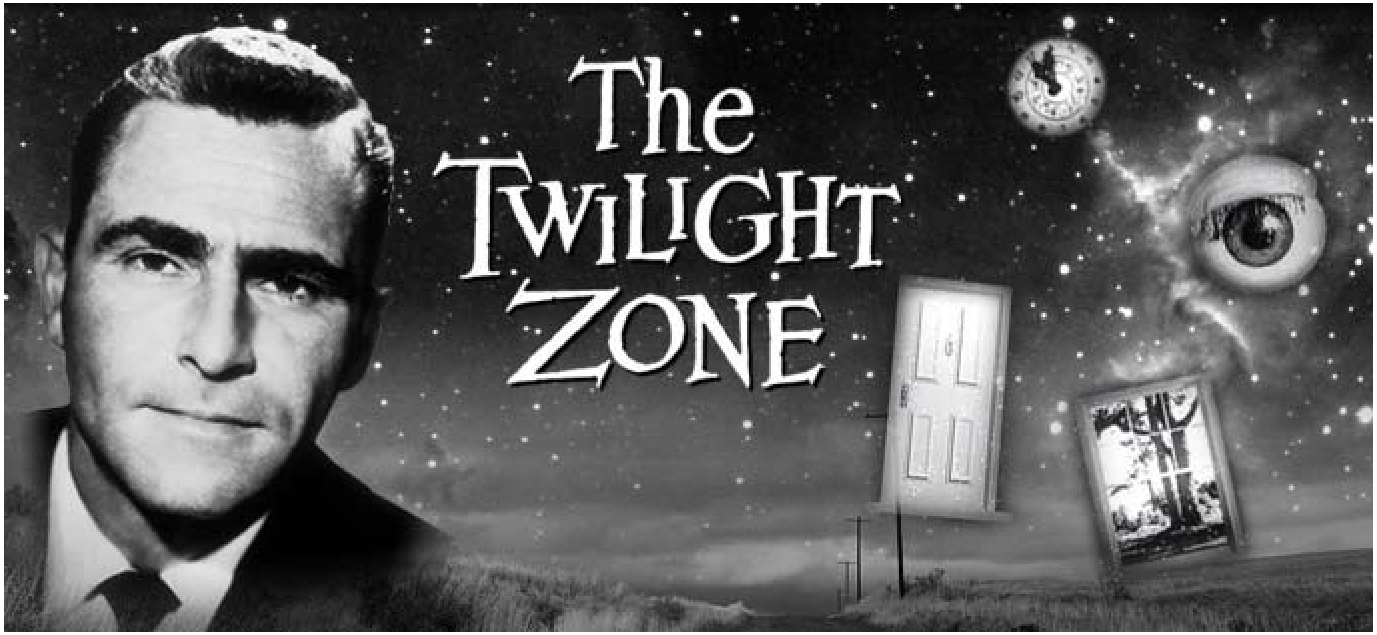
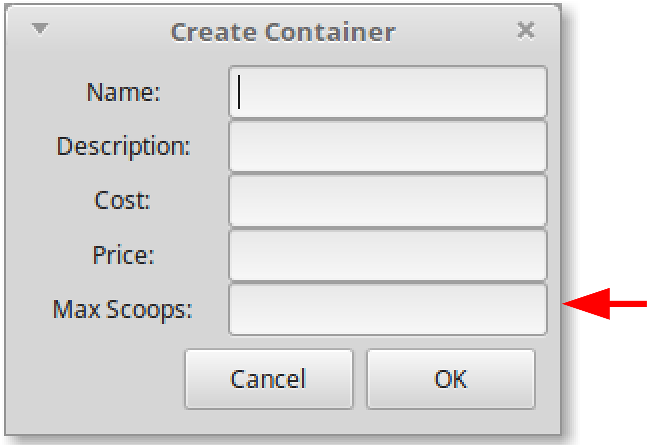
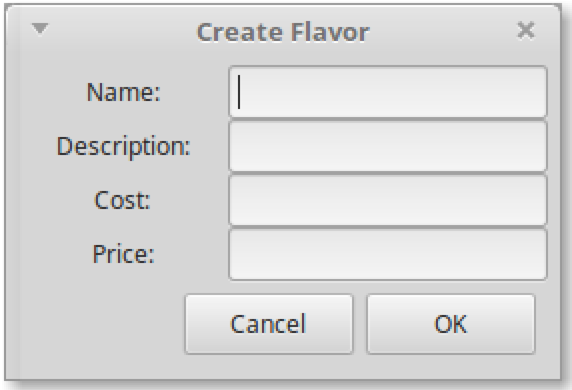


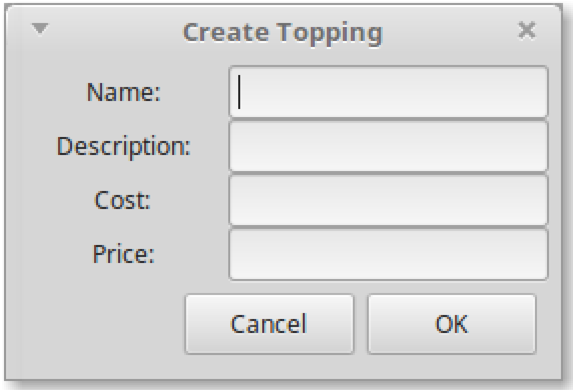
Image © 1959 CBS Productions. Fair use for educational purposes is asserted.

It’s a Warm Fall Day...

- I wanted a single dialog instance to be used to create ice cream flavors, toppings, AND containers
  - Containers need an extra field – max_scoops







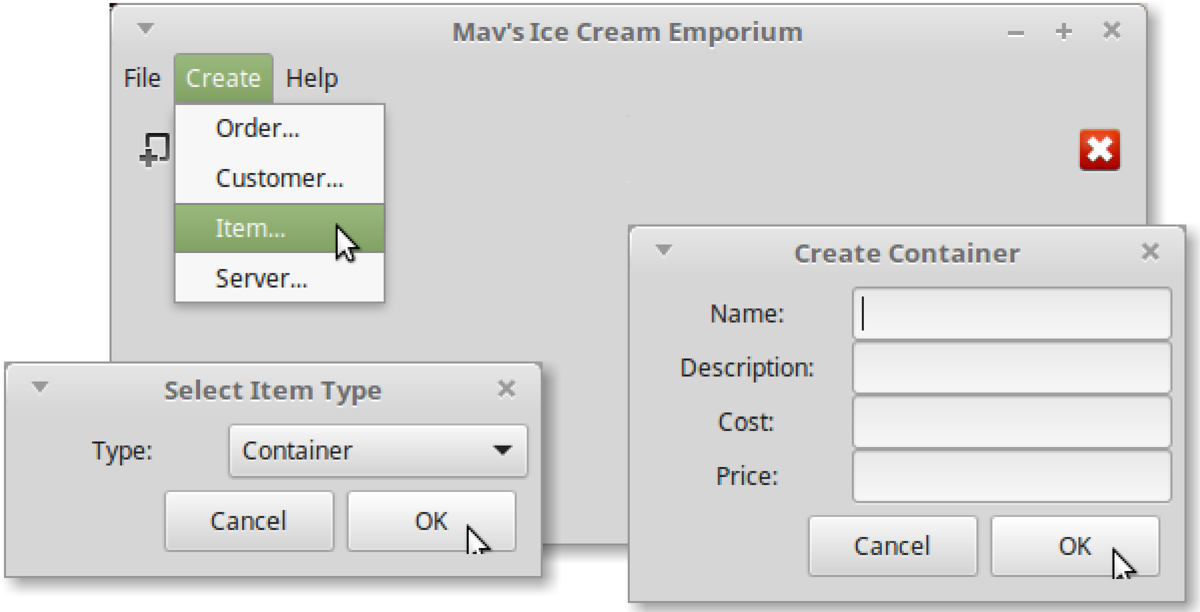
...The Unsuspecting Prof...

- Solution? A simple “if” should do the trick

```
void Mainwin::on_create_item_click() {
    // ...Display a drop down to select item_type as CONTAINER, FLAVOR, or TOPPING

    Gtk::Dialog dialog;
    if (item_type == CONTAINER) dialog.set_title("Create Container");
    else if (item_type == SCOOP) dialog.set_title("Create Flavor");
    else dialog.set_title("Create Topping");
    dialog.set_transient_for(*this);
    // ...Add 4 entry fields for Name, Description, Cost, and Price

    if (item_type == CONTAINER) {  // Add an extra entry field if this is a container
        Gtk::HBox b_max_scoops;

        Gtk::Label l_max_scoops{"Max Scoops:"};
        l_max_scoops.set_width_chars(WIDTH);
        b_max_scoops.pack_start(l_max_scoops, Gtk::PACK_SHRINK);

        Gtk::Entry e_max_scoops;
        e_max_scoops.set_max_length(WIDTH*4);
        b_max_scoops.pack_start(e_max_scoops, Gtk::PACK_SHRINK);
        dialog.get_vbox()->pack_start(b_max_scoops, Gtk::PACK_SHRINK);
    }
```

...is startled by...

- Wait – where’s the entry for Max Scoops???
- How would you debug this problem?



Type: Container

Name:

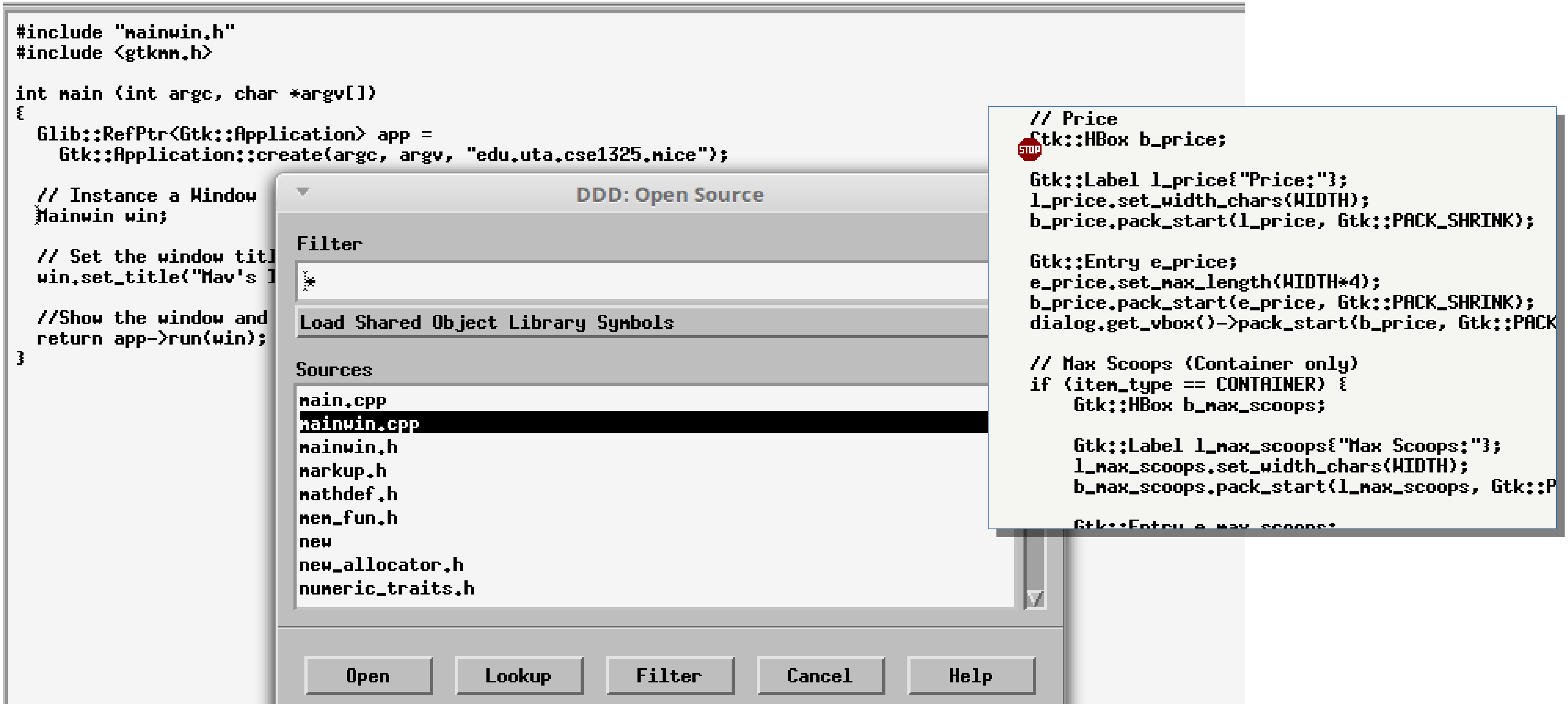
Description:

Cost:

Price:

...an unexpected behavior,...

- Right – run the debugger, open mainwin.cpp, and then set a breakpoint before the if
  - “Step” into the if to determine why it’s skipped



```
#include "mainwin.h"
#include <gtkmm.h>

int main (int argc, char *argv[])
{
  Glib::RefPtr<Gtk::Application> app =
    Gtk::Application::create(argc, argv, "edu.uta.cse1325.mice");

  // Instance a Window
  Mainwin win;

  // Set the window titl
  win.set_title("Mav's I

  //Show the window and
  return app->run(win);
}
```

Filter

*

Load Shared Object Library Symbols

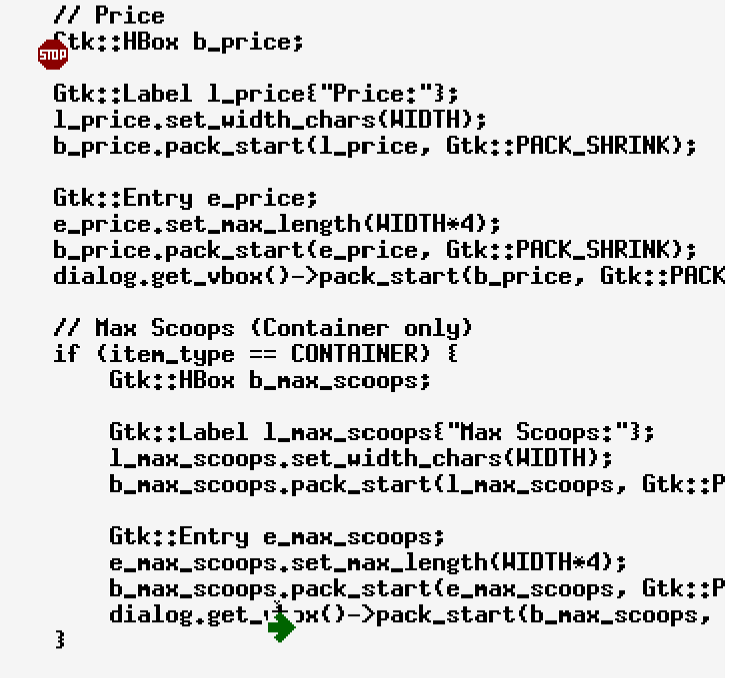
Sources

```
main.cpp
mainwin.cpp
mainwin.h
markup.h
mathdef.h
mem_fun.h
new
new_allocator.h
numeric_traits.h
```

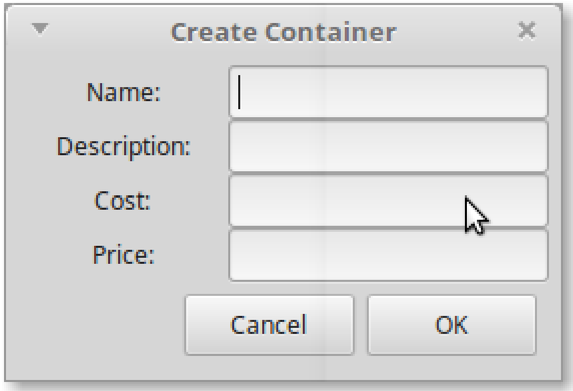
```
    // Price
    Gtk::HBox b_price;

    Gtk::Label l_price{"Price:"};
    l_price.set_width_chars(WIDTH);
    b_price.pack_start(l_price, Gtk::PACK_SHRINK);

    Gtk::Entry e_price;
    e_price.set_max_length(WIDTH*4);
    b_price.pack_start(e_price, Gtk::PACK_SHRINK);
    dialog.get_vbox()->pack_start(b_price, Gtk::PACK

    // Max Scoops (Container only)
    if (item_type == CONTAINER) {
        Gtk::HBox b_max_scoops;

        Gtk::Label l_max_scoops{"Max Scoops:"};
        l_max_scoops.set_width_chars(WIDTH);
        b_max_scoops.pack_start(l_max_scoops, Gtk::P

        Gtk::Entry e_max_scoops;
```

...causing a seismic disruption...

- Wait... what?!?!?
  - The code for the 5th entry *is being executed*!
  - But the 5th entry isn’t displayed

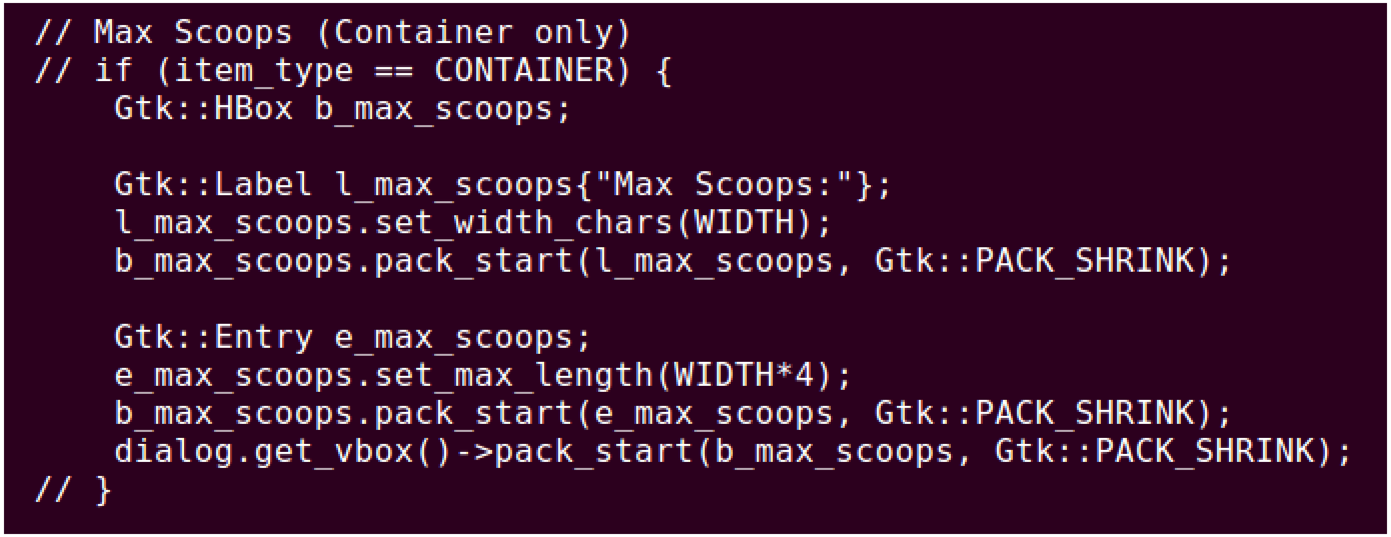


```
// Price
Gtk::HBox b_price;

Gtk::Label l_price{"Price:"};
l_price.set_width_chars(WIDTH);
b_price.pack_start(l_price, Gtk::PACK_SHRINK);

Gtk::Entry e_price;
e_price.set_max_length(WIDTH*4);
b_price.pack_start(e_price, Gtk::PACK_SHRINK);
dialog.get_vbox()->pack_start(b_price, Gtk::PACK

// Max Scoops (Container only)
if (item_type == CONTAINER) {
    Gtk::HBox b_max_scoops;

    Gtk::Label l_max_scoops{"Max Scoops:"};
    l_max_scoops.set_width_chars(WIDTH);
    b_max_scoops.pack_start(l_max_scoops, Gtk::P

    Gtk::Entry e_max_scoops;
    e_max_scoops.set_max_length(WIDTH*4);
    b_max_scoops.pack_start(e_max_scoops, Gtk::P
    dialog.get_vbox()->pack_start(b_max_scoops,
}
```

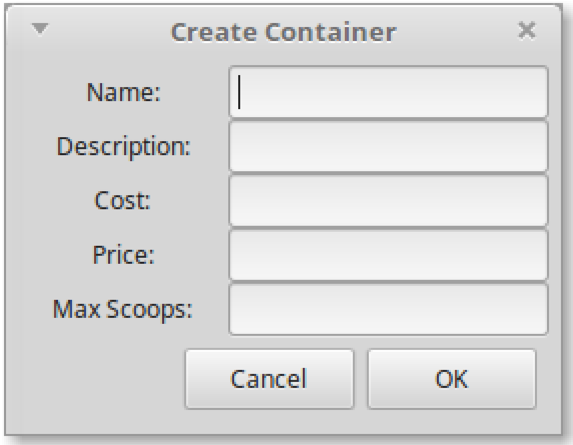


...in the space / time continuum...

- OK. So clearly running the code inside the “if” does NOT create the 5th entry
  - But let’s confirm that assumption by commenting out the “if”



```
// Max Scoops (Container only)
// if (item_type == CONTAINER) {
    Gtk::HBox b_max_scoops;

    Gtk::Label l_max_scoops{"Max Scoops:"};
    l_max_scoops.set_width_chars(WIDTH);
    b_max_scoops.pack_start(l_max_scoops, Gtk::PACK_SHRINK);

    Gtk::Entry e_max_scoops;
    e_max_scoops.set_max_length(WIDTH*4);
    b_max_scoops.pack_start(e_max_scoops, Gtk::PACK_SHRINK);
    dialog.get_vbox()->pack_start(b_max_scoops, Gtk::PACK_SHRINK);
// }
```



Create Container

Name:

Description:

Cost:

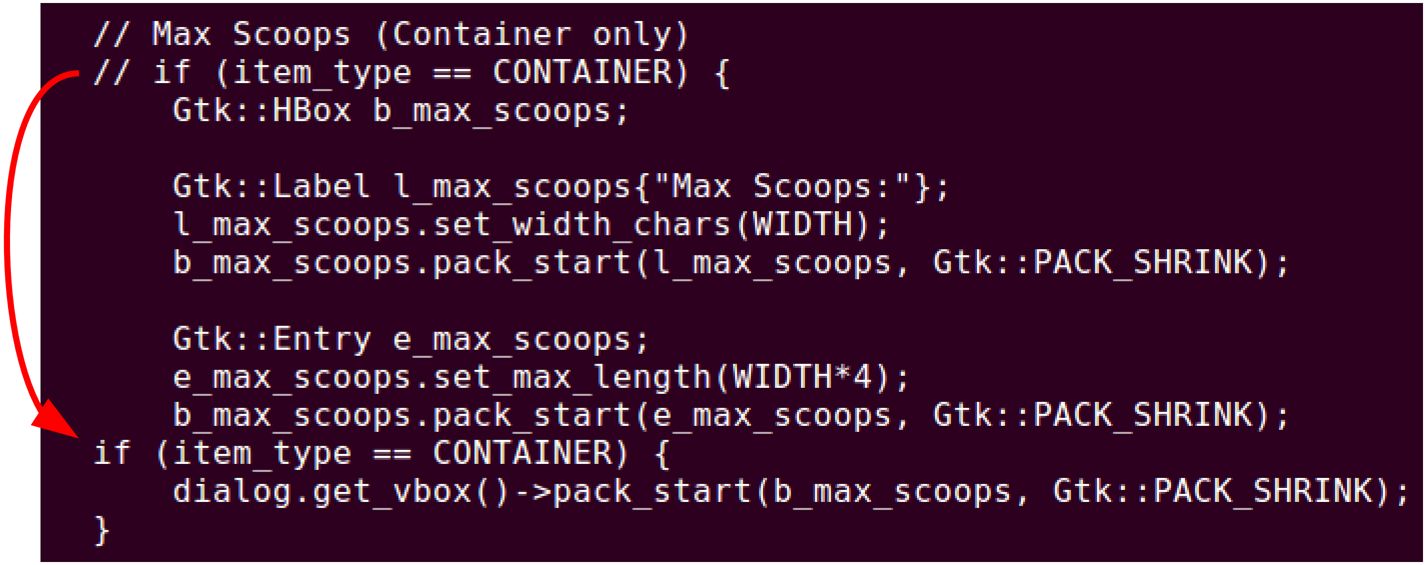
Price:

Max Scoops:

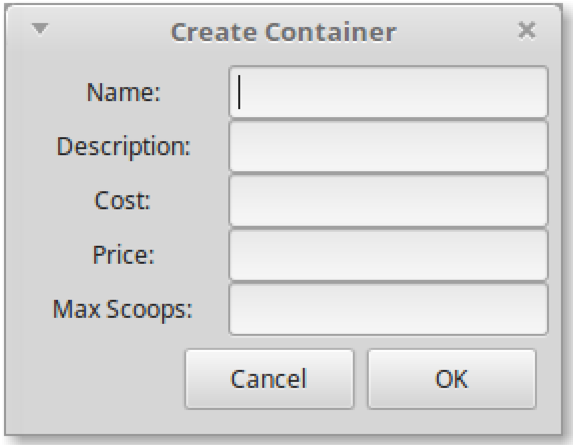
Noooooooooo...
Now what?

...and renders logic impotent...

- Yes, clearly the logic of our code is just fine
  - But *possibly* the C++ compiler and / or gtkmm isn’t accurately implementing our logic
  - Time to play “what if...”
- “What if” the optimizer is removing some needed code?
  - Let’s move the “if” and see if that changes anything



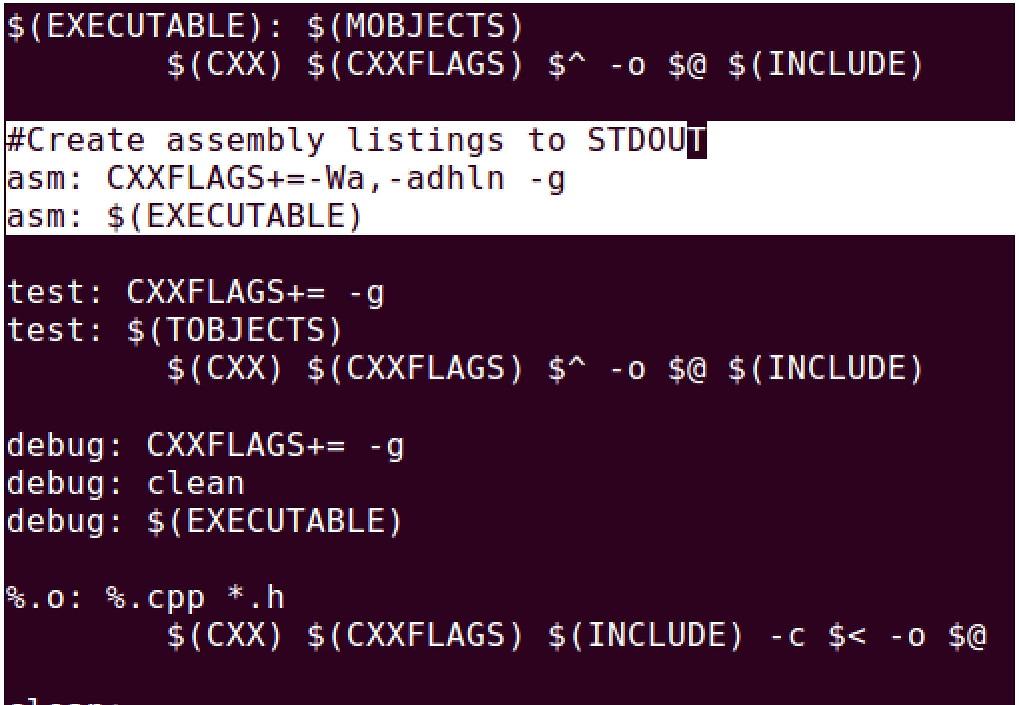
```
// Max Scoops (Container only)
// if (item_type == CONTAINER) {
    Gtk::HBox b_max_scoops;

    Gtk::Label l_max_scoops{"Max Scoops:"};
    l_max_scoops.set_width_chars(WIDTH);
    b_max_scoops.pack_start(l_max_scoops, Gtk::PACK_SHRINK);

    Gtk::Entry e_max_scoops;
    e_max_scoops.set_max_length(WIDTH*4);
    b_max_scoops.pack_start(e_max_scoops, Gtk::PACK_SHRINK);
if (item_type == CONTAINER) {
    dialog.get_vbox()->pack_start(b_max_scoops, Gtk::PACK_SHRINK);
}
```



This works.
But *why* does it work?

...except in the face...

- We have another weapon – the assembly view!
  - We can see the assembly produced for each line of C++ code in our program
  - Perhaps this will reveal the cause of this problem
- Use the intuitively (ahem) named “-Wa,-adhln -g” option

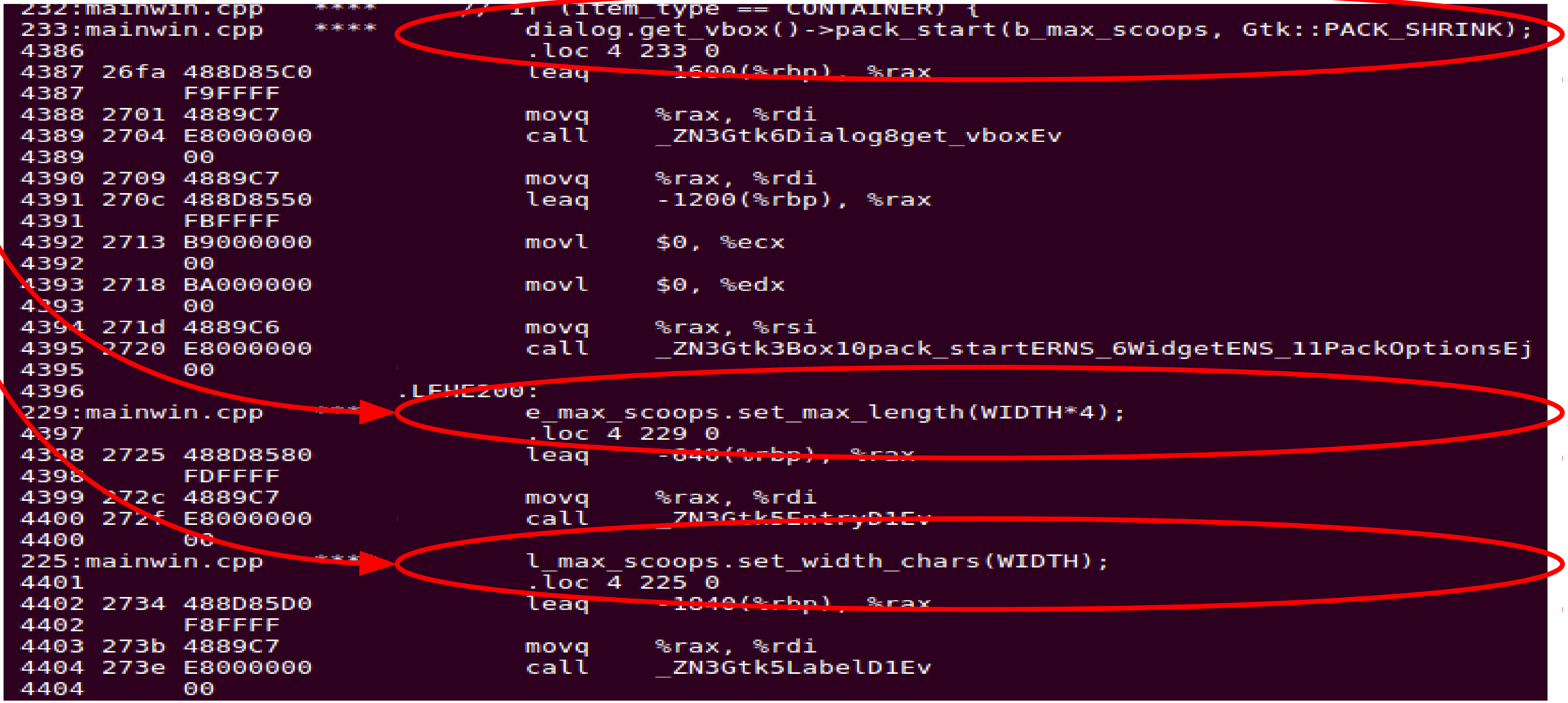


```
$(EXECUTABLE): $(MOBJECTS)
        $(CXX) $(CXXFLAGS) $^ -o $@ $(INCLUDE)

#Create assembly listings to STDOUT
asm: CXXFLAGS+=-Wa,-adhln -g
asm: $(EXECUTABLE)

test: CXXFLAGS+= -g
test: $(TOBJECTS)
        $(CXX) $(CXXFLAGS) $^ -o $@ $(INCLUDE)

debug: CXXFLAGS+= -g
debug: clean
debug: $(EXECUTABLE)

%.o: %.cpp *.h
        $(CXX) $(CXXFLAGS) $(INCLUDE) -c $< -o $@

clean:
```

Type “make asm > temp.txt” to use this Makefile, as the assembly code is sent to STDOUT instead of the .o file.

...of the right tools!

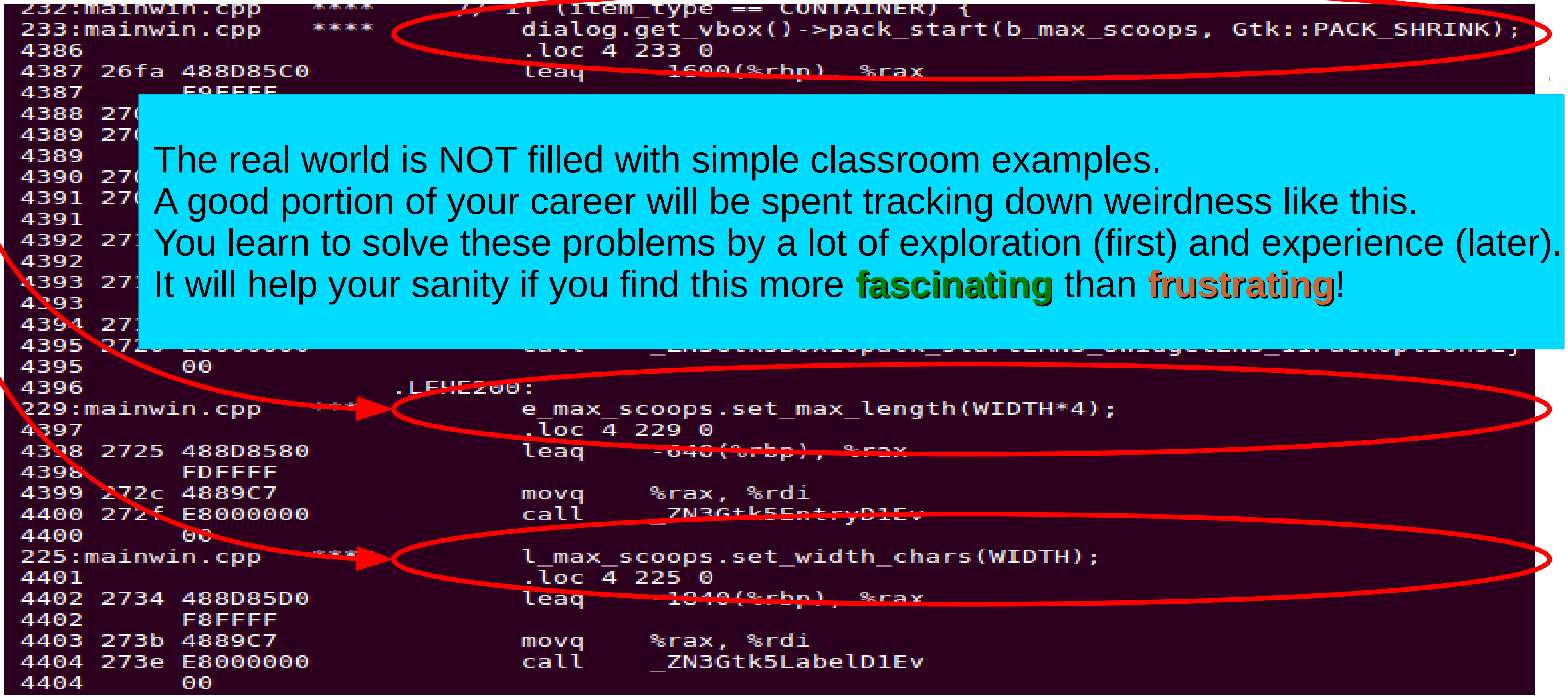
- And there's the problem (from compiling our original code)
  - g++ is moving config of our 5th entry to AFTER it has been added to the dialog's VBox!



```
232:mainwin.cpp   ****        // if (item_type == CONTAINER) {
233:mainwin.cpp   ****           dialog.get_vbox()->pack_start(b_max_scoops, Gtk::PACK_SHRINK);
4386                             .loc 4 233 0
4387 26fa 488D85C0               leaq    -1600(%rbp), %rax
4387      F9FFFF
4388 2701 4889C7                 movq    %rax, %rdi
4389 2704 E8000000               call    _ZN3Gtk6Dialog8get_vboxEv
4389      00
4390 2709 4889C7                 movq    %rax, %rdi
4391 270c 488D8550               leaq    -1200(%rbp), %rax
4391      FBFFFF
4392 2713 B9000000               movl    $0, %ecx
4392      00
4393 2718 BA000000               movl    $0, %edx
4393      00
4394 271d 4889C6                 movq    %rax, %rsi
4395 2720 E8000000               call    _ZN3Gtk3Box10pack_startERNS_6WidgetENS_11PackOptionsEj
4395      00
4396                      .LEHE200:
229:mainwin.cpp   ****           e_max_scoops.set_max_length(WIDTH*4);
4397                             .loc 4 229 0
4398 2725 488D8580               leaq    -640(%rbp), %rax
4398      FDFFFF
4399 272c 4889C7                 movq    %rax, %rdi
4400 272f E8000000               call    _ZN3Gtk5EntryD1Ev
4400      00
225:mainwin.cpp   ****           l_max_scoops.set_width_chars(WIDTH);
4401                             .loc 4 225 0
4402 2734 488D85D0               leaq    -1040(%rbp), %rax
4402      F8FFFF
4403 273b 4889C7                 movq    %rax, %rdi
4404 273e E8000000               call    _ZN3Gtk5LabelD1Ev
4404      00
```

...of the right tools!

- And there's the problem (from compiling our original code)
  - g++ is moving config of our 5th entry to AFTER it has been added to the dialog's VBox!

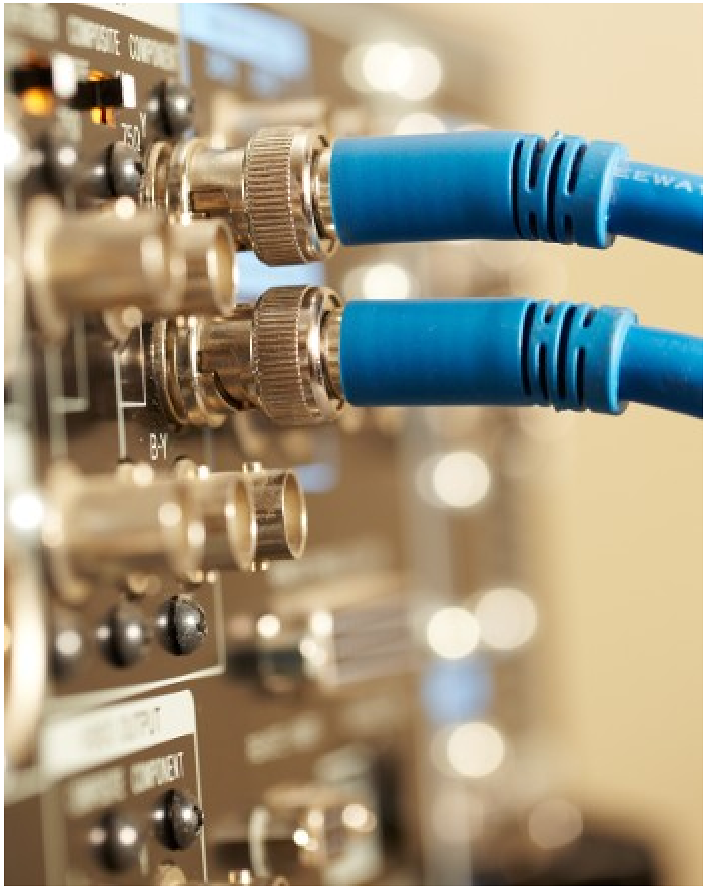


```
232:mainwin.cpp   ****        // if (item_type == CONTAINER) {
233:mainwin.cpp   ****           dialog.get_vbox()->pack_start(b_max_scoops, Gtk::PACK_SHRINK);
4386                             .loc 4 233 0
4387 26fa 488D85C0               leaq    -1600(%rbp), %rax
4387      F9FFFF
4388 27
4389 27
4389
4390 27
4391 27
4391
4392 27
4392
4393 27
4393
4394 27
4395 2720 E8000000               call    _ZN3Gtk3Box10pack_startERNS_6WidgetENS_11PackOptionsEj
4395      00
4396                     .LEHE200:
229:mainwin.cpp   ****           e_max_scoops.set_max_length(WIDTH*4);
4397                             .loc 4 229 0
4398 2725 488D8580               leaq    -640(%rbp), %rax
4398      FDFFFF
4399 272c 4889C7                 movq    %rax, %rdi
4400 272f E8000000               call    _ZN3Gtk5EntryD1Ev
4400      00
225:mainwin.cpp   ****           l_max_scoops.set_width_chars(WIDTH);
4401                             .loc 4 225 0
4402 2734 488D85D0               leaq    -1840(%rbp), %rax
4402      F8FFFF
4403 273b 4889C7                 movq    %rax, %rdi
4404 273e E8000000               call    _ZN3Gtk5LabelD1Ev
4404      00
```

The real world is NOT filled with simple classroom examples.
A good portion of your career will be spent tracking down weirdness like this.
You learn to solve these problems by a lot of exploration (first) and experience (later).
It will help your sanity if you find this more **fascinating** than **frustrating**!

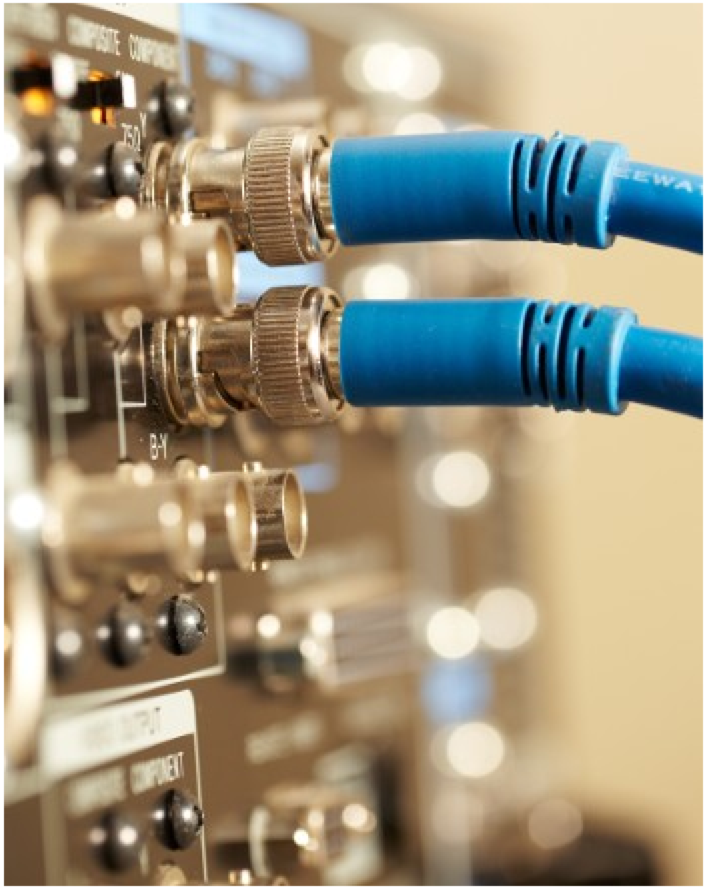
Overview

- Text Formatting
  - Manipulators
  - String Streams
  - Characters
- Files
  - Open Modes
  - Text vs Binary
  - Random Access
- Strategy Pattern
- UML Activity Diagram



Types of (Data) I/O

- Individual values
  - See Chapters 4, 10
- Streams
  - See Chapters 10-11
- Graphics and GUI
  - See Chapters 12-16
- Text
  - Type driven, formatted
  - Line oriented
  - Individual characters
- Numeric
  - Integer
  - Floating point
  - User-defined types



Streams vs printf / scanf

(Adapted from the C++ FAQ)

- Compared to printf and scanf, streams are
  - **More type-safe**: The object type is known at compile time, while "%" fields are evaluated at runtime
  - **Less error prone**: Streams require no redundant "%" tokens that must align with the object types
  - **Extensible**: Streams are easily and uniquely defined for each new class. Imagine the chaos if every class defined it own incompatible "%" fields!
  - **Inheritable**: Streams belong to a class hierarchy, meaning anything can be treated as a stream
- Printf / scanf are
  - Significantly faster in some cases (see premature optimization)

https://isocpp.org/wiki/faq/input-output#iostream-vs-stdio

A Stroustrup Observation

- As programmers we prefer regularity and simplicity
  - But, our job is to meet people's expectations
- People are very fussy, and some very particular, and some downright *picky* about the way their output looks
  - They often have good reasons to be
  - Convention and tradition rules – domain-specific vocabularies
    - What does 110 mean?
    - What does 123,456 mean?
    - What does (123) mean?
  - The world of output formats is weirder than you could possibly imagine

Output formats

- Integer values
  - **1234** (decimal)
  - **2322** (octal)
  - **4d2** (hexadecimal)
- Floating point values
  - **1234.57** (general)
  - **1.2345678e+03** (scientific)
  - **1234.567890** (fixed)
- Precision (for floating-point values)
  - **1234.57** (precision 6)
  - **1234.6** (precision 5)
- Fields
  - **|12|** (default for **|** followed by **12** followed by **|**)
  - **| 12|** (**12** in a field of 4 characters)

Numerical Base Output

dec hex oct

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << dec << 1234 << "\t(decimal)\n"
         << hex << 1234 << "\t(hexadecimal)\n"
         << oct << 1234 << "\t(octal)\n";
// The '\t' character is a “tab”
```

- Results

```
1234    (decimal)
4d2     (hexadecimal)
2322    (octal)
```

“Sticky” Manipulators

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << 1234 << '\t'
         << hex << 1234 << '\t'
         << oct << 1234 << '\n';
    cout << 1234 << '\n'; // the octal base is still in effect
```

- Results

```
1234      4d2      2322
2322
```

Most manipulators are “sticky”, and remain in effect until changes. A few are transient, and only affect the next output. “A few” may mean “just setw”, though.

Other Manipulators

showbase noshowbase

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << 1234 << '\t'
         << hex << 1234 << '\t'
         << oct << 1234 << endl;
    cout << showbase << dec;      // show bases via prefix
    cout << 1234 << '\t'
         << hex << 1234 << '\t'
         << oct << 1234 << '\n';
```

- The opposite of showbase is noshowbase

- Results

```
1234     4d2     2322
1234     0x4d2   02322
```

hex   octal

Floating-point Manipulators

defaultfloat scientific fixed

- You can change floating-point output format
  - **defaultfloat – iostream** chooses best format using **n** digits (default)
  - **scientific –** one digit before the decimal point plus exponent; **n** digits after **.**
  - **fixed –** no exponent; **n** digits after the decimal point

```
// simple test:
cout << 1234.56789 << "\t(defaultfloat)\n"
     << fixed << 1234.56789 << "\t(fixed)\n"
     << scientific << 1234.56789 << "\t(scientific)\n";
```

- Results

```
1234.57            (defaultfloat)
1234.567890        (fixed)
1.234568e+03       (scientific)
```

Precision Manipulator

setprecision(digits)

- Precision (the default is 6) from <iomanip>
  - **defaultfloat** – precision is the number of digits
  - **scientific** – precision is the number of digits after the . (dot)
  - **fixed** – precision is the number of digits after the . (dot)

```
// example:
    cout << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
         << scientific << 1234.56789 << '\n';
    cout << general << setprecision(5)
         << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
         << scientific << 1234.56789 << '\n';
    cout << general << setprecision(8)
         << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
         << scientific << 1234.56789 << '\n';
```

- Results (note the rounding):

```
1234.57          1234.567890      1.234568e+03
1234.6           1234.56789       1.23457e+03
1234.5679        1234.56789000    1.23456789e+03
```

Output field width

setw(min_width)

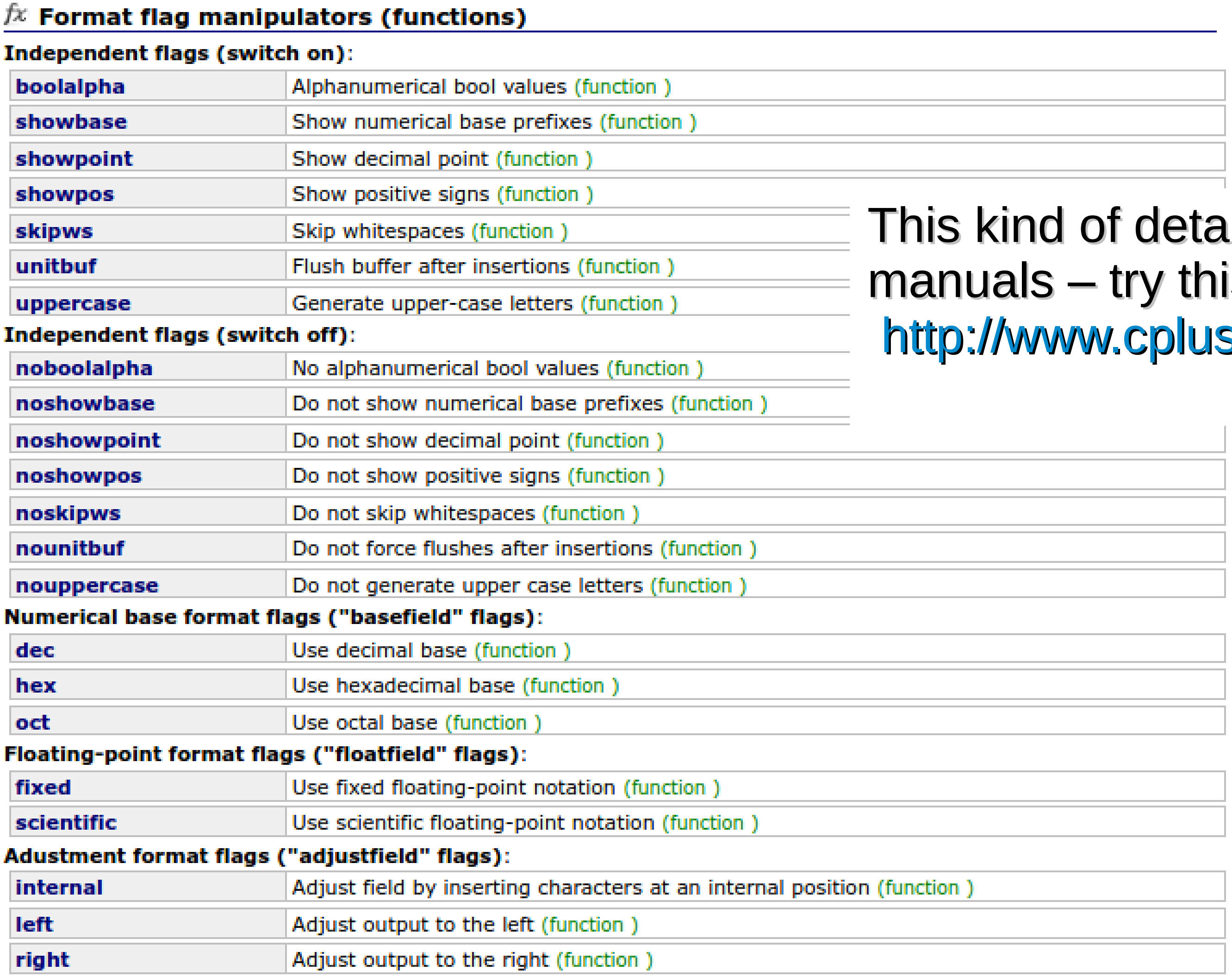
- Width is the number of characters to be used for the next output operation
  - **Beware:** width is transient and applies to next output only (it doesn’t “stick” like precision, base, and floating-point format)
  - **Beware:** output is never truncated to fit into field
    - (better a bad format than a bad value)

```
#include <iomanip>
    cout << 123456 <<'|'<< setw(4) << 123456 << '|'
         << setw(8) << 123456 << '|' << 123456 << "|\n";
    cout << 1234.56 <<'|'<< setw(4) << 1234.56 << '|'
         << setw(8) << 1234.56 << '|' << 1234.56 << "|\n";
    cout << "asdfgh" <<'|'<< setw(4) << "asdfgh" << '|'
         << setw(8) << "asdfgh" << '|' << "asdfgh" << "|\n";
```

- Results

```
123456|123456|  123456|123456|
1234.56|1234.56| 1234.56|1234.56|
asdfgh|asdfgh|  asdfgh|asdfgh|
```

Observation



fx Format flag manipulators (functions)

Independent flags (switch on):

boolalpha	Alphanumerical bool values (function)
showbase	Show numerical base prefixes (function)
showpoint	Show decimal point (function)
showpos	Show positive signs (function)
skipws	Skip whitespaces (function)
unitbuf	Flush buffer after insertions (function)
uppercase	Generate upper-case letters (function)

Independent flags (switch off):

noboolalpha	No alphanumerical bool values (function)
noshowbase	Do not show numerical base prefixes (function)
noshowpoint	Do not show decimal point (function)
noshowpos	Do not show positive signs (function)
noskipws	Do not skip whitespaces (function)
nounitbuf	Do not force flushes after insertions (function)
nouppercase	Do not generate upper case letters (function)

Numerical base format flags ("basefield" flags):

dec	Use decimal base (function)
hex	Use hexadecimal base (function)
oct	Use octal base (function)

Floating-point format flags ("floatfield" flags):

fixed	Use fixed floating-point notation (function)
scientific	Use scientific floating-point notation (function)

Adustment format flags ("adjustfield" flags):

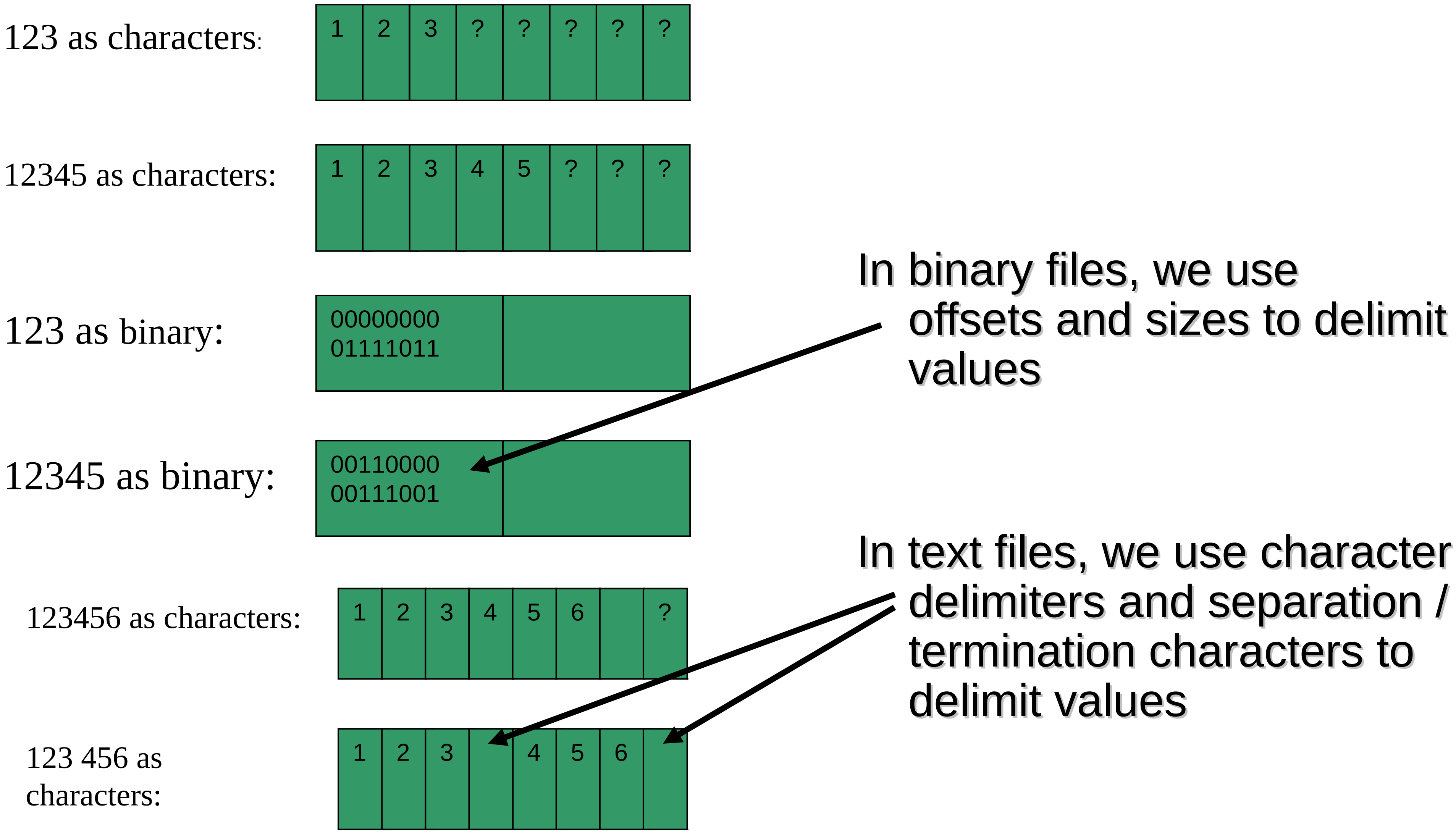
internal	Adjust field by inserting characters at an internal position (function)
left	Adjust output to the left (function)
right	Adjust output to the right (function)

This kind of detail is why you need (online) manuals – try this one:
http://www.cplusplus.com/reference/ios/

File open modes

- By default, an **ifstream** opens its file for reading
- By default, an **ofstream** opens its file for writing
- Alternatives:
  - **ios_base::app** *// append (i.e., output adds to the end of the file)*
  - **ios_base::ate** *// “at end” (open and seek to end)*
  - **ios_base::binary** *// binary mode – beware of system specific behavior*
  - **ios_base::in** *// for reading*
  - **ios_base::out** *// for writing*
  - **ios_base::trunc** *// truncate file to 0-length*
- A file mode is optionally specified after the name of the file:
  - **ofstream of1 {name1};** *// defaults to ios_base::out*
  - **ifstream if1 {name2};** *// defaults to ios_base::in*
  - **ofstream ofs {name, ios_base::app};** *// append rather than overwrite*
  - **fstream fs {"myfile", ios_base::in | ios_base::out};** *// both in and out*

Text vs. binary files



123 as characters:

1	2	3	?	?	?	?	?

12345 as characters:

1	2	3	4	5	?	?	?

123 as binary:

00000000 01111011	

12345 as binary:

00110000 00111001	

123456 as characters:

1	2	3	4	5	6		?

123 456 as characters:

1	2	3		4	5	6	

In binary files, we use offsets and sizes to delimit values

In text files, we use character delimiters and separation / termination characters to delimit values

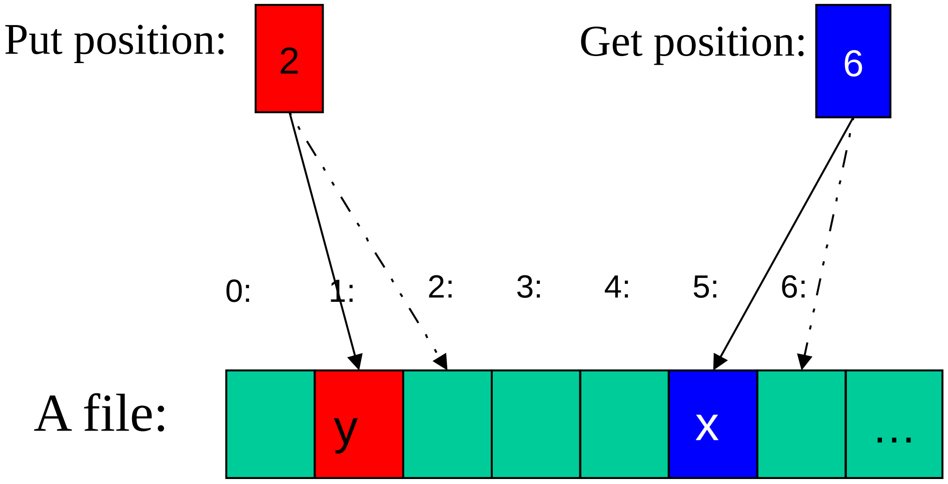
Text vs. binary

- Use text whenever possible
  - You can read it (without a fancy program)
  - You can debug your programs more easily
  - Text is portable across different systems
  - Size (compressed) is typically comparable
  - Most information can be represented reasonably as text
- Use binary when you must
  - E.g. image files, sound files for faster decoding
  - Compressed and / or encrypted files

Binary File I/O

```
int main()   // use binary input and output for C++ 11 and later
{
    cout << "Please enter input file name\n";
    string iname; cin >> iname;
    ifstream ifs {iname,ios_base::binary}; // note: binary
    if (!ifs) error("can't open input file ", iname);

    cout << "Please enter output file name\n";
    string oname; cin >> oname;
    ofstream ofs {oname,ios_base::binary}; // note: binary
    if (!ofs) error("can't open output file ", oname);

    // “binary” tells the stream not to try anything clever with the bytes

    vector<int> v;

    // read from binary file:
    for (int i; ifs.read(as_bytes(i),sizeof(int)); )    // note: reading bytes
        v.push_back(i);

    // … do something with v …

    // write to binary file:
    for(int i=0; i<v.size(); ++i)
        ofs.write(as_bytes(v[i]),sizeof(int)); // note: writing bytes
    return 0;
}
```

Positioning in a filestream



```
fstream fs {name};   // open for input and output (C++ 11 and later)

fs.seekg(5); // move reading position (‘g’ for ‘get’) to 5 (the 6th character)
char ch;
fs>>ch;      // read the x and increment the reading position to 6
cout << "sixth character is " << ch << '(' << int(ch) << ")\n";

fs.seekp(1); // move writing position (‘p’ for ‘put’) to 1 (the 2nd character)
fs<<'y';     // write and increment writing position to 2
```

Positioning

- Whenever you can
  - Use simple streaming
    - Streams/streaming is a very powerful metaphor
    - Write most of your code in terms of “plain” **istream** and **ostream**
    - Default backups for file modifications are fairly easy to implement, e.g., rename the old file with a trailing '~' and write the updated file to the original filename
  - Positioning is far more error-prone
    - Handling of the end of file position is system dependent and basically unchecked
    - A subtle bug can destroy the file being edited

String streams

A **stringstream** (from <sstream>) reads/writes from/to a **string** rather than a file or a keyboard/screen.

This adds all stream capabilities to your string editing arsenal

```
double str_to_double(string s)
        // if possible, convert characters in s to floating-point value
{
    istringstream is {s}; // make a stream so that we can read from s
    double d;
    is >> d;
    if (!is) error("double format error: “,s);
    return d;
}

double d1 = str_to_double("12.4");     // testing
double d2 = str_to_double("1.34e-3");
double d3 = str_to_double("twelve point three");    // will call error()
```

http://www.cplusplus.com/reference/sstream/stringstream/

String streams

- See textbook, cplusplus.com, or Stack Overflow for **ostringstream**
- String streams are very useful for
  - formatting into a fixed-sized space (think GUI)
  - for extracting typed objects out of a string

Type vs. line

- Read a whitespace-terminated string

```
string name;
cin >> name;      // input: Dennis Ritchie
cout << name << '\n'; // output: Dennis
```

- Read a line

```
string name;
getline(cin, name);   // input: Dennis Ritchie
cout << name << '\n'; // output: Dennis Ritchie

// now what? Maybe:

istringstream ss(name);
ss >> first_name;
ss >> second_name;
```

Characters

- You can also read individual characters

```
for (char ch; cin>>ch; ) {    // read into ch, skip whitespace characters
    if (isalpha(ch)) {
             // do something
    }
}

for (char ch; cin.get(ch); ) {    // read into ch, handle whitespace separately
    if (isspace(ch)) {
             // do something
    }
    else if (isalpha(ch)) {
             // do something else
    }
}
```

Character classification functions

- If you use character input, you often need one or more of these (from header **<cctype>**):

– **isspace(c)**	*// is **c** whitespace? (' ', '\t', '\n', etc.)*
– **isalpha(c)**	*// is **c** a letter? ('a'..'z', 'A'..'Z') note: not '_'*
– **isdigit(c)**	*// is **c** a decimal digit? ('0'..'9')*
– **isupper(c)**	*// is **c** an upper case letter?*
– **islower(c)**	*// is **c** a lower case letter?*
– **isalnum(c)**	*// is **c** a letter or a decimal digit?*

etc.

http://cplusplus.com/reference/cctype/

Line-oriented input

- Prefer **>>** to **getline()**
  - i.e. avoid line-oriented input when you can

- People often use **getline()** because they see no alternative
  - But it easily gets messy
  - When trying to use **getline()**, you often end up
    - using **>>** to parse the line from a **stringstream**
    - using **get()** to read individual characters

```
int a, b;
while (infile >> a >> b)
{
    // process pair (a,b)
}
```

```
std::string line;
while (std::getline(infile, line))
{
    std::istringstream iss(line);
    int a, b;
    if (!(iss >> a >> b)) { break; } // error

    // process pair (a,b)
}
```

Code from http://stackoverflow.com/questions/7868936/read-file-line-by-line
Used under “non-commercial use” clause of the license agreement.

New C++14 Literals

- Binary literals
  - **0b1010100100000011**
- Digit separators
  - **0b1010'1001'0000'0011**
  - Can also be used for for decimal, octal, and hexadecimal numbers
- User-Defined Literals (UDLs) in the standard library
  - Time: **2h+10m+12s+123ms+3456ns**
  - Complex: **2+4i**

Behavioral

Strategy Pattern

- The **Strategy** pattern (sometimes called the Policy Pattern) enables an algorithm's behavior to be modified at runtime
  - Provides a common interface to multiple methods
  - Dynamically selects between methods based on a specific criteria

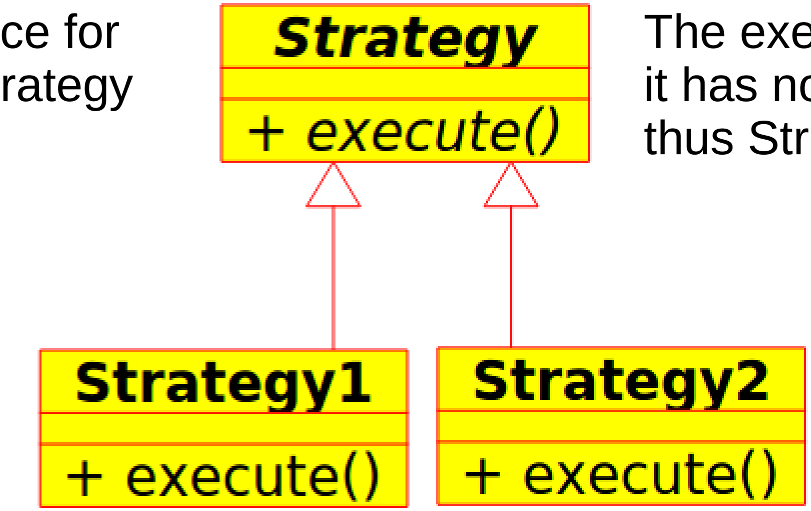
- For example, a security package may use the Strategy pattern to select different levels of file scanning for malware, depending on the file's source

Behavioral

The Strategy Pattern

(Slightly Simplified)

The interface for executing a strategy



The execute() method is classically *pure* virtual;
it has no implementation,
thus Strategy cannot be instanced.

One strategy

Another strategy

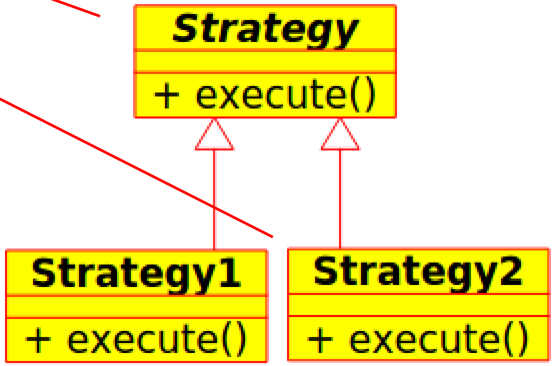
Behavioral

The Strategy Pattern

(Slightly Simplified)

```
class RobotBillingStrategy {
 public:
  virtual double getPrice(double listPrice) = 0;
};
class FullPrice : public RobotBillingStrategy {
 public:
  double getPrice(double listPrice) override {
    return listPrice;
  }
};
class HalfPrice : public RobotBillingStrategy {
 public:
  double getPrice(double listPrice) override {
    return listPrice * 0.5;
  }
};
class Customer {
 public:
  Customer(bool newCustomer) {
    if (newCustomer) strategy = new HalfPrice;
    else strategy = new FullPrice;
  }
  double getBill(double productCost) {
    return strategy->getPrice(productCost);
  }
 private:
  RobotBillingStrategy *strategy;
};
```

This makes the method *pure* virtual



Polymorphism!

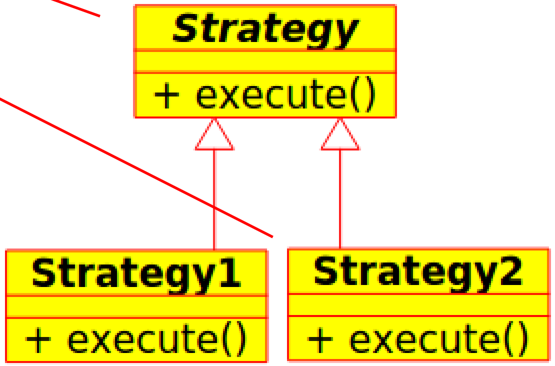
Behavioral

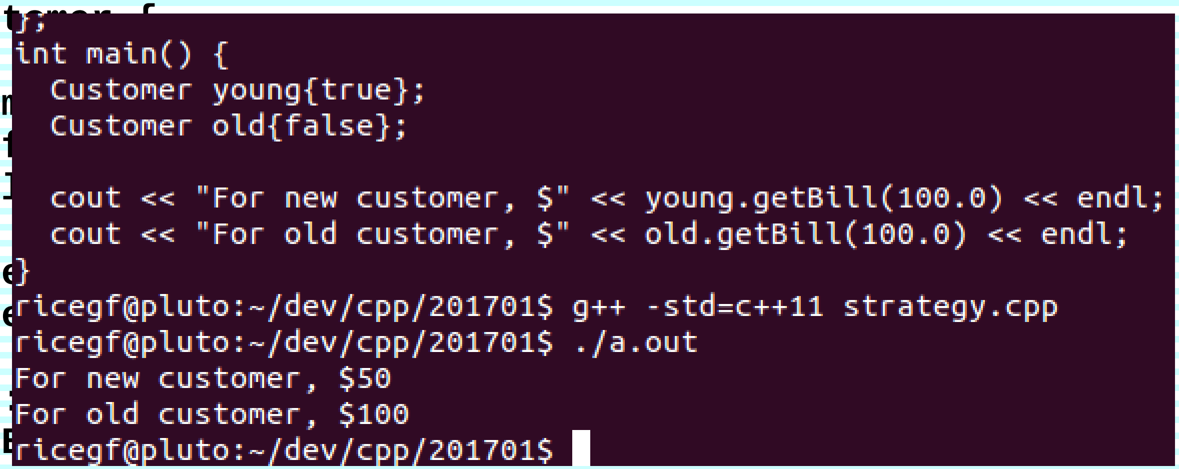
The Strategy Pattern

(Slightly Simplified)

```
class RobotBillingStrategy {
  public:
    virtual double getPrice(double listPrice) = 0;
};
class FullPrice : public RobotBillingStrategy {
  public:
    double getPrice(double listPrice) override {
        return listPrice;
    }
};
class HalfPrice : public RobotBillingStrategy {
  public:
    double getPrice(double listPrice) override {
        return listPrice * 0.5;
    }
};
class Cust
  public:
    Custon
        if
        el
    }
    double
        re
    }
  private:
    RobotB
};
```

This makes the method *pure* virtual





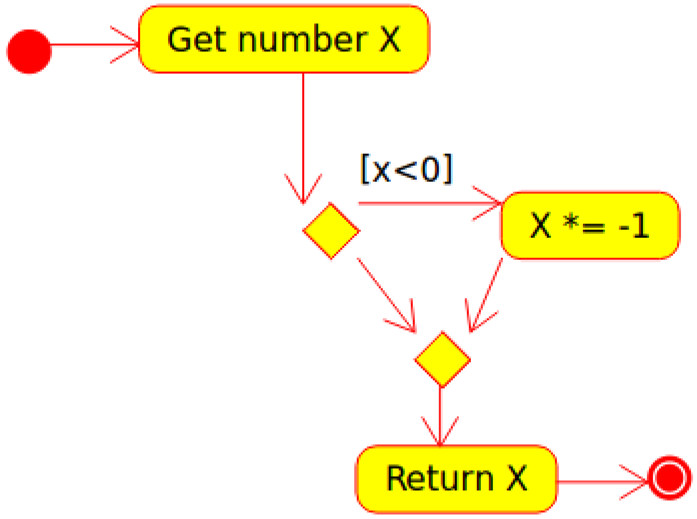
```
};
int main() {
  Customer young{true};
  Customer old{false};

  cout << "For new customer, $" << young.getBill(100.0) << endl;
  cout << "For old customer, $" << old.getBill(100.0) << endl;
}
ricegf@pluto:~/dev/cpp/201701$ g++ -std=c++11 strategy.cpp
ricegf@pluto:~/dev/cpp/201701$ ./a.out
For new customer, $50
For old customer, $100
ricegf@pluto:~/dev/cpp/201701$
```

UML Activity Diagram

- The UML Activity diagram displays a sequence of activities at the algorithm level
  - Similar to a classic “flow chart” or “data flow diagram”
- Represents decisions as well as concurrency
  - Supports decision points – take only one path
  - Supports forks and joins – take all paths, and later sync back up
- Supports hierarchies
  - An activity can contain another Activity Diagram

Trivial Example Activity Diagram

Absolute Value



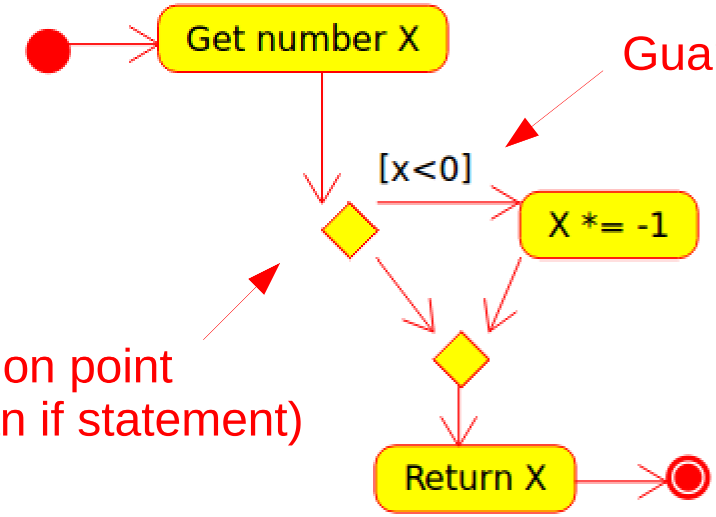
http://www.uml-diagrams.org/activity-diagrams.html

Trivial Example Activity Diagram

Activity to perform

Always start here!
(Must be exactly 1)

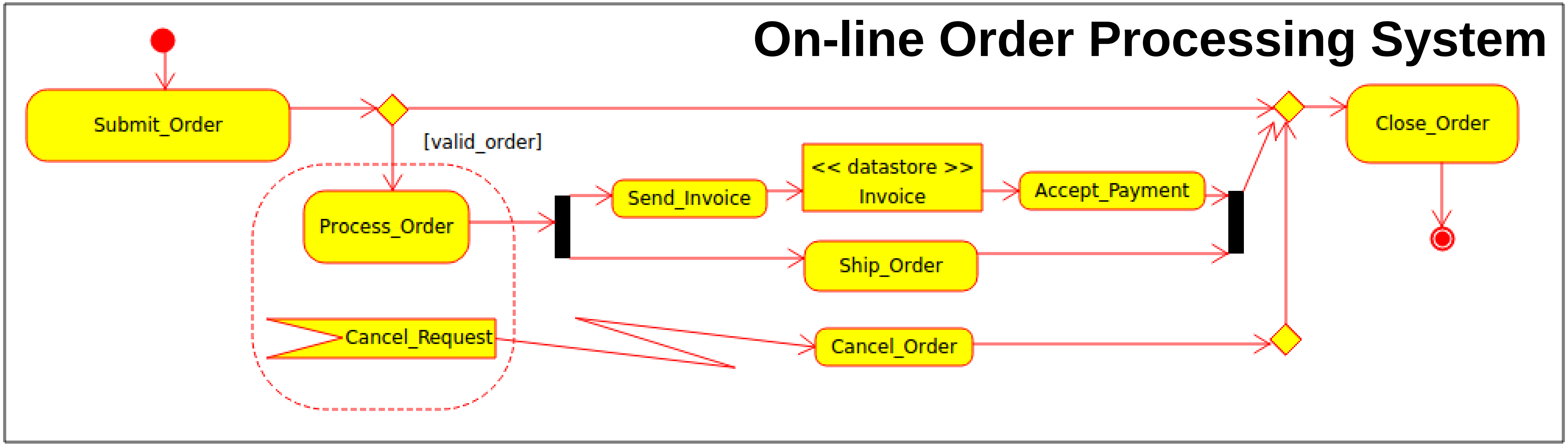
Guard (conditional)



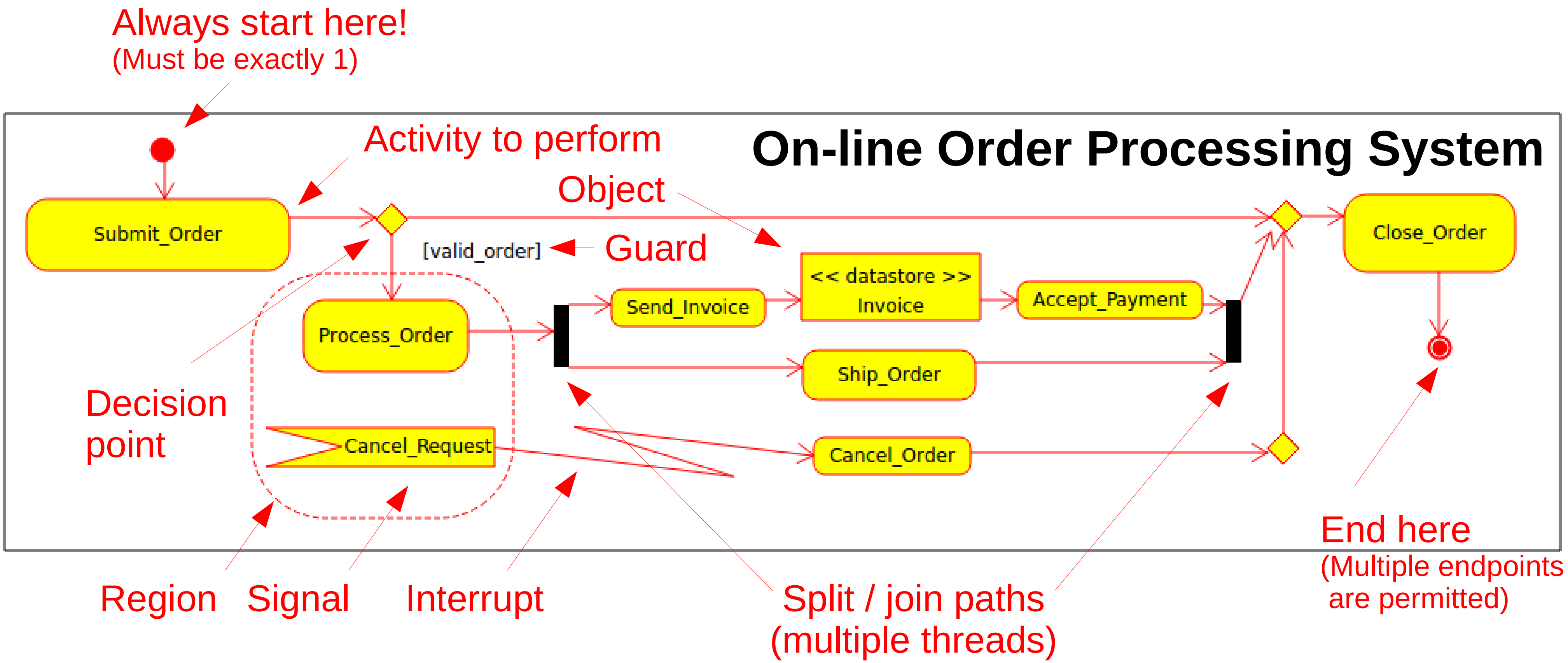
Decision point
(like an if statement)

End here
(Multiple endpoints
are permitted)

Example Activity Diagram



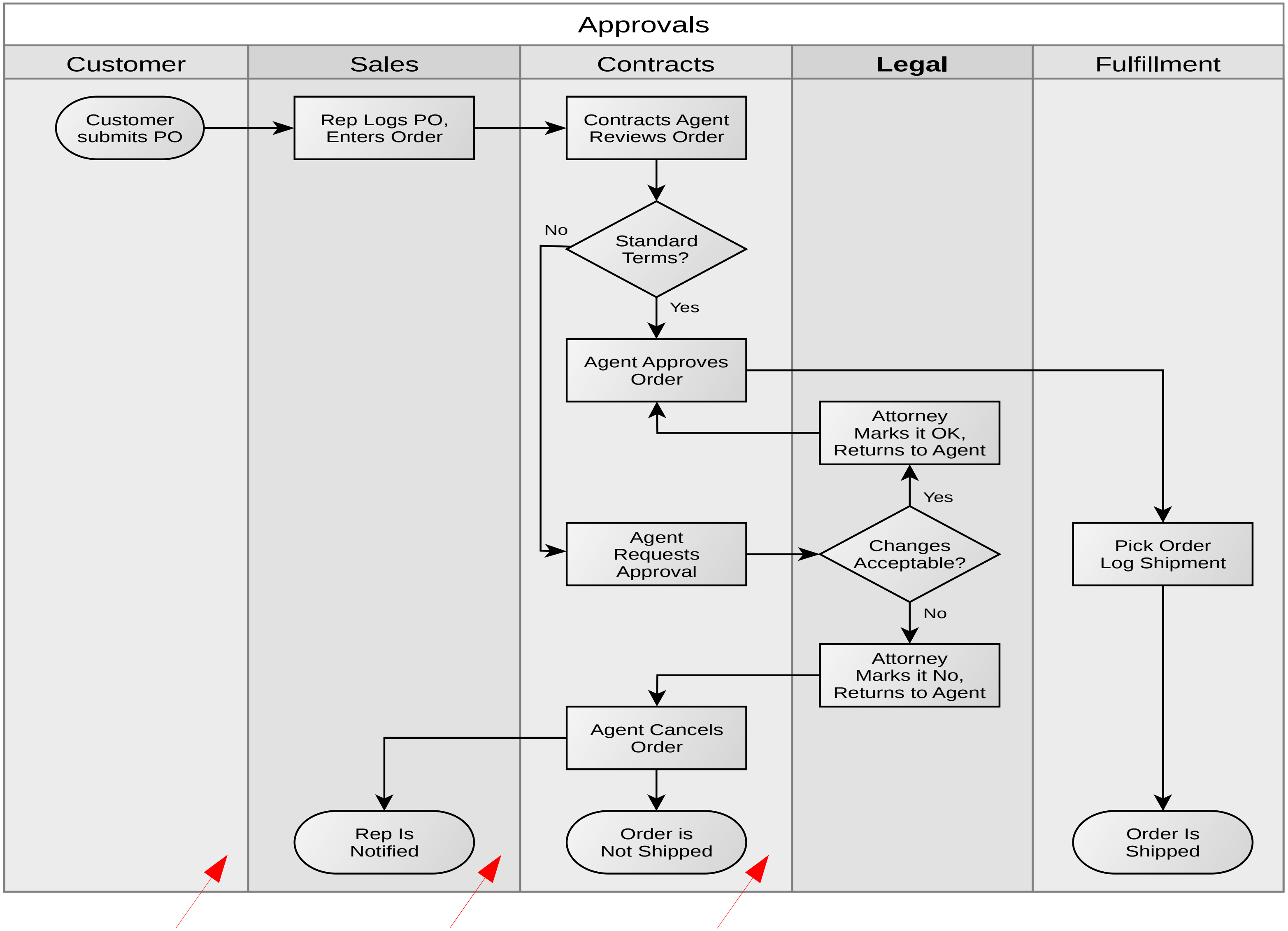
Example Activity Diagram



http://www.uml-diagrams.org/activity-diagrams.html

Swimlanes

(Sometimes Called Partitions)



Swimlanes assign responsibility for activities

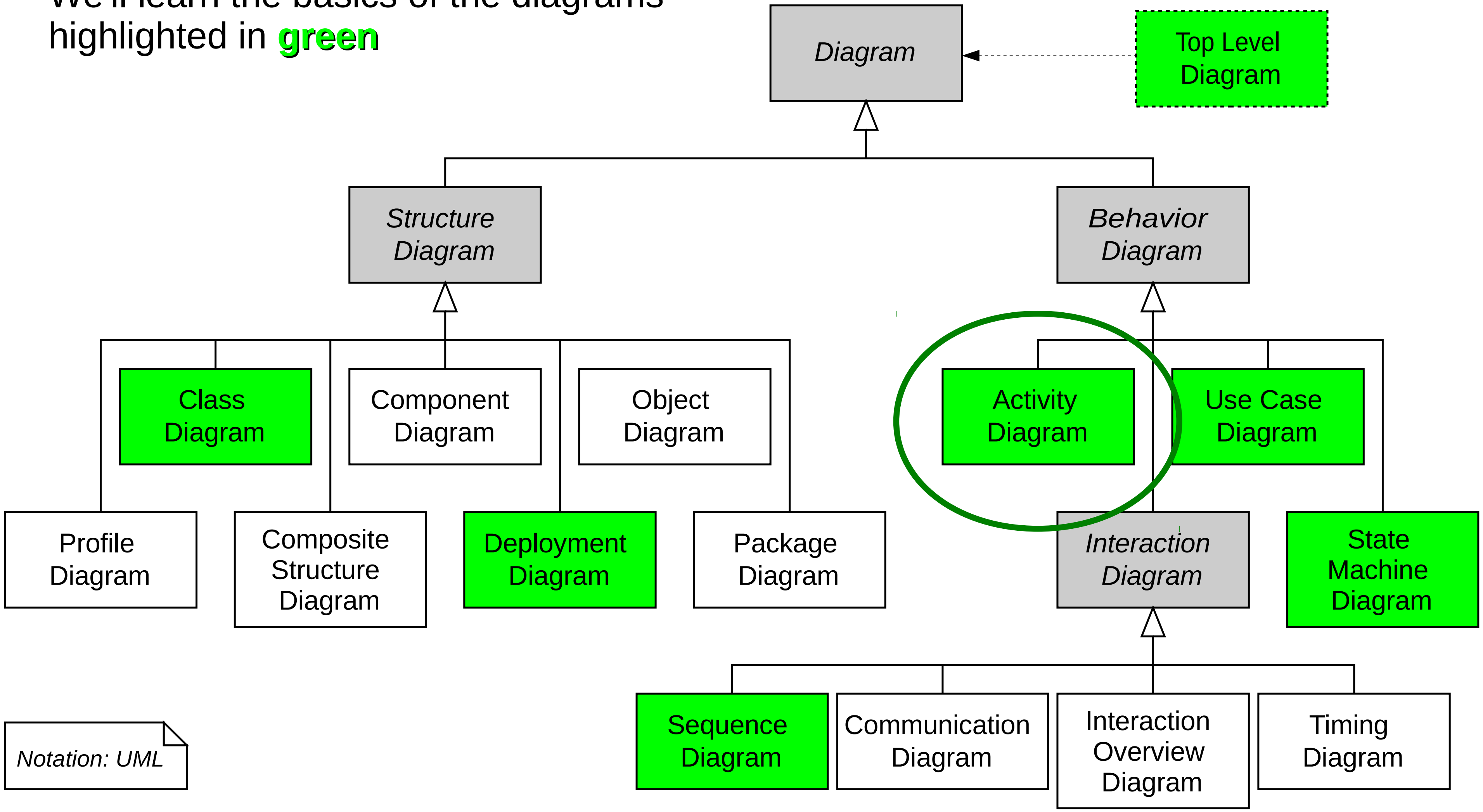
By Paul Kerr - http://commons.wikimedia.org/wiki/File:Approvals.jpg, CC0, https://commons.wikimedia.org/w/index.php?curid=21550293

Activity vs Sequence

- Activity Diagrams focus on actions (flow of control between methods or packages)
  - Depicts the entire algorithm
  - Time is not represented
- Sequence Diagrams focus on interactions (communication between objects)
  - Depicts one possible thread through the algorithm
  - Time flows downward

The Activity Diagram in Context

We'll learn the basics of the diagrams highlighted in **green**



Original source: Wikipedia, Public Domain SVG

Quick Review

- True or False: Most (but not all) stream operators are “sticky”. If False, which is it? If True, give an example of each.
- True or False: Stream operators are constants and thus cannot accept parameters.
- To output hexadecimal numbers via cout, include the ___ operator in the stream. To precede hexadecimal numbers with “0x”, include _______ in the stream.
- What happens if a value exceeds the specified stream output width?
- True or False: Binary file operations are inherently less portable than text file operations.
- Stream operations can target string variables by using the _________ class.
- The _________ pattern enables an algorithm’s behavior to be modified at runtime.

Next Week

- Review chapter 11 in Stroustrop
  - Do the drills!
- Thursday is Embedded Programming, with UML Statechart diagrams, implementation, and the State Design Pattern
- Sprint #2 (Homework #8) is due Thursday, November 2 at 8 am
  - Teams that have begun GUI implementation – screenshots required of each GUI window or dialog!