Project-Based C++

A PBL APPROACH TO LEARNING MODERN C++

CHUCK ALLISON, UTAH VALLEY UNIVERSITY

Notice

This is Open Content

Very interested in your views on teaching/learning C++

Feel free to comment on the programing projects I present

- Including the order they're presented to students

Try to remember what you were like as a sophomore in college…

Some Context

CS 3370 – C++ Software Development

Students have had CS1 and CS2 in C++ already

- But their first course is in C# (this is a 4th course for most)
- Remember: they are *students*; they don't know what you know
- Goal: Gain intermediate-to-advanced C++ language mastery

Some Context

CS 3370 – C++ Software Development

Students have had CS1 and CS2 in C++ already

- But their first course is in C#
- Remember: they are *students*; they don't know what you know
- Goal: Gain intermediate-to-advanced C++ language mastery

Major Emphases:

- Memory management (low-level implementation of library components, RAII)
- Using C++11/14 and its Standard Library (string, vector, map, algorithms + lambdas)
- Introduces Concurrency

Some Context

CS 3370 – C++ Software Development

Students have had CS1 and CS2 in C++ already

- But their first course is in C#
- Remember: they are *students*; they don't know what you know
- Goal: Gain intermediate-to-advanced C++ language mastery

Major Emphases:

- Memory management (low-level implementation of library components, RAII)
- Using C++11/14 and its Standard Library (string, vector, map, algorithms + lambdas)
- Introduces Concurrency

Organized around 7 *projects*

- The material presented supports the projects

Project-Based Learning

Courses are a sequence of *projects*

Each project starts with a **problem to solve**

- Focuses students' minds
- Gives a mental framework for organizing what they subsequently learn

Project-Based Learning

Courses are a sequence of *projects*

Each project starts with a **problem to solve**

- Focuses students' minds
- Gives a mental framework for organizing what they subsequently learn

The problem is given **in context**

- Based on what students already know
- **Driving Question** introduces the need for the project

Project-Based Learning

Courses are a sequence of *projects*

Each project starts with a **problem to solve**

- Focuses students' minds
- Gives a mental framework for organizing what they subsequently learn

The problem is given **in context**

- Based on what students already know
- **Driving Question** introduces the need for the project

A **Rubric** for evaluating project quality is developed up-front

Project-Based Learning

Courses are a sequence of *projects*

Each project starts with a **problem to solve**

- Focuses students' minds
- Gives a mental framework for organizing what they subsequently learn

The problem is given **in context**

- Based on what students already know
- **Driving Question** introduces the need for the project

A **Rubric** for evaluating project quality is developed up-front

Students must must provide a post-project **reflection** describing their experience

- What did they learn?
- What obstacles were overcome?

The Projects

The Projects

1. Deque (**new**, **delete**, dynamic size, references)
2. Class-based Memory Pool (**operator** new/delete, implicit structure)
3. XML-to-Binary File Converter (strings, binary file I/O, **unique_ptr**)
4. Numeric Data Reduction (algorithms, lambda expressions)
5. Word-to-line Cross-Reference Generator (**map**, custom strict weak orders, **regex**)
6. Implement **dynamic_bitset** (aka **vector<bool>**) without manual memory management
   - std::vector does the memory management (operator overloading, proxy classes, **&&**)
7. Pipelined producer-consumer threads (**thread**, **mutex**, **condition_variable**, **async**, **atomic**)

Module 1

MEMORY MANAGEMENT 1 – IMPLEMENTING A SIMPLE DEQUE

About Memory Management

3 Types of memory:

Static

Automatic (stack)

Heap (“free store”)

Which is most efficient?

Which is most flexible?

What kind of memory do containers like **vector** use internally?

Why is memory management important?

Driving Question

How can we as C++ programmers use good dynamic memory management techniques to create a random access, sequential data structure that allows efficient addition and removal of elements at either end of the sequence?

Issues:

- Arrays and vectors won't work, because they are fixed at the left end
  - Would have to *shift* elements to the *right* to *insert*
  - Would have to *shift* elements to the *left* to *delete*
  - This is the *opposite* of efficient
- Example: Consider a *queue*
  - Need to insert at one end and delete at the other

review program 1 spec

PROG1.DOCX

In Module 1 You Will Learn About…

lvalues vs. rvalues

Pointers:

- pointer arithmetic
- pointers and Arrays
- pointers to dynamic memory on the heap
- pointers and **const**

Memory Management

- RAII: using **new** in constructors and **delete** in destructors
- growing dynamic arrays

References (aliases, reference parameters as “automatic pointers”)

C++11 features: **auto**, **decltype**, uniform initialization, **nullptr**, range-based **for** loops, type aliases with **using**, **=delete**

Sample Reflections

Challenges: The hardest part was figuring out what pointers I needed and figuring out the relationship between p_front and p_back. I also attempted to more precisely center the data by moving where the middle pointer was, say if you only added data using push_front the middle pointer would be farther in the right of the array to better make use of space. However that was quite a bit harder and after realizing I didn't need to do it that way to pass the tests I gave up.

Learned: I learned how hard it is to get pointers correct, and how much easier drawing it out on paper makes figuring out the relationships. I also learned how it important it is to get p_front and p_back just right in terms of where they start and end.

I like C# memory management, in C++ this is an added concern.

C++ adds gray hairs to people.

Module 2

CLASS-BASED MEMORY POOL

More about **new** and **delete**

Consider the following calls to **new**:

```
int* p = new int;
int* q = new int[10];
```

Now consider what happens when **delete** is called:

```
delete p;
delete [] q;
```

How does the system know how many bytes to return to the free store in each case?

Driving Question

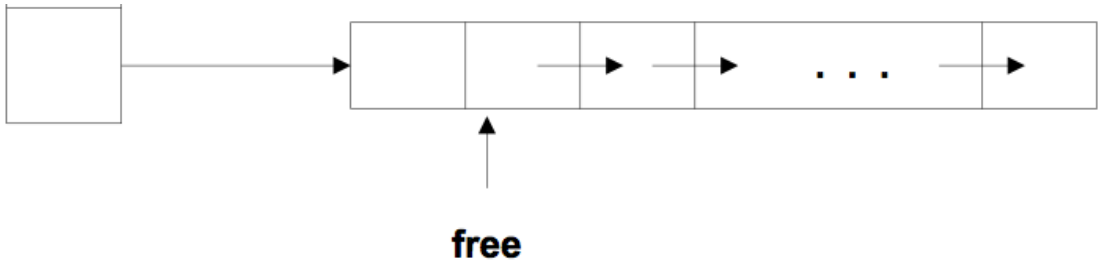
How can we as talented C++ developers use advanced memory management techniques to control and optimize the use of heap memory for all objects of an arbitrary, given class?

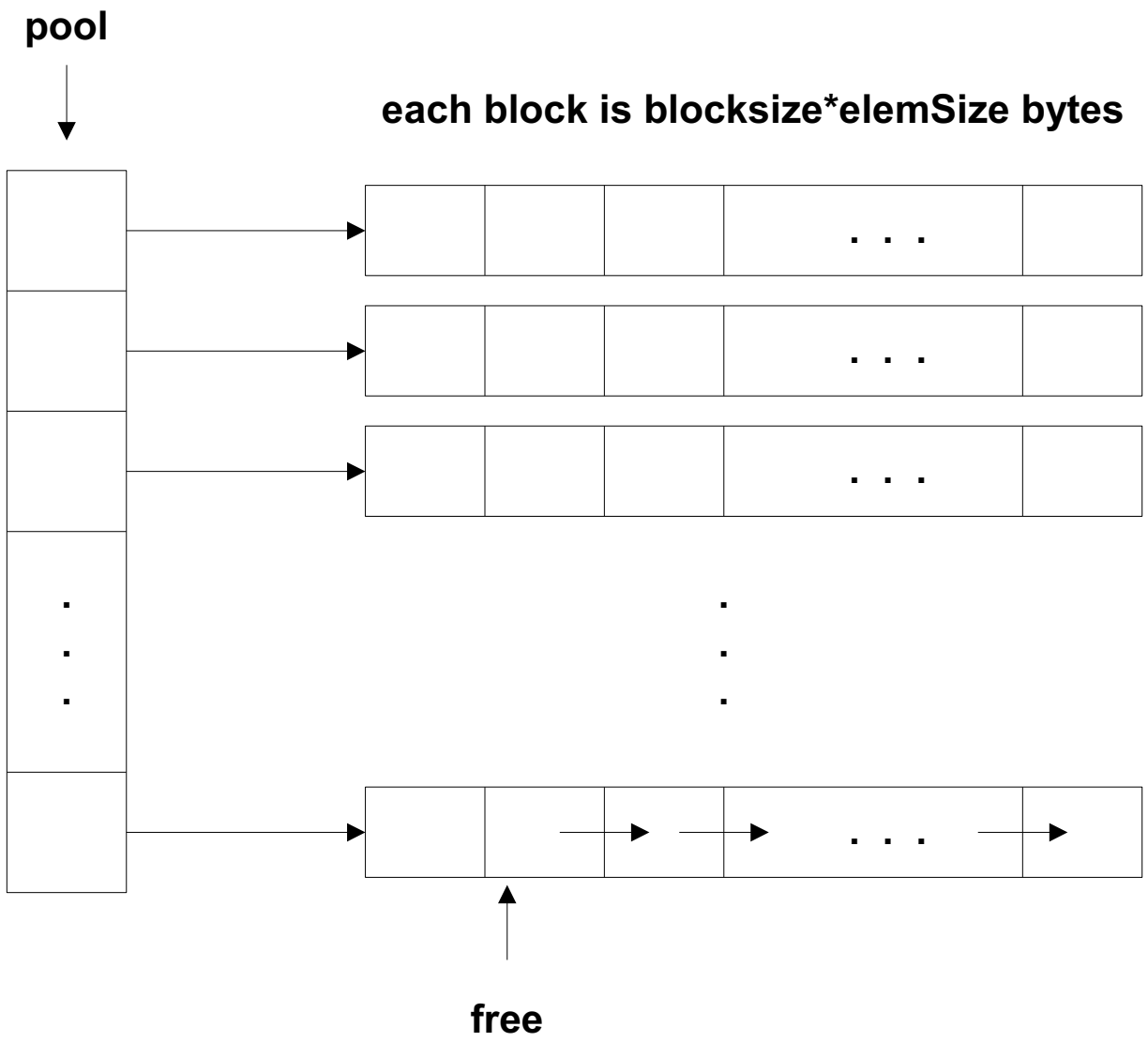
Issues:

- There is overhead for every invocation of the **new** operator
- To lose some of this overhead, can we allocate space for many objects in a single invocation of the **new** operator? We did this for Program 1.
- More important, can we *reuse* spaces that were previously used, even if they are released (via **delete**) in random order?

Implicitly Linked Cells

A Pool Class





Prog2.docx

You Will Learn About…

More Memory Management

- changing the way **new** and **delete** work
- creating a separate “heap” for a class
- reinterpreting memory at an arbitrary address (pointer “puns”)

Managing Object State and Lifetime

- RAII – “Resource Acquisition is Initialization”
- Controlling initialization, copying, assignment, and cleanup
- Smart pointers
- “Placement **new**” for flexible, low-level initialization

Static Memory

- static data members, static initialization, factory functions

The hardest part for me was wrapping my head around reinterpret cast. The main insight for me was realizing that the reason you do it is because when dereferencing a char*, it will give you char, and reinterpreting to a char** will cause the dereference to cause it to act as a char*. Another aspect that took awhile was figuring out how to do the grow and allocating block array functions, not necessarily the code itself, but figuring out what to even do. I feel like I understand well now the purpose of reinterpret_cast in these scenarios, and also how to use pointers better.

```
// Summary: Here are the things I had to learn:
//  I was reminded of how to create classes with both a cpp and h file.
//  I learned how to overload operators -- especially new, delete, and <<; also static functions.
//  I was exposed to the Simple Factory Method design pattern
//  I got practice with creating a 'dynamic' array
//  I learned that default parameters are only put in the declaration, not function definition.
//  I learned something about double pointers, and reinterpret_cast'ing pointers
//  I learned about placement new.
//  I learned about friend functions
//  I learned about how objects are deleted when they go out of scope
//  I got more practice with the std::copy function
//  I learned how to do your own memory management for a class!
//  I learned how to create a static class variable
//  I got practice with pointers.
//  I learned about the two roles of 'new': allocating memory (operator new) and the c'tor
//  I learned about the two roles of 'delete': the d'tor and freeing memory (operator delete)
```

Module 3

STRING PARSING; FIXED-LENGTH-RECORD BINARY FILES

Have You Ever Wondered…

…how database systems store data in files?

Relational: Oracle, SQL Server, MySQL, PostgreSQL

NoSQL: MongoDB, CouchDB, BerkeleyDB

How is object data written to storage?

How are numbers written to storage?

Driving Question

How does a C++ programmer do I/O various file formats? In particular, how can we process XML data and fixed-length, *raw data* interchangeably?

Side Query:

- What happens with the following code, if **s** is a **std::string** and **f** is a **std::ofstream**?

```
f.write(reinterpret_cast<char*>(&s),sizeof(s));
```

Review program 3 spec

PROG3.DOCX

You Will Learn About…

Command-line arguments

std::string

- string processing function
- <cctype> and its character processing functions
- String iterators

I/O Streams

- stream state
- string streams
- file I/O
- Stream manipulators

Exception Safety

Remarks: this was the hardest programming assignment I think I've ever had to do. I constantly ran into obscure errors or reasons why something wouldn't work, all involving file I/O. The part that took me the longest by far was the fromXML parsing. I saw after I finished that I could have used the lowercase compare function, but I think the way I did it works well enough using std::transform.

The most frustration came from not getting the salary to update. Then a revelation happened where I realized I wasn't moving the stream position back to the beginning to update the correct data. I have a better understanding of streams now, and beginning to understand why C++ development is so hard.

I learned how to go through a stream and extract information when in XML style. Frankly I'm still not fully comfortable with parsing as I feel that I could easily mess up by a single character. For this reason I prefer solidly tested libraries and languages that will do it for me. When I began coding the parsing the XML portion I felt as if I was using a small hammer and small chisel to break a big slap of concrete. Time consuming, progress is small, and at the same time I knew that there are jack hammers (libraries, languages) that can get the job done much quicker. When I code in C++ I always feel this way. Blitzmax and C# get the job done quicker, and I often don't need this level of control.

I also learned that I can pass an iostream object to a ostream, since inheritance allows it view 'void store(std::iostream& io) const' for more details--saved me some gray hairs induced by C++.

Unique_ptrs are neat to use, but I did notice that one cannot assign it to a variable and then place the variable into the vector, it looks like one has to place the object directly into the vector.

Module 4

DATA REDUCTION WITH ALGORITHMS

Numeric Data Processing

Since C++ (along with C) is the fastest high-level language, you would imagine it is a good choice for working with mathematical calculations. You would be right!

But in addition to execution speed, C++ allows fabulous development speed because of the high-level abstractions in its standard library…

Driving Question

How does one efficiently process numeric data in C++? How can we make optimum use of the standard library to minimize the amount of work we have to do?

Issues:

- Hand-crafting loops is error prone
- Lots of list processing logic is already done for you

Review program 4 spec

PROG4.DOCX

You Will Learn About…

Generic Algorithms

Function Objects

std::bind

Lambda Expressions

Iterators

Module 5

CROSS-REFERENCE GENERATOR

Driving Question

What is the best way to create a word-to-line cross-reference from a text file? (This could be used to know where certain variables are used in code, for example.)

Issues:

- We want to know where (i.e., in which lines) each word of a text file appears.
- We want to *preserve* uppercase and lowercase as they appear in the file, yet we want the same "word" in different cases to be grouped together in an alphabetical listing:
  - x : 2, 10, 11, 20, 71
  - X : 1, 22, 100, 121

```
A                       : 48:1
a                       : 9:1, 10:1, 12:2, 14:1, 17:2, 19:1, 26:1, 27:1, 28:2,
                        : 39:1, 41:1, 43:1, 45:2, 46:2, 49:1, 50:2, 51:1, 56:3,
                        : 81:1, 82:1, 94:1, 111:1, 112:1, 114:1, 117:1, 132:1, 135:1,
                        : 138:1, 142:2, 143:1, 144:1, 152:1, 156:1, 161:2, 163:1, 164:1,
                        : 167:1, 169:1, 175:1, 182:2, 190:1, 192:1
about                   : 16:1, 29:1, 166:1, 190:1, 191:1
above                   : 137:1
accompanied             : 6:1
across                  : 26:1
admit                   : 20:1
advancing               : 170:1
After                   : 166:1
after                   : 130:1
again                   : 155:1
```

Prog5.docx

You Will Learn About…

Sequence Containers

- **vector**, **deque**, **list**, **forward_list**, **array**

Special, Higher-level Containers

- **stack**, **queue**, **priority_queue**

Associative Containers

- **set**, **multiset**, **unordered_set**, **unordered_multiset**
- **map**, **multimap**, **unordered_map**, **unordered_multimap**

Comparators and Ordering

- strict partial orders (aka “strict weak orders”)

Regular Expressions

I had always been intimidated by regular expressions. How could you blame me after seeing one. After reading through the book's section on regular expressions and watching a few video's I figured it out. This assignment was helpful in that it forced me to just learn regex.

I had a great deal of frustration trying to figure out why my inner map wasn't keeping the values I inserted into it. I tried several different variations of map::insert and also map::operator[]. A helpful person on IRC pointed out to me that the auto keyword will copy a by value even though operator[] returned by reference.

Module 6

DYNAMIC BIT STRINGS

Driving Question

How can we process individual bits as if they were *addressable*, like separate elements in a sequence, but still save as much memory as possible? We want an expandable bit array that works like a string or vector with index capability via operator[ ].

Issues:

- Individual bits are not addressable
- Large files, such as video, where the setting of individual bits is significant, take up lots of memory
- We want the best of both worlds: addressable bits but bits packed into integers to save space.

Prog6.docx tbitarray.cpp test.h

You Will Learn About…

Bitwise Operations

Special **vector** Operations

Move Semantics

Operator Semantics
- order of evaluation
- pre-vs.-post ++ and --

Operator Overloading
- unary and binary operators
- member vs. non-member operators
- implicit conversions to and from class types
- **const** vs. non-**const** operator functions

Templates

This was an extremely fun but frustrating assignment. early on I had thought i had implemented the comparitors in a nice and clever way using std::algorithm functions, only to realize (tho obvious in hindsight) that they wouldnt work. I wish i hadnt been so busy this week so i could have tried implementing some methods more efficiently... looping though the bits one by one is not necessary for many operations. I could have used bit-wise operations to "move" around bits in chunks, and cases where operations span many blocks could have just used algorithm/vector functions and shifted the result or something. Shame that near the end of the semester is when i really want to spend more time on the programs, but have less time than normal :)

Module 7

PIPELINED TASKS

Moore’s Law

The number of transistors in a single chip doubles every 18 months

- so computing power has doubled accordingly

Approaching a physical limit at the atomic level between 2015 and 2020

How can we attain more power despite this limit?

Multiple Cores

Allow multiple machine instructions to execute *simultaneously*

- 1 per core

How can we take advantage of this?

Most programs execute sequentially

- one instruction at a time
- one statement after another

Humans also *think* sequentially!

Driving Question

Many problems are best solved as a series of *sequential steps*, where some of the steps produce streams (sequences of unknown size) of data. How can we arrange for subsequent steps to process data elements as soon as they receive them from a previous step, without waiting for the previous step to complete *all* its work? In other words, we want the different steps to run *concurrently*.

Issues:

- We need a way to separate the steps into distinct units of functionality that can run *interdependently* and *simultaneously*.

Prog7.docx

You Will Learn About…

Concurrency

Parallelism (a special case of concurrency)

Threads

Shared Memory
- Mutexes
  - **lock/unlock**, **lock_guard**, **unique_lock**
- condition_variables
  - **wait**, **notify_one**, **notify_all**

Task-based Concurrency
- promises and futures
- **async**

Atomic Operations

Very useful to learn about concurrency; free lunch ended years ago IMO. I would like to learn some patterns for designing classes around concurrency as I can see common logic in this program that ought to be factored out (there are 2 sets of identical variables, function checks and loops, etc.), but I had difficulty determining a good paradigm to use (that wasn't overly complex). possibly I overlooked some existing C++ library functions/classes that may have been helpful. I would very much like to see your solution for this since I'm sure you took care to cleanly resolve code duplication.

If you don't mind, I would like to see your solution after you grade this. I want to see how you were able to cleanly (and hopefully simply) factor out duplicate code. I had 2 sets of identical variables and some duplicate logic that I know could have been done differently, but I couldn't come up with anything that felt right. It felt like have duplicate code, or have something convoluted and hard to debug...

Summary

Learning via Projects is an effective learning and organizational tool

Focuses student learning

Calls for self-examination and introspection

Oft-quoted feedback from graduates: “This class helped me get my job!”