CSE 1325: Object-Oriented Programming

Lecture 23 / Chapters 23-24

Text Processing, Numerics, and the Strategy Pattern

Mr. George F. Rice
george.rice@uta.edu

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

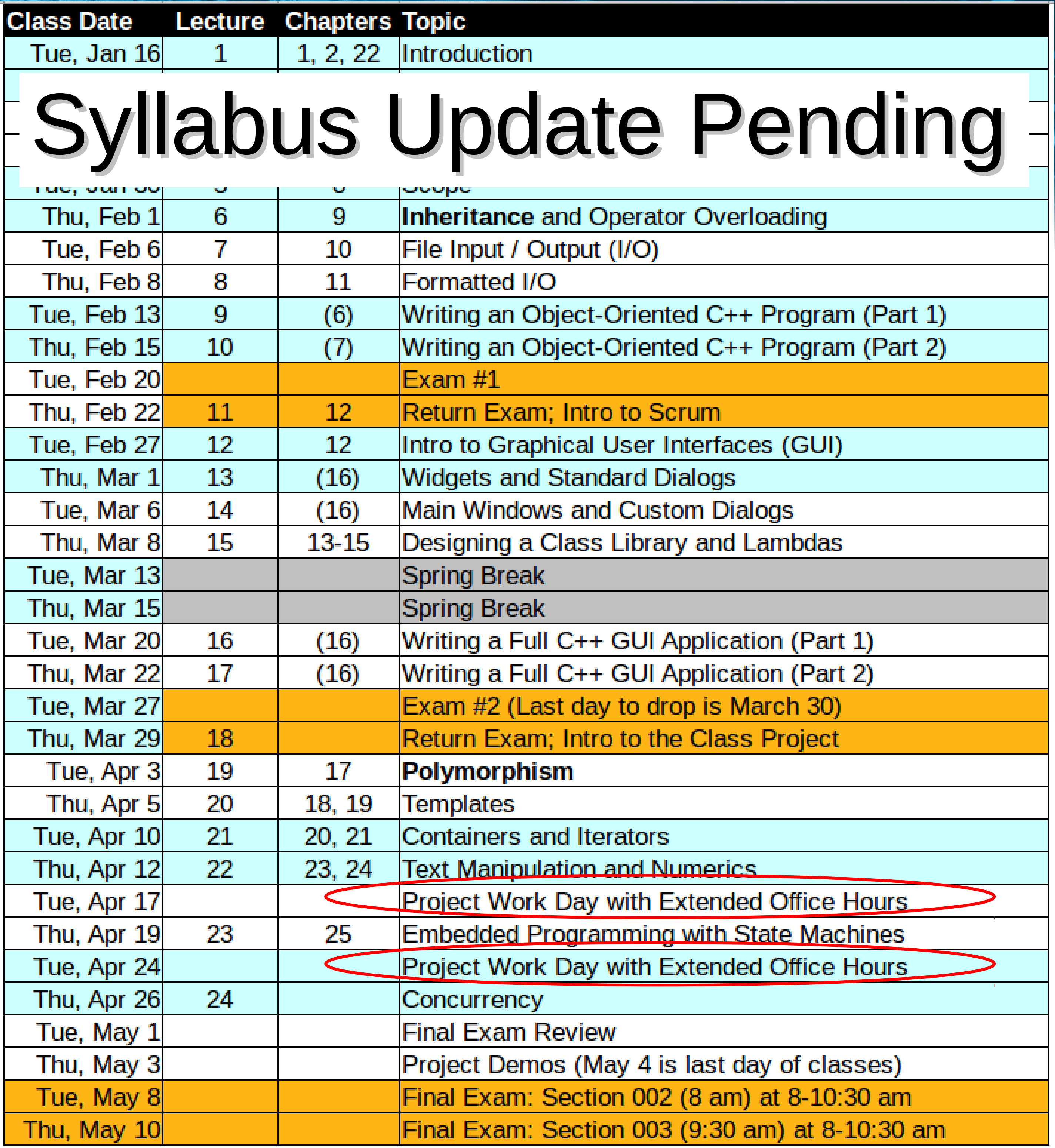
ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Quick Review

- The four basic types of STL containers in C++ are
  (a) Memory Managed Containers (b) Unmanaged Containers
  (c) **Sequence Containers** (d) **Sequence Adapters**
  (e) **Associative Containers** (f) Associative Adapters
  (g) **Unordered Associative Containers**
- **Sequence Adapters** are a façade for **Sequence Containers**.
- A(n) **iterator** is a pointer-like instance of a nested class used to access items managed by the outer class instance/
- **True** or False: With iterators, the end of the sequence is “one past the last element”, not “the last element”
- Two types of iterators are typically provided, **iterator** and **const iterator**.
- Name some common iterator and container methods in the STL.
  Iterator: dereferencing, operator++ and --, operator=, operator[ ], math and logic
  Container: begin, end, front, back, size, empty, push/pop_back/front, insert, erase
- True or **False**: It is a good practice to derive your classes from STL classes.
- What is a copy constructor? **Constructs object from another instance of the same class**
  Copy assignment operator? **Operator= that opies elements from rhs to lhs**
- Define the Rule of 3. **If you need a destructor, copy constructor, or copy assignment operator for your class, you almost certainly need all 3**

A Note on Grades

- **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.
- The deadline to file a grade appeal is 2 weeks from the day the grade is posted



Syllabus Update Pending

Class Date	Lecture	Chapters	Topic
Tue, Jan 16	1	1, 2, 22	Introduction
Tue, Jan 30	5	8	Scope
Thu, Feb 1	6	9	**Inheritance** and Operator Overloading
Tue, Feb 6	7	10	File Input / Output (I/O)
Thu, Feb 8	8	11	Formatted I/O
Tue, Feb 13	9	(6)	Writing an Object-Oriented C++ Program (Part 1)
Thu, Feb 15	10	(7)	Writing an Object-Oriented C++ Program (Part 2)
Tue, Feb 20			Exam #1
Thu, Feb 22	11	12	Return Exam; Intro to Scrum
Tue, Feb 27	12	12	Intro to Graphical User Interfaces (GUI)
Thu, Mar 1	13	(16)	Widgets and Standard Dialogs
Tue, Mar 6	14	(16)	Main Windows and Custom Dialogs
Thu, Mar 8	15	13-15	Designing a Class Library and Lambdas
Tue, Mar 13			Spring Break
Thu, Mar 15			Spring Break
Tue, Mar 20	16	(16)	Writing a Full C++ GUI Application (Part 1)
Thu, Mar 22	17	(16)	Writing a Full C++ GUI Application (Part 2)
Tue, Mar 27			Exam #2 (Last day to drop is March 30)
Thu, Mar 29	18		Return Exam; Intro to the Class Project
Tue, Apr 3	19	17	**Polymorphism**
Thu, Apr 5	20	18, 19	Templates
Tue, Apr 10	21	20, 21	Containers and Iterators
Thu, Apr 12	22	23, 24	Text Manipulation and Numerics
Tue, Apr 17			Project Work Day with Extended Office Hours
Thu, Apr 19	23	25	Embedded Programming with State Machines
Tue, Apr 24			Project Work Day with Extended Office Hours
Thu, Apr 26	24		Concurrency
Tue, May 1			Final Exam Review
Thu, May 3			Project Demos (May 4 is last day of classes)
Tue, May 8			Final Exam: Section 002 (8 am) at 8-10:30 am
Thu, May 10			Final Exam: Section 003 (9:30 am) at 8-10:30 am

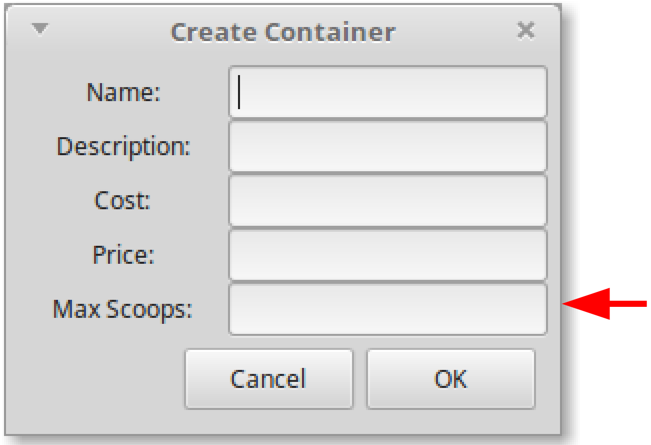
Overview: Strings, Maps, and the Strategy Pattern

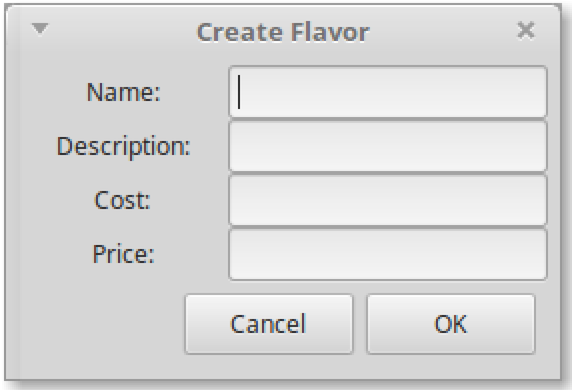
- Debugging Exercise
- Agile Estimating
- Text Processing
  - Class to/from strings
  - Regular expressions
- Numerics
  - Maps
- Strategy Pattern

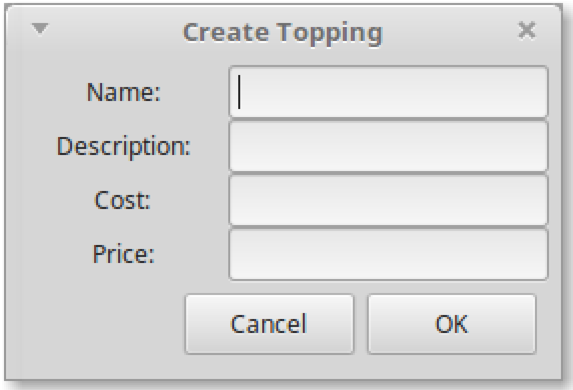
https://www.publicdomainpictures.net/en/view-image.php?image=104269&picture=cargo-container-ship

Debugging Exercise
The Goal

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







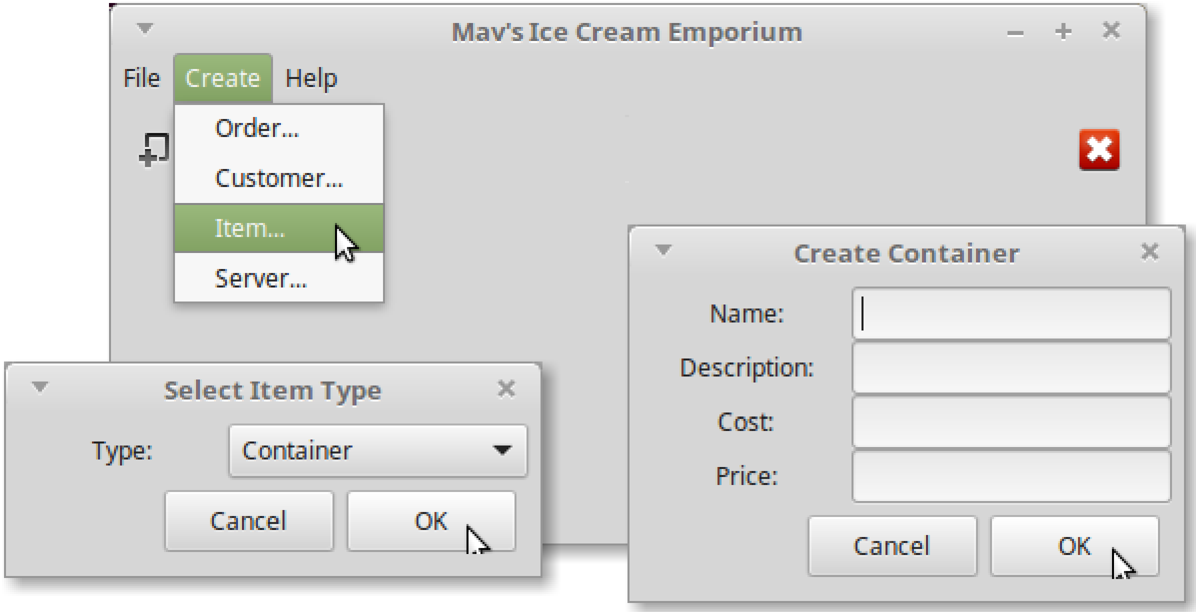
The Code

- 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);
    }
```

The Bug

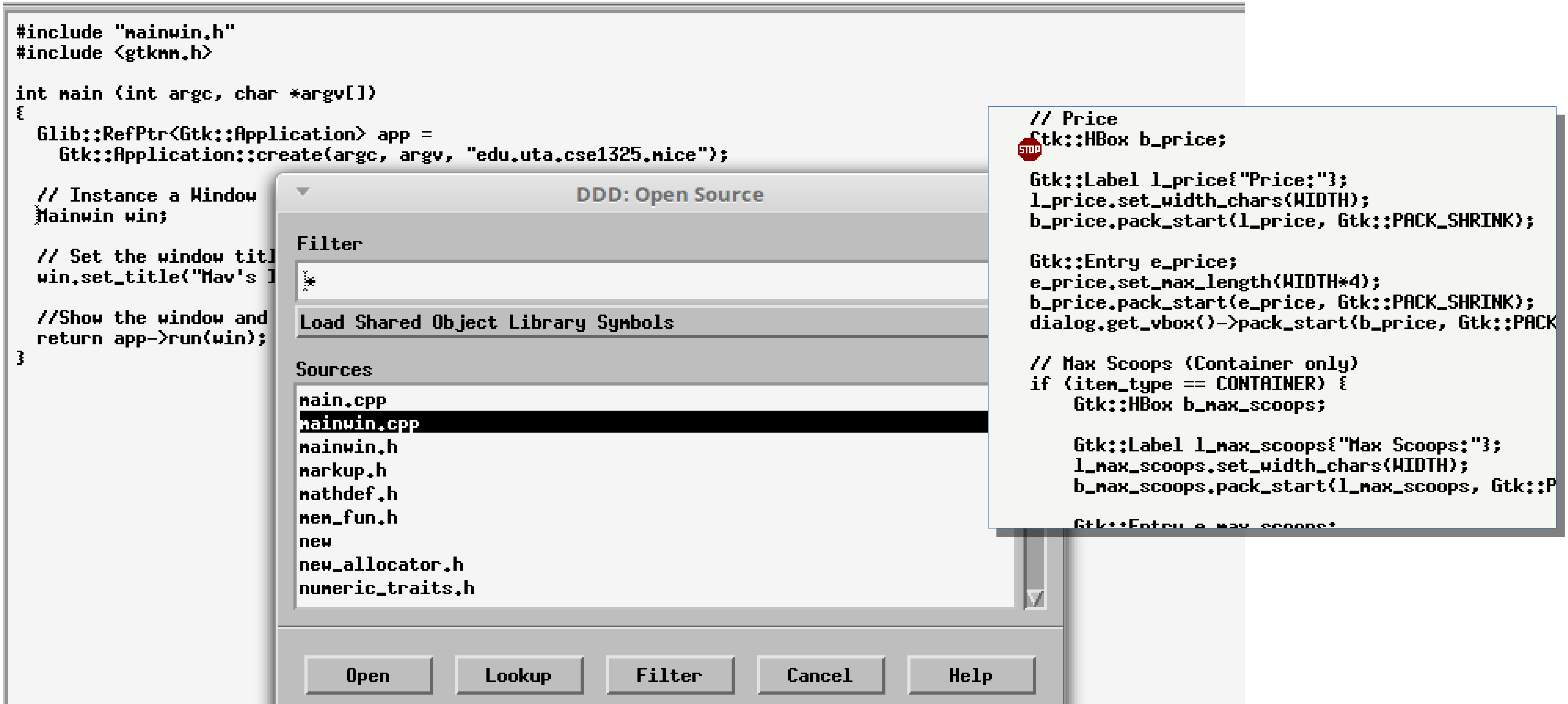
- Wait – where’s the entry for Max Scoops???
- How would you debug this problem?



Type: Container

The Debugger

- 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);
}
```

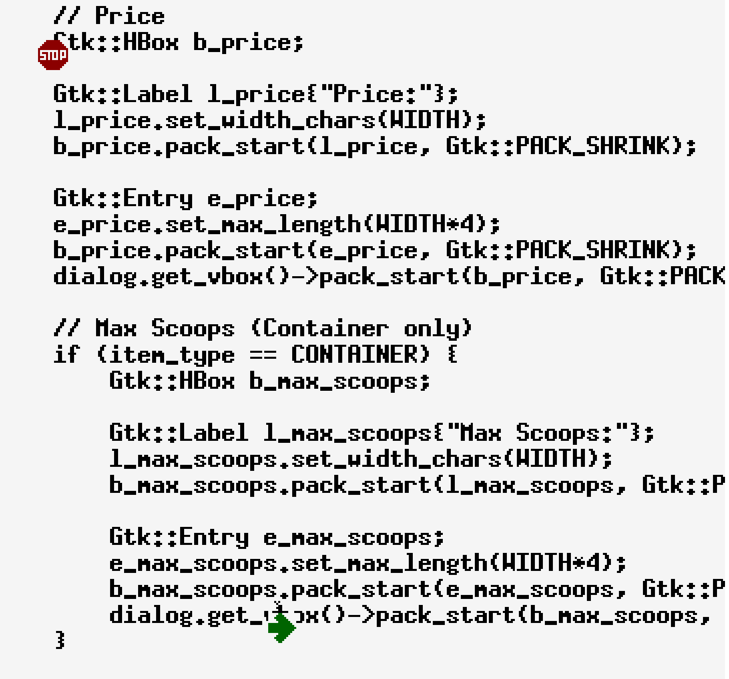
Filter: `*`

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

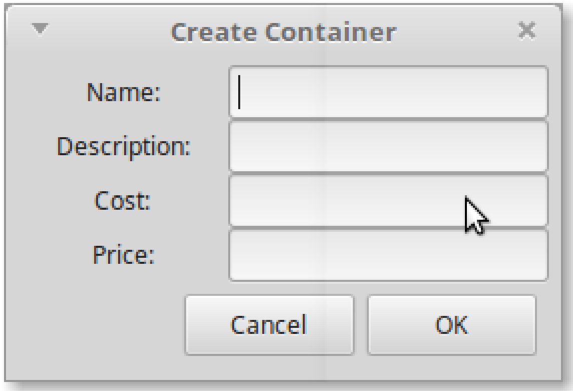
```
    // Price
    Gtk::HBox b_price;

    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;
```

The Conundrum

- Wait... what?!?!?
  - The code for the 5th entry *is being executed*!
  - But the 5th entry is still not displayed

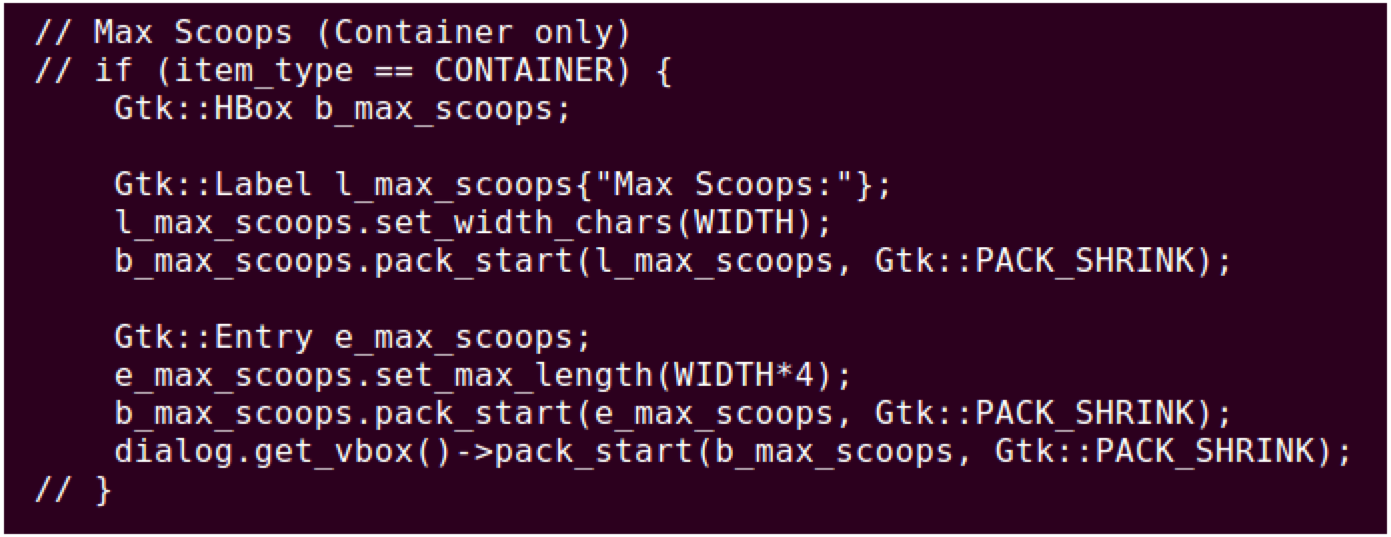


```
// Price
Gtk::HBox b_price;

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,
}
```

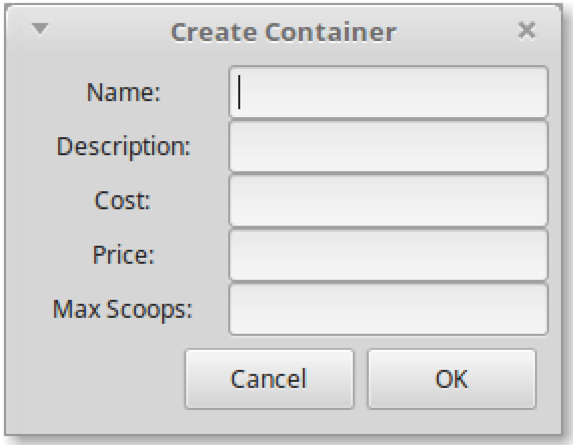


The Experiment

- 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”



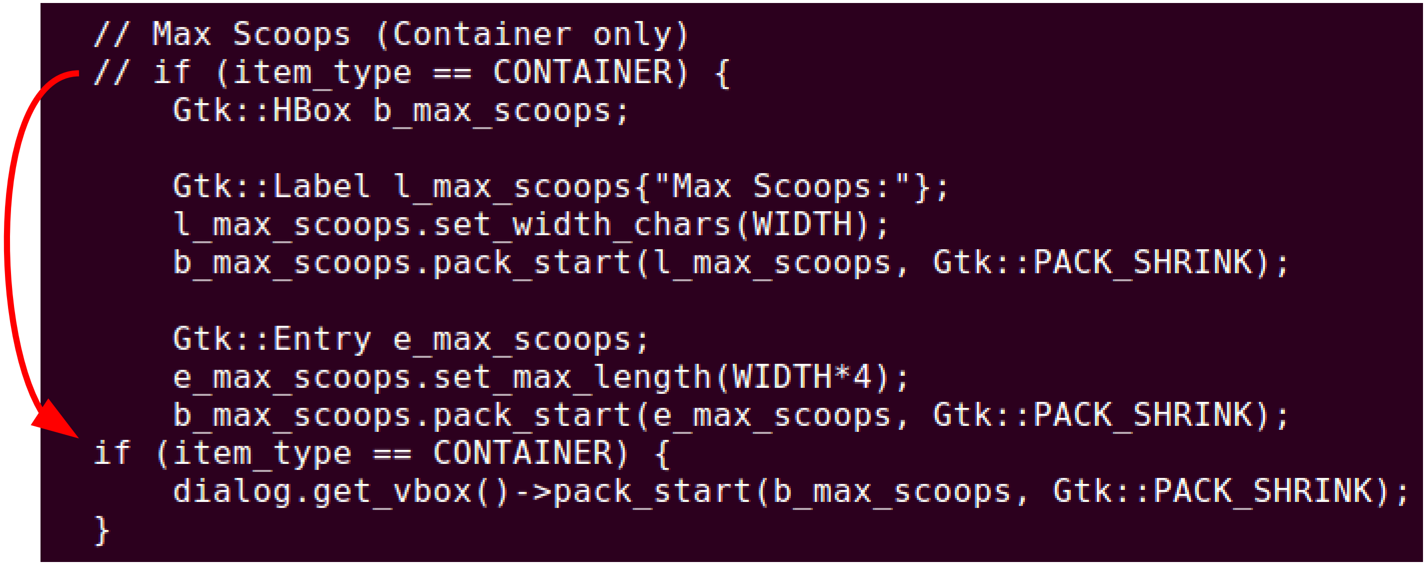
```
// Max Scoops (Container only)
// if (item_type == CONTAINER) {
    Gtk::HBox b_max_scoops;

    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);
// }
```



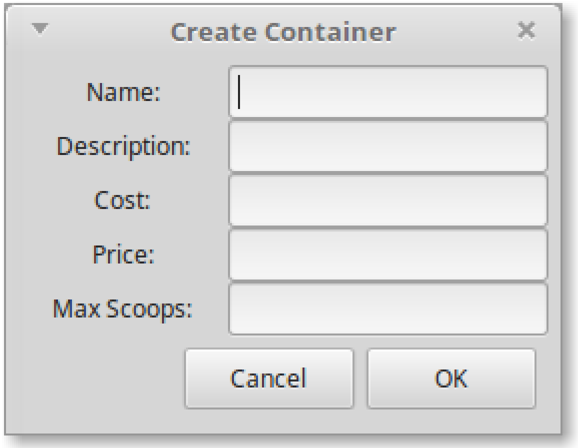
Noooooooooo...
Now what?

The Result

- The code works if NOT inside the if
- The code doesn’t work if it IS inside the if
- Now what? Let’s move the if down as far as practical
  - b_max_scoops is <u>always</u> created,
    but only added if the dialog is for a container



```
// 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?

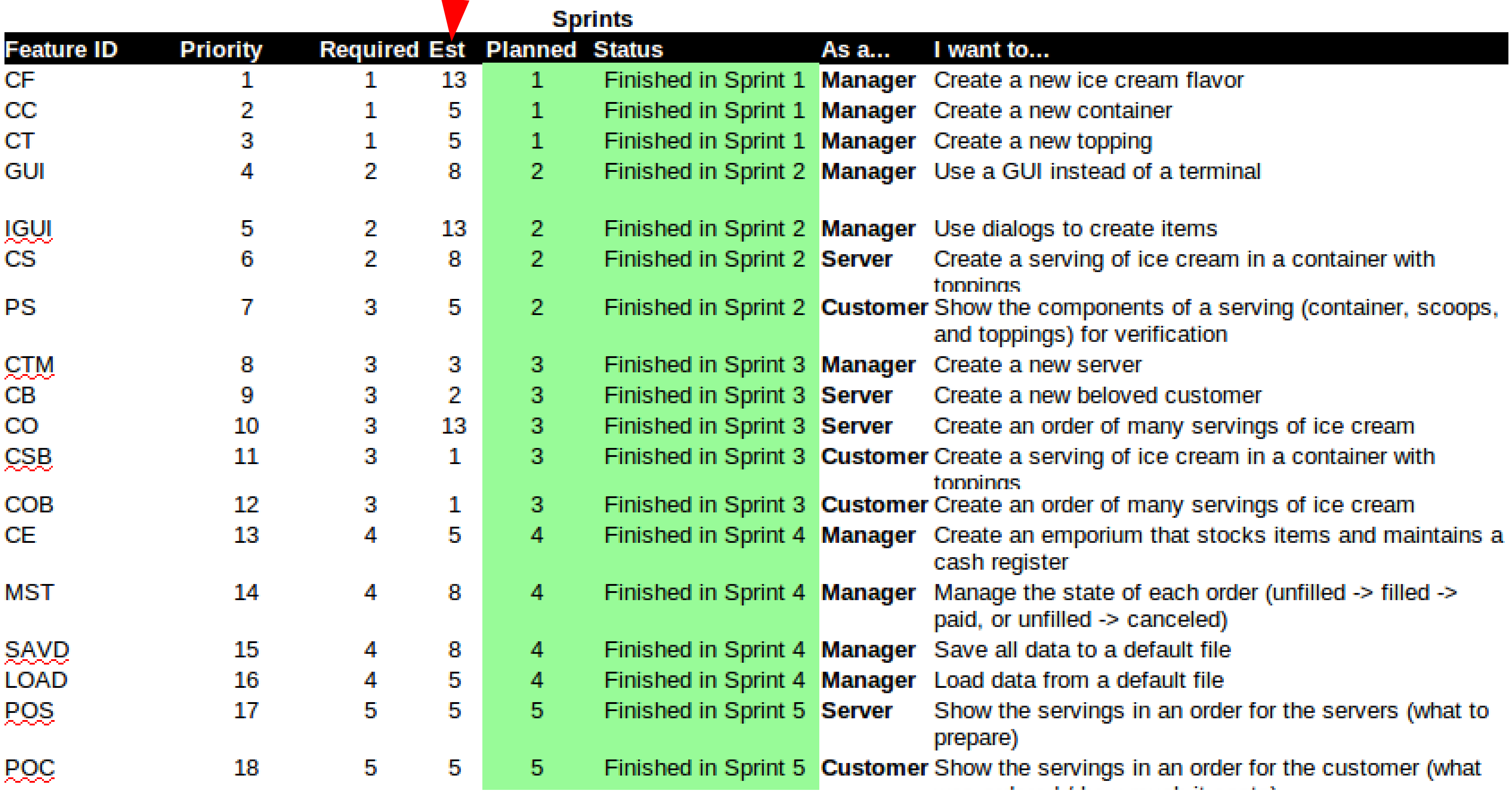
The Bug

- It’s a scope problem – b_max_scoop is being popped from the stack *before the dialog is displayed*

```
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) {  // CREATES A NEW LOCAL (IF) SCOPE!
        Gtk::HBox b_max_scoops;    // CREATES THE SCOOP BOX IN LOCAL SCOPE!

        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);

    }                              // EXITS LOCAL SCOPE!
```

Agile Estimating

How were the “Estimates” created?



				Sprints			
Feature ID	**Priority**	**Required**	**Est**	**Planned**	**Status**	**As a...**	**I want to...**
CF	1	1	13	1	Finished in Sprint 1	**Manager**	Create a new ice cream flavor
CC	2	1	5	1	Finished in Sprint 1	**Manager**	Create a new container
CT	3	1	5	1	Finished in Sprint 1	**Manager**	Create a new topping
GUI	4	2	8	2	Finished in Sprint 2	**Manager**	Use a GUI instead of a terminal
IGUI	5	2	13	2	Finished in Sprint 2	**Manager**	Use dialogs to create items
CS	6	2	8	2	Finished in Sprint 2	**Server**	Create a serving of ice cream in a container with toppings
PS	7	3	5	2	Finished in Sprint 2	**Customer**	Show the components of a serving (container, scoops, and toppings) for verification
CTM	8	3	3	3	Finished in Sprint 3	**Manager**	Create a new server
CB	9	3	2	3	Finished in Sprint 3	**Server**	Create a new beloved customer
CO	10	3	13	3	Finished in Sprint 3	**Server**	Create an order of many servings of ice cream
CSB	11	3	1	3	Finished in Sprint 3	**Customer**	Create a serving of ice cream in a container with toppings
COB	12	3	1	3	Finished in Sprint 3	**Customer**	Create an order of many servings of ice cream
CE	13	4	5	4	Finished in Sprint 4	**Manager**	Create an emporium that stocks items and maintains a cash register
MST	14	4	8	4	Finished in Sprint 4	**Manager**	Manage the state of each order (unfilled -> filled -> paid, or unfilled -> canceled)
SAVD	15	4	8	4	Finished in Sprint 4	**Manager**	Save all data to a default file
LOAD	16	4	5	4	Finished in Sprint 4	**Manager**	Load data from a default file
POS	17	5	5	5	Finished in Sprint 5	**Server**	Show the servings in an order for the servers (what to prepare)
POC	18	5	5	5	Finished in Sprint 5	**Customer**	Show the servings in an order for the customer (what

Strings Matter

- All data can be represented by text
  - Books, articles, web pages
  - Tables of structured information (e.g., XML, JSON)
  - Email, SMS, social media
  - Graphics (e.g., vector formats)
  - Software code(!)
  - Binary data (e.g., uuencode, uudecode)
  - All languages and way too many emojis
- Text is very portable (except that annoying \n, \r, \r\n thingie)
- Text is easily created and edited by your choice of text editor

Options for Representing Strings

- Old C-style strings (zero-terminated char array)
  - #include <cstring> (or <string.h>)
  - strlen(s)
  - strcmp(s1, s2)
- Std::string (a class!)
  - #include <string>
  - s.size()
  - s1 == s2
- Std::basic_string (a template!!!)
  - A template for making string-like classes
  - Std::string is simply **`typedef std::basic_string<char> string;`**
  - Std::basic_string<wchar_t> is the same, for 16-bit chars, etc.
- Proprietary types, e.g., gtkmm’s Glib::ustring (good for Unicode!)

Translating Objects to Strings

- string pi = to_string(3.14159265)
- to_string can be instanced from a simple template
  for any class

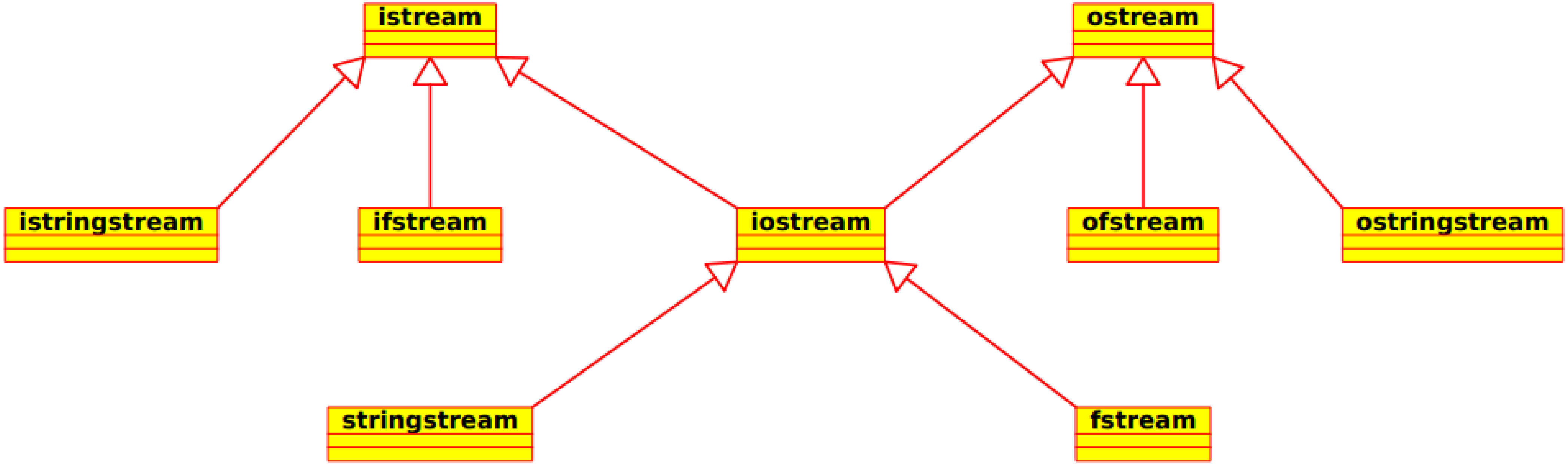
```
template<class T>
string to_string(const T& t) {
    ostringstream os;
    os << t;
    return os.str();
}
class Coordinate {
  public:
    Coordinate(int x, int y) : _x{x}, _y{y} { }
    int x() const {return _x;}
    int y() const {return _y;}
  private:
    int _x, _y;
};
ostream& operator<<(ostream& os, const Coordinate& c) {
    os << "(" << c.x() << "," << c.y() << ")";
    return os;
}
int main() {
    Coordinate c{3,4};
    string s = to_string<Coordinate>(c);
    cout << s << endl;
}
```

Translating Strings to Objects

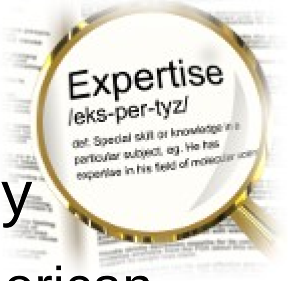
- stoX (where X is a primitive)
  - int i = stoi(“42”);
  - double e = stod(“2.71828”);
- A template again can be instanced to take care of classes

```
template<class T>
T stox(const string& s) {
    istringstream is(s);
    T t;
    if (!(is >> t)) throw bad_from_string{};
    return t;
}
// …
    Coordinate c = stox<Coordinate>(“(3,4)”);
```

String I/O (Partial) Class Hierarchy



Regular Expressions


Expertise
/eks-per-tyz/
def. Special skill or knowledge in a particular subject, eg. He has expertise in his field of molecular science

- A **regular expression** (regex) is a string that defines a search or search-and-replace pattern for other strings
- Two standards exist, and C++ supports both simultaneously
  - The Perl standard dates back to the 1950s, first proposed by American mathematician Stephen Cole Kleene, implemented in 1968 in the original Unix text editor QED, and enhanced in 1986 for Perl
  - The POSIX standard with more verbose but flexible syntax was adopted in 1992
- A regex is often an excellent choice for text manipulation, command or data parsing, or input validation
  - Regex are built into Perl and Javascript, and available as standard classes in C++, C#, Java, and Python. They are not part of ANSI C.
- The regex syntax is terse, cryptic, and borderline write-only, yet exceeding useful – learn it!

Special vs Non-Special Characters

- A non-special character matches itself
- A special character will match 0 or more characters
  - For example, . will match any one character, while .+ will match one or more characters
  - \s is a special character that matches a space or tab, while \s* matches zero or more spaces and tabs
  - A special character preceded by a \ matches itself, so \. matches a period and \\ matches a single backslash
  - In C++ strings (other than raw strings), another \ is required to escape the \ in the regular expression
    - So \s* would be "\\s*" in a C++ string

Special Characters

- . matches any single character except a newline or carriage return
- [ ] matches any single character inside the brackets
  - [^] matches any character NOT inside the brackets
  - [a-z] matches any characters in the range a to z
  - [cat|dog] matches cat OR dog
- {m,n} matches the previous element at least m but no more than n times
  - * matches the previous element 0 or more times (same as {0,})
  - ? matches the previous element 0 or 1 times (same as {0,1})
  - + matches the previous element 1 or more times (same as {1,})
- ^ matches the start and $ the end of a line
- ( ) forms a sub-match for later reference
  - \N matches the Nth sub-match's actual text (start at 1, NOT 0!)

Character Matching Examples

- a.c matches abc, acc, a/c
- r[au]n matches ran or run
- [^2-9][0-9] matches 17 or 03 or K9, but not 21 or 99
- 3\.3* matches 3., 3.3, 3.33, or 3.3333333333
- ([0-9]+) = \1 matches 3 = 3 and 42 = 42, but not 21 = 39
- ^[hH]ello matches hello or Hello, but only at the start of a line
- Sincerely(yours)?,$ matches Sincerely, or Sincerely yours, but only at the end of a line
- Subject: (F[Ww]:|R[Ee]:)?(.*) matches an email subject line for forwards and replies only
- [a-zA-Z] [a-zA-Z_0-9]* matches a C++ identifier

Character Classes

(C++ supports both syntax families)

- Unix / Linux traditional syntax
  - \s matches a whitespace char
    - \S anything else
  - \w matches a word char ([A-Za-z0-9_])
    - \W anything else
  - \d matches a digit ([0-9])
    - \D anything else
  - \x represents a hexadecimal digit
    - \X anything else
  - \n, \r, \t represent newline, carriage return, and tab as usual

- ECMA class syntax
  - [:alnum:]alpha-numerical character
  - [:alpha:] alphabetic character
  - [:blank:] blank character
  - [:cntrl:] control character
  - [:digit:] decimal digit character
  - [:graph:] character with graphical rep
  - [:lower:] lowercase letter
  - [:print:] printable character
  - [:punct:] punctuation mark character
  - [:space:] whitespace character
  - [:upper:] uppercase letter
  - [:xdigit:] hexadecimal digit character
  - [:d:] decimal digit character
  - [:w:] word character
  - [:s:] whitespace character

http://www.cplusplus.com/reference/regex/ECMAScript/

More Character Matching Examples

- Xa{2,3} matches Xaa Xaaa
- Xb{2} matches Xbb
- Xc{2,} matches Xcc Xccc Xcccc Xcccc
- int\s+x\s*=\s*\d+; matches an integer definition
- CSE\d{4} matches a CSE class
- \w{2}-\d{4,5} matches 2 letters and 4 or 5 digits
- \d{5}(-\d{4})? matches a 5 or 9 digit zip code
- [^aeiouy] matches not an English vowel

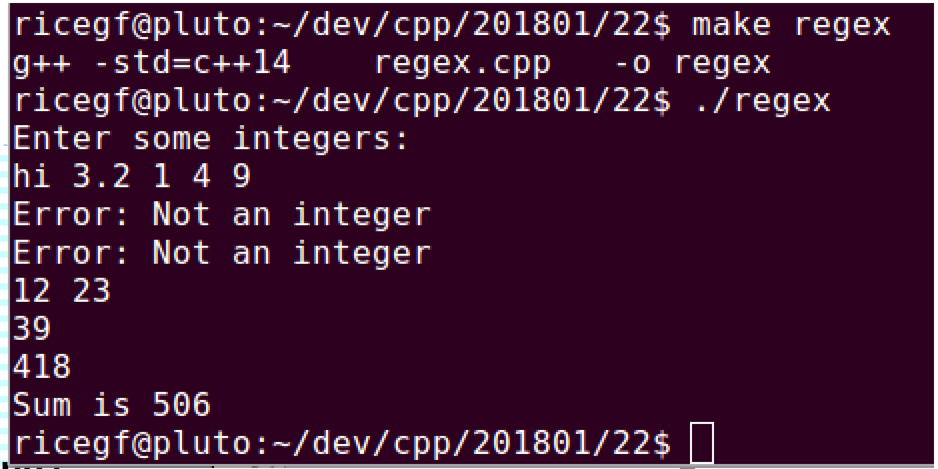
Some C++ Regex-Related Classes and Functions

- The regex class is constructed with the regular expression string as a parameter
  - If the regular expression has syntax errors, the constructor throws a regex_error exception
  - Once instanced, the regex may be used in multiple matches and searches
- regex_match(string_to_search, regex) returns true if the entire string_to_search matches the regex
- regex_search(string_to_search, regex) returns true if any substring of string_to_search matches regex

http://www.cplusplus.com/reference/regex/

A Simple C++ Regex Example

```
#include <iostream>
#include <regex>
#include <string>

int main() {
    std::string input;
    std::regex integer{"(\\+|-)?[[:digit:]]+"};
    std::cout << "Enter some integers:" << std::endl;

    int sum = 0;
    while(std::cin>>input) {
        if(std::regex_match(input,integer))
            sum += stoi(input);
        else
            std::cerr << "Error: Not an integer" << std::endl;
    }
    std::cout << "Sum is " << sum << std::endl;
}
```



```
ricegf@pluto:~/dev/cpp/201801/22$ make regex
g++ -std=c++14     regex.cpp    -o regex
ricegf@pluto:~/dev/cpp/201801/22$ ./regex
Enter some integers:
hi 3.2 1 4 9
Error: Not an integer
Error: Not an integer
12 23
39
418
Sum is 506
ricegf@pluto:~/dev/cpp/201801/22$ 
```

Regex are often VERY useful for data validation!

Regex and Beyond

- Regex are extremely powerful text manipulators
  - We’ve only scratched the thin outer surface
- Regex are widely used outside C++
  - The gedit text editor can search (and replace) using regex
  - Command line tools like grep (literally “global regular expression print”), find, and awk use regex
  - Perl famously extends regex power to ludicrous (and incredibly useful!) extremes
- If you deal with much text, you need to learn regex!

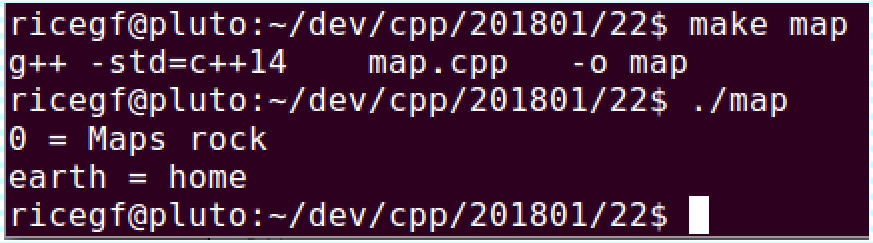
Maps

- A std::map is an associative container that stores elements in a key-value pair
  - It is essentially a vector using any type (including *your* class) as the index type
  - planetary_mass[“earth”] = 5.97; // in yottagrams, of course
- Keys must be unique
  - Only one value is associated with each key*
- Keys are in sorted order
  - Searching for an element in the map through key is thus fast(er)
  - Effectively logarithmic time
- A C++ map is roughly equivalent to a Java NavigableMap or a Python dict

* Use a multimap instead to store multiple values per key

Map Verbose Example

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <iterator>
using namespace std;

int main() {
    // With vectors (using int as the index type)
    vector<string> s;
    s.push_back("Maps rock");
    for (int i=0; i < s.size(); ++i)
       cout << i << " = " << s[i] << endl;

    // With maps (using string as the index type)
    map<string, string> m;
    m.insert(make_pair("earth", "home"));  // No push_back!
    for (map<string, string>::iterator it = m.begin(); it != m.end(); ++it) {
        cout << it->first << " = " << it->second << endl;
    }
}
```



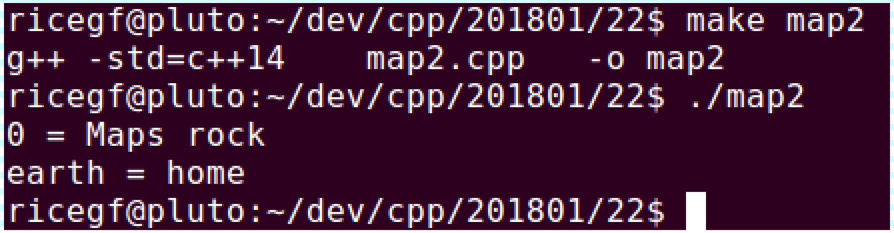
```
ricegf@pluto:~/dev/cpp/201801/22$ make map
g++ -std=c++14     map.cpp    -o map
ricegf@pluto:~/dev/cpp/201801/22$ ./map
0 = Maps rock
earth = home
ricegf@pluto:~/dev/cpp/201801/22$ 
```

Pairs

- A std::pair is a template that couples two values of different types
  - The first value is accessed via ->first
  - The second value is accessed via ->second
  - Constructors are available
    - pair<string,string> planet("earth", “home”);
  - More common is the factory make_pair
    - auto planet = make_pair("earth", "home");
- **A map thus manages pairs** – key-value pairs
- A pair is a type of *tuple*, a finite ordered list of elements – specifically, a 2-tuple

Easier Map Example

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <iterator>
using namespace std;

int main() {
    // With vectors (using int as the index type)
    vector<string> s;
    s.push_back("Maps rock");
    for (int i=0; i < s.size(); ++i)
       cout << i << " = " << s[i] << endl;

    // With maps (using string as the index type)
    map<string, string> m;
    m["earth"] = "home";
    for (auto it : m) cout << it.first << " = " << it.second << endl;
}
```

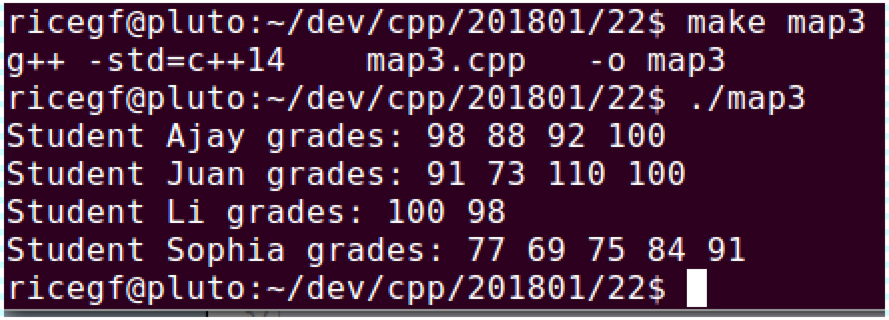


```
ricegf@pluto:~/dev/cpp/201801/22$ make map2
g++ -std=c++14    map2.cpp   -o map2
ricegf@pluto:~/dev/cpp/201801/22$ ./map2
0 = Maps rock
earth = home
ricegf@pluto:~/dev/cpp/201801/22$ 
```

Practical Map Example: Gradebook as a Ragged Array

```cpp
#include <iostream>
#include <vector>
#include <map>

int main() {
    typedef std::string Student;
    typedef std::vector<int> Grades;
    std::map<Student, Grades> gradebook;

    gradebook["Li"] = {100,98};
    gradebook["Ajay"] = {98,88,92,100};
    gradebook["Juan"] = {91,73,110,100};
    gradebook["Sophia"] = {77,69,75,84,91};

    for (auto student : gradebook) {
        std::cout << "Student " << student.first << " grades: ";
        for (int grade : student.second) std::cout << grade << ' ';
        std::cout << std::endl;
    }
}
```

Note that the keys are sorted!



```
ricegf@pluto:~/dev/cpp/201801/22$ make map3
g++ -std=c++14    map3.cpp   -o map3
ricegf@pluto:~/dev/cpp/201801/22$ ./map3
Student Ajay grades: 98 88 92 100
Student Juan grades: 91 73 110 100
Student Li grades: 100 98
Student Sophia grades: 77 69 75 84 91
ricegf@pluto:~/dev/cpp/201801/22$ 
```

Common Map Operations

- empty() returns true if the map contains no pairs
- size() returns the number of pairs in the map
- operator[ ] (rvalue) provides random access by key, adding a default value to the map if not already defined for that key
  - at(key) is like operator[ ], but instead throws an out_of_range exception if key isn’t defined
- operator[ ] (lvalue) silently overwrites the existing value for a key
  - insert(pair p) adds a pair to the map only if the key doesn’t already exist
- begin() and end() return the usual iterators
- find(key) returns an iterator to the associated value, or map::end if the key is not in the map
- erase(key) or erase(iterator1[, iterator2]) deletes the values pointed to by the iterators from the map

Other Map Types

- Multimap allows multiple values per key
- Set uses the key as the value, essentially storing a list of values alone
- Multiset counts the number of times that a key is added, with the count as the value
- Unordered_X (where X is map, set, multimap, or multiset) don’t automatically sort the keys, and are thus somewhat faster for non-search operations when searching is rare or isn’t needed

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

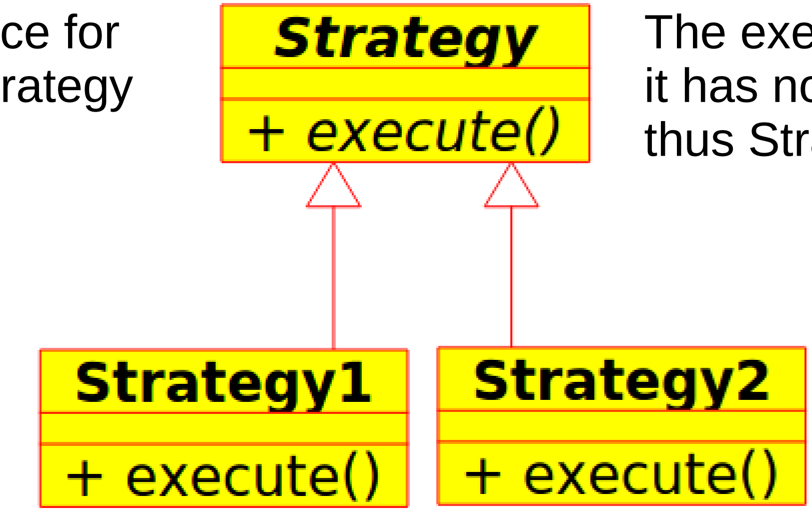
- For example, a security package may use the Strategy pattern to select different levels of file scanning for malware, depending on the file's source

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

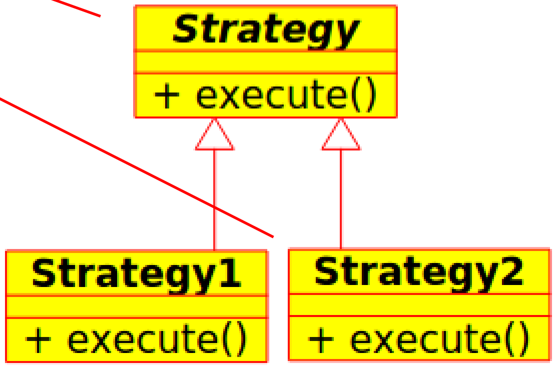
Behavioral

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!

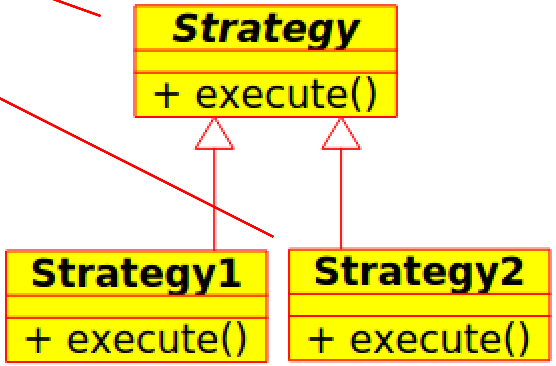
Behavioral

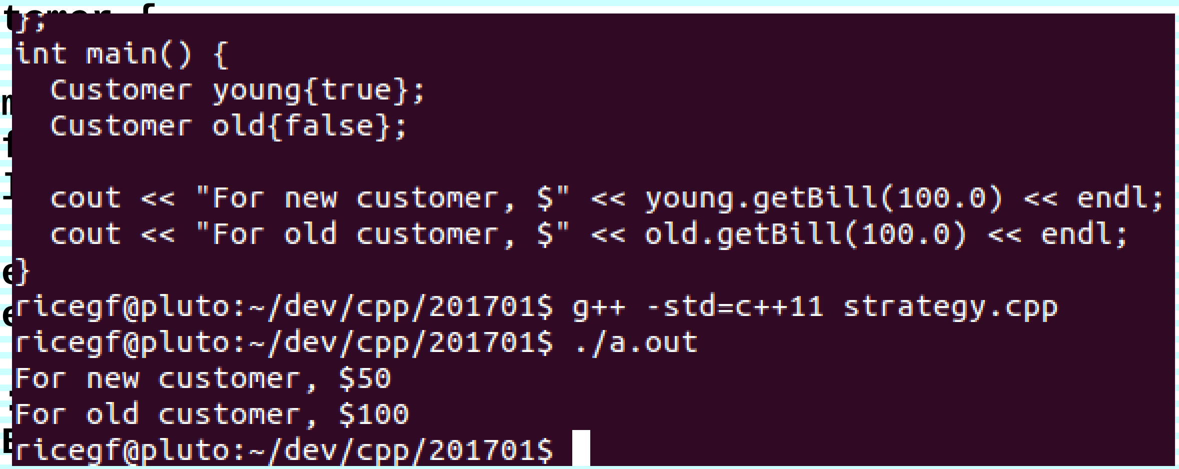
The Strategy Pattern

(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 Customer {
  public:
    Custon
        if
        el
    }
    double
        re
    }
  private:
    RobotE
};
```

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, $100
ricegf@pluto:~/dev/cpp/201701$
```

Quick Review

- List 4 options for representing strings in C++ and the most significant advantage of each.
- A(n) ________ __________ is a string that defines a search or search-and-replace pattern for other strings
  - True or False: C++ supports both Perl and Posix versions
  - List several special characters and their meaning
  - Explain how to provide a list of options, only one of which must match
  - Explain how to express (1) zero or one (2) zero or more (3) one or more (4) five to seven
  - How is a regex “compiled”?
  - Explain the difference between regex_match and regex_search

Quick Review

- How is a map similar to a vector? What’s the most significant difference?
- How are key / value pairs accessed in a map?
  (a) value = map[key]
  (b) map.key and map.value
  (c) iterator->key and iterator->value
  (d) iterator->first and iterator->second
- Which are common map operations?
  (a) navigate (b) begin and end (c) operator[ ] (d) find
- The __________ pattern enables an algorithm’s behavior to be modified at runtime.