Which Machine am I Coding To?

Patrice Roy
Patrice.Roy@USherbrooke.ca
Université de Sherbrooke
Collège Lionel-Groulx

C++ Users Meeting, Montréal, juin 2017

Who am I?

- Father of five (four girls, one boy), ages 22 to 4
- Feeds and cleans up after a varying number of animals
- Used to write military flight simulator code, among other things
- Full-time teacher since1998
- Works a lot with game programmers
- Incidentally, WG21 and WG23 member
- Involved in SG14

How we will proceed...

- I have a few things to say
- You can ask questions whenever you want
  - Be nice, of course; let's remain civilized
- You have a right to disagree
- I might not have all the answers
  - I should have a few
  - I'll do my best for the rest

The Abstract…

- When writing a C++ program, we tend to think of the strengths and weaknesses of our computer, just as we think of our algorithms, data structures, and probably of language features we want to use (or we want to avoid), and we code accordingly
- To some, it might be surprising to learn that C++ is actually specified in terms of an abstract machine, with its own characteristics. If this is indeed a surprise for you, then you might be interested in knowing more about this machine
  - It's been there for a long time, and it influences the way we program as well as the way the language was, and is.
- The aim of this talk is to provide a practical overview of what the C++ abstract machine is, how it affects the way we program and how it affects language design itself
- It will probably most interesting to intermediate audiences who would like a closer look to some of the abstract underpinnings of the language.

The Plan…

- How we write programs, and what we count on
- What an abstract machine is
- What the C++ abstract machine is
- How the C++ abstract machine is represented (stack? registers? memory? pointer arithmetic? rules?)
- How the C++ abstract machine influences what is / isn't a correct C++ program
- Relation to undefined behavior
- Impact on program design
- Impact on language evolution
- Concrete examples of the impact the C++ abstract machine has on our programming practice

How we write programs, and what we count on

Abstract machine?

- What is an abstract machine? Opinions vary… but tend to converge
  - "*Programming language specifications (not just C and C++, all high-level programming language specifications), define the languages in terms of an abstract machine, which, in this usage, is the simplest imaginary computer capable of executing a program in the source language (or a family of languages, as in the case of the JVM).*"
    - Sergey Zubkov, 2015, answering on https://www.quora.com/What-is-C++-abstract-machine

Abstract machine?

- What is an abstract machine? Opinions vary… but tend to converge
  - "*An abstract machine is a model or design which permit step by step execution. Modern programming languages are not designed for a specific target […] but a range of targets. So language specifications are designed with a minimal hypothetical machine in mind that permits random access of contiguous blocks of memory (as opposed to stack based machine used for languages like postscript), basic arithmetic and logical operations, branching capabilities etc. This model is called abstract because it omits many details of real (hardware) machines. This minimal abstract design enables compiler writers to write compiler for a variety of targets.*
  - *This abstract machine was single-threaded for older C/C++ specifications but starting with C++11 (C++11/14/17), this abstract machine allows atomic operation enabling multi-threading environment.*"
    - Mohit Jain, 2015, answering on https://www.quora.com/What-is-C++-abstract-machine

Abstract machine?

- What is an abstract machine? Opinions vary… but tend to converge
  - "*An abstract machine, also called an abstract computer, is a theoretical model of a computer hardware or software system used in automata theory. Abstraction of computing processes is used in both the computer science and computer engineering disciplines and usually assumes a discrete time paradigm.*"
    - https://en.wikipedia.org/wiki/Abstract_machine (last checked on June 26, 2017 around 22:00)
    - They add, somewhat rightfully (as opinions vary): "*Not to be confused with virtual machine*"

Abstract machine?

- What is an abstract machine? Opinions vary… but tend to converge
  - "*Abstract machines are the core of what defines a programming language. A wide variety of operating systems and plethora of hardware ensures a bewildering number of things we call computers. When we write a program we generally don't want to think about the differences between all these platforms. We'd like to somehow be abstracted from all that and simply worry about our problem domain. To do so we choose some portable programming language. Something which doesn't care about what hardware it is running on.*
  - *But if we don't say what machine our program is running on, how do we know what the program is actually doing? A program is nothing more than a series of instructions; they have to control something. To give them something to control, yet avoid dependence on a specific piece of hardware, we invent a new machine. We define the characteristics of this machine and then describe the language as a way to manipulate that machine.*
  - *This machine is called the abstract machine […].*"
    - Edaqua Mortoray, author of the Leaf language, 2012, https://mortoray.com/2012/06/18/abstract-machines-interpreters-and-compilers/
    - He adds (rightfully?), "*In academia both Turning Machines and Lambda Calculus are examples of abstract machines*".

C++’s abstract machine?

C++'s abstract machine?

- In Bjarne Stroustrup's own words, from 2005:
  - "*C++ is used in essentially every application areas (*sic.*), incl. [...long list...]. How does C++ support such an enormous range of applications? The basic answer is:"by good use of hardware and effective abstraction"*."
    - http://www.stroustrup.com/abstraction-and-machine.pdf
    - Clearly, C++'s abstract machine is a very important concept of the standard

C++'s abstract machine?

- In Bjarne Stroustrup's own words, from 2005:
  - "*C++ maps directly onto hardware. Its basic types (such as char, int, and double) map directly into memory entities (such as bytes, words, and registers), most arithmetic and logical operations provided by processors are available for those types. Pointers, arrays, and references directly reflect the addressing hardware. There is no "abstract", "virtual" or mathematical model between the C++ programmer's expressions and the machine's facilities. This allows relatively simple and very good code generation.*"
    - http://www.stroustrup.com/abstraction-and-machine.pdf
    - Note the "such as" for the mapping of types to hardware. Don't assume this will be your compiler's mapping!

C++'s abstract machine?

- In Bjarne Stroustrup's own words, from 2005:
  - "*C++'s model, which with few exceptions is identical to C's [...]. For example, there is nothing in C++ that portably expresses the idea of a 2nd level cache, a memory-mapping unit, ROM, or a special purpose register [...] Using C++, we can get really close to the hardware, if that's what we want.*"
    - http://www.stroustrup.com/abstraction-and-machine.pdf

(C's abstract machine?)

- Informally
  - (taken from a summary provided by David Chisnall)
    - http://cs.swan.ac.uk/~csdavec/HPC/1IntroAndPointers.pdf
  - Very close to the way a "real" machine works
  - Language constructs tend to map to short sequences of machine instructions
  - Avoids imposing costs on a real program, e.g.:
    - No garbage collection
    - No bounds-checking of arrays

(C's abstract machine?)

- Informally
  - (taken from a summary provided by David Chisnall)
    - http://cs.swan.ac.uk/~csdavec/HPC/1IntroAndPointers.pdf
  - Memory exposed as a flat array of individually addressable bytes
  - Pointers represent typed addresses
    - Casts are not really verified
    - It's on the programmer's shoulders to know what type, if any, lies at a given memory location
    - Can point to invalid memory locations
    - Can point to a function
  - Simple, efficient, but handle with care

C++'s abstract machine?

- Now, what does the standard have to say?
  - Without much of a surprise given their influence, the key ideas behind C++'s abstract machine appear early in the text

C++'s abstract machine?

- [intro.execution] p.1:
  - "*The semantic descriptions in this International Standard define a parameterized nondeterministic abstract machine. This International Standard places no requirement on the structure of conforming implementations. In particular, they need not copy or emulate the structure of the abstract machine. Rather, conforming implementations are required to emulate (only) the observable behavior of the abstract machine as explained below.*"
    - http://eel.is/c++draft/intro.execution#1

C++'s abstract machine?

- [intro.execution] p.1, fn. 6:
  - "*This provision is sometimes called the "as-if" rule, because an implementation is free to disregard any requirement of this International Standard as long as the result is as if the requirement had been obeyed, as far as can be determined from the observable behavior of the program. For instance, an actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no side effects affecting the observable behavior of the program are produced.*"
    - http://eel.is/c++draft/intro.execution#footnote-6

C++'s abstract machine?

- Right away, something interesting: the abstract machine is…
  - Parameterized
  - Non-deterministic
- C++ implementations have to behave, for all observable purposes (sort of), as if you were executing your code on its abstract machine
  - Program execution can omit computation of anything that has no impact on the result of your computation

C++'s abstract machine?

- Example: which loop leads to shorter code?
  - https://godbolt.org/g/bvbaZ8

C++'s abstract machine?

- Example: which loop leads to shorter code?
  - https://godbolt.org/g/HqU5MA
  - Oops!

C++'s abstract machine?

- Example: find on a vector or on a set?
  - http://quick-bench.com/djSxBRQ50fjsLLRvt6CjW6TQGlI

C++'s abstract machine?

- Example: find on a vector or on a set?
  - http://quick-bench.com/acOXEKxYReLE74gkjuw3kSzO_ig
  - Oops!

C++'s abstract machine?

- "*Reading an object designated by a volatile glvalue, modifying an object, calling a library I/O function, or calling a function that does any of those operations are all* side effects, *which are changes in the state of the execution environment*"
  - http://eel.is/c++draft/intro.execution#14
  - If your program has no observable side effect it's *as-if* it never executed

C++'s abstract machine?

- The *as-if* rule is very powerful
  - You want it!
  - Careful in multithreaded programs with shared state
    - Compilers can get overly aggressive if you're not explicit enough
    - End up with fast, wrong programs!
- A note on program size
  - A small, no-op program can be much bigger than a few bytes
    - Due to linking with libraries, including the standard library
    - Work does not end with what the compiler generates!

C++'s abstract machine?

- Implementation-defined elements
  - `sizeof(int)` for example, or `sizeof(T)` for most types `T`
  - Note that `sizeof(char)==1`
    - One byte, not necessarily eight bits
  - "*Certain aspects and operations of the abstract machine are described in this International Standard as implementation-defined [...] These constitute the parameters of the abstract machine. Each implementation shall include documentation describing its characteristics and behavior in these respects.*"
    - http://eel.is/c++draft/intro.execution#2

C++'s abstract machine?

- Unspecified elements
  - For example, evaluation of expressions in a *new-expression* if the allocation function fails to allocate memory
  - "*Where possible, this International Standard defines a set of allowable behaviors. These define the nondeterministic aspects of the abstract machine. An instance of the abstract machine can thus have more than one possible execution for a given program and a given input.*"
    - http://eel.is/c++draft/intro.execution#3

C++'s abstract machine?

- Undefined elements
  - For example, attempting to modify a `const` object
  - "*Certain other operations are described in this International Standard as undefined [...]. [Note: This International Standard imposes no requirements on the behavior of programs that contain undefined behavior. — end note].*"
    - http://eel.is/c++draft/intro.execution#4

C++'s abstract machine?

- Undefined elements (cont'd)
  - "*A conforming implementation executing a well-formed program shall produce the same observable behavior as one of the possible executions of the corresponding instance of the abstract machine with the same program and the same input. However, if any such execution contains an undefined operation, this International Standard places no requirement on the implementation executing that program with that input (not even with regard to operations preceding the first undefined operation).*"
    - http://eel.is/c++draft/intro.execution#5

C++'s abstract machine?

- Program execution
  - RAII is part of the execution model
  - "*An instance of each object with automatic storage duration is associated with each entry into its block. Such an object exists and retains its last-stored value during the execution of the block and while the block is suspended (by a call of a function or receipt of a signal).*"
    - http://eel.is/c++draft/intro.execution#6
  - "*The lifetime of an object o of type T ends when: if T is a class type with a non-trivial destructor, the destructor call starts, or the storage which the object occupies is released, or is reused by an object that is not nested within o.*"
    - http://eel.is/c++draft/basic.life

C++'s abstract machine?

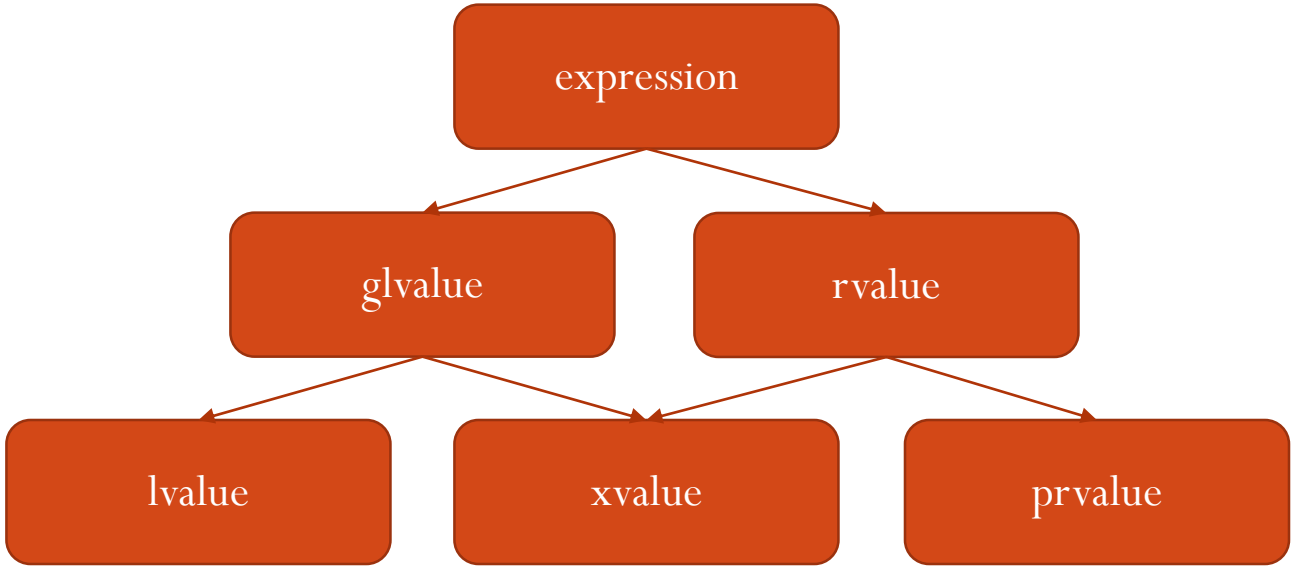
- Program execution (cont'd)
  - "*The least requirements on a conforming implementation are:*
    - *Accesses through volatile glvalues are evaluated strictly according to the rules of the abstract machine.*
    - *At program termination, all data written into files shall be identical to one of the possible results that execution of the program according to the abstract semantics would have produced.*
    - *The input and output dynamics of interactive devices shall take place in such a fashion that prompting output is actually delivered before a program waits for input. What constitutes an interactive device is implementation-defined.*
  - *These collectively are referred to as the observable behavior of the program.*
    [Note: More stringent correspondences between abstract and actual semantics may be defined by each implementation. — end note]"
    - http://eel.is/c++draft/intro.execution#7

C++'s abstract machine?

- Also defines such things as…
  - Rules regarding operator precedence
    - The compiler will respect what you wrote an apply normal precedence rules
    - It can rewrite your expression but cannot introduce overflows or exceptions (where applicable) in so doing
  - What is an expression
    - … and a subexpression
    - … and a full-expression
    - … etc.

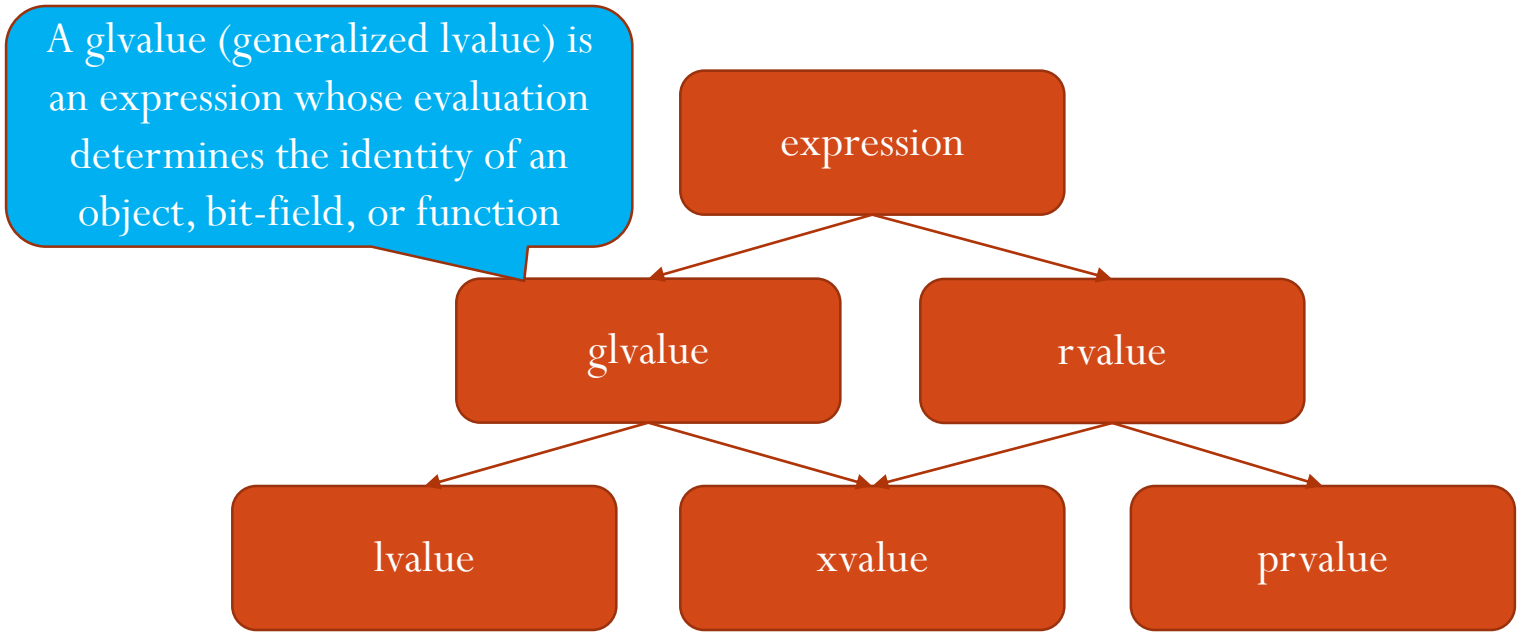
C++'s abstract machine?

- Types of expression (taxonomy)
  - http://eel.is/c++draft/basic.lval#1



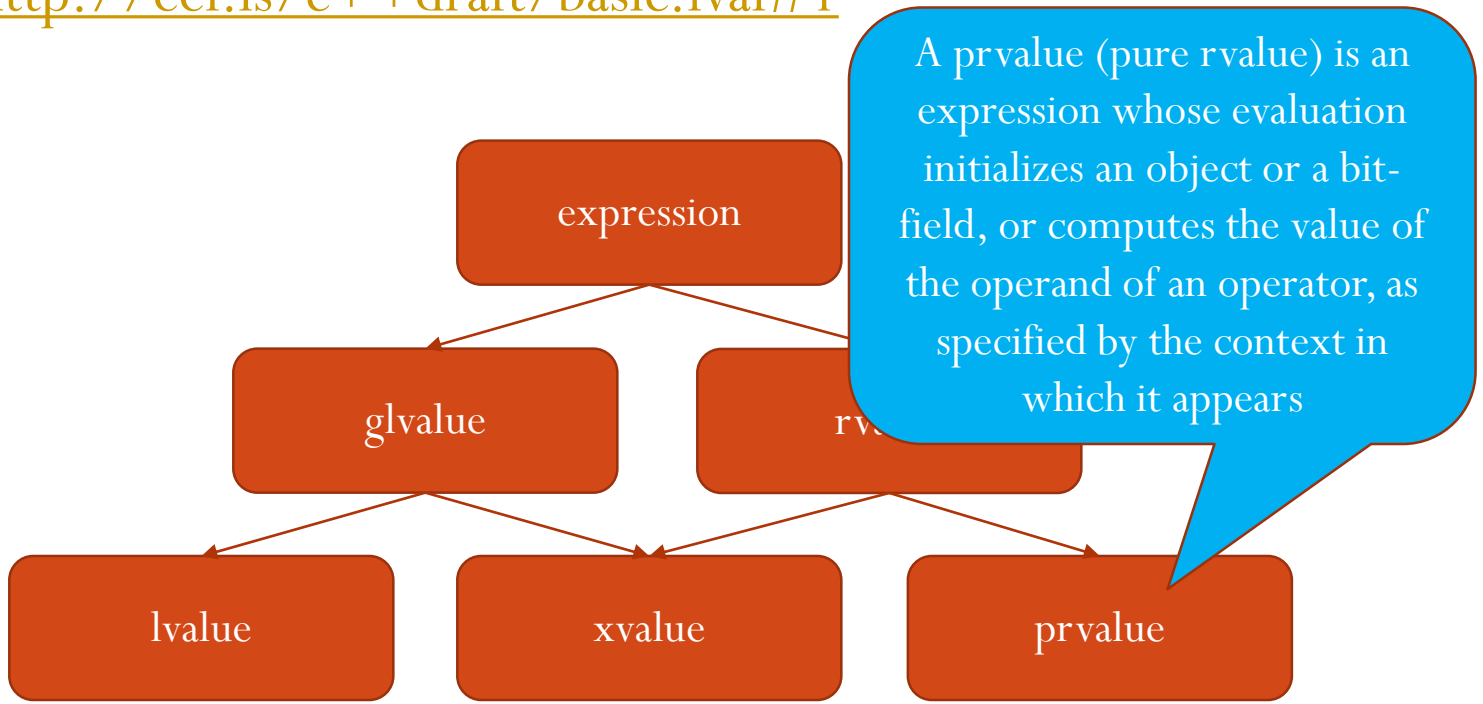
C++'s abstract machine?

- Types of expression (taxonomy)
  - http://eel.is/c++draft/basic.lval#1



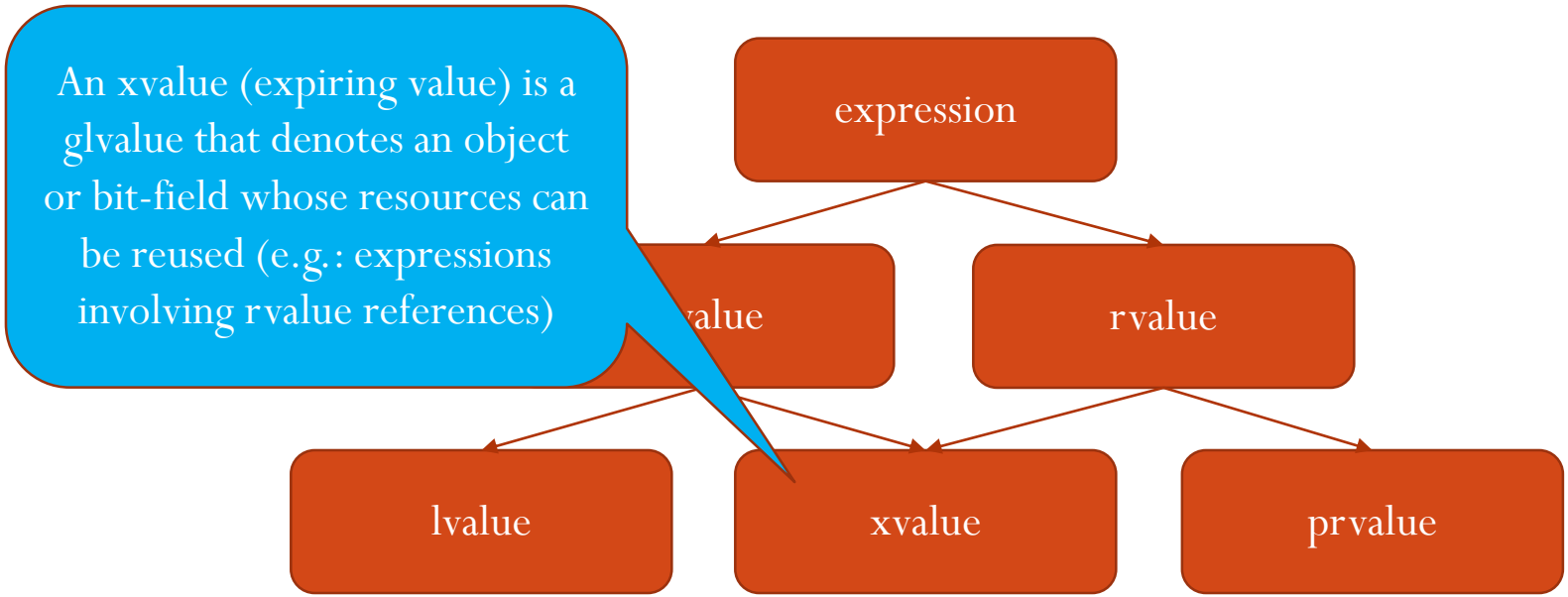
C++'s abstract machine?

- Types of expression (taxonomy)
  - http://eel.is/c++draft/basic.lval#1



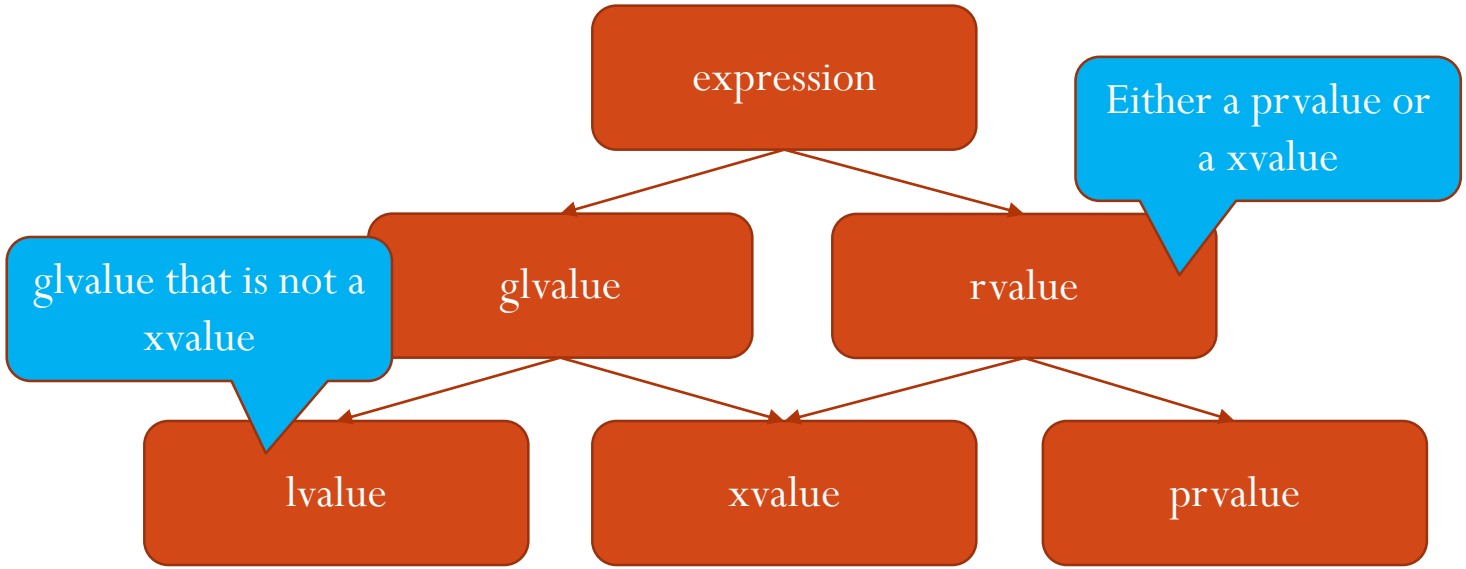
C++'s abstract machine?

- Types of expression (taxonomy)
  - http://eel.is/c++draft/basic.lval#1



C++'s abstract machine?

- Types of expression (taxonomy)
  - http://eel.is/c++draft/basic.lval#1



C++'s abstract machine?

- Function `main()` is special:
  - "*A program shall contain a global function called main. Executing a program starts a main thread of execution in which the main function is invoked, and in which variables of static storage duration might be initialized and destroyed. It is implementation-defined whether a program in a freestanding environment is required to define a main function.*"
    - This allows special implementations on embedded systems
  - "*An implementation shall not predefine the main function. This function shall not be overloaded. Its type shall have C++ language linkage and it shall have a declared return type of type int, but otherwise its type is implementation-defined. An implementation shall allow both*
    - *a function of () returning int and*
    - *a function of (int, pointer to pointer to char) returning int*
  - *as the type of main.*"
    - http://eel.is/c++draft/basic.start.main

C++'s abstract machine?

- Function `main()` is special (cont'd):
  - "*The function main shall not be used within a program. The linkage of main is implementation-defined. A program that defines main as deleted or that declares main to be inline, static, or constexpr is ill-formed. The main function shall not be declared with a linkage-specification. A program that declares a variable main at global scope or that declares the name main with C language linkage (in any namespace) is ill-formed. The name main is not otherwise reserved.*
  - *[...] A return statement in main has the effect of leaving the main function (destroying any objects with automatic storage duration) and calling std::exit with the return value as the argument. If control flows off the end of the compound-statement of main, the effect is equivalent to a return with operand 0.*"
    - http://eel.is/c++draft/basic.start.main

C++'s abstract machine?

- Memory model (overview)
  - "*The fundamental storage unit in the C++ memory model is the byte. A byte is at least large enough to contain any member of the basic execution character set and the eight-bit code units of the Unicode UTF-8 encoding form and is composed of a contiguous sequence of bits, the number of which is implementation-defined. [...] The memory available to a C++ program consists of one or more sequences of contiguous bytes. Every byte has a unique address.*"
    - http://eel.is/c++draft/intro.memory#1

C++'s abstract machine?

- Memory model (overview)
  - "*Comparing unequal pointers to objects is defined as follows:*
    - *If two pointers point to different elements of the same array, or to subobjects thereof, the pointer to the element with the higher subscript compares greater.*
    - *If two pointers point to different non-static data members of the same object, or to subobjects of such members, recursively, the pointer to the later declared member compares greater provided the two members have the same access control and provided their class is not a union.*
    - *Otherwise, neither pointer compares greater than the other.*"
    - http://eel.is/c++draft/expr.rel#3

C++'s abstract machine?

- Memory ordering
  - Compilers and the underlying platform reorder your operations under the *as-if* rule
    - That rule is more complex with multithreading
  - C++ offers SCR-DRF, or Sequential Consistency if Data Race Free
    - Being sequentially consistent means you can understand the execution you got based on your source code
      - Seems evident, but it really isn't…
    - A data race occurs when (a) a single object can be accessed concurrently by at least two threads (b), at least one of the accesses is a write, and (c) there is no synchronization
      - A data race is a thing of evil

C++'s abstract machine?

- Memory ordering (cont'd)
  - Synchronization operations are typically done with mutexes and atomics
    - Some operations on atomics are not synchronization operations, in particular those on relaxed atomics
  - "*All modifications to a particular atomic object M occur in some particular total order, called the modification order of M. [Note: There is a separate order for each atomic object. There is no requirement that these can be combined into a single total order for all objects. In general this will be impossible since different threads may observe modifications to different objects in inconsistent orders. —end note]*"
    - http://eel.is/c++draft/intro.races#4
  - When the code is sequentially consistent, after a program execution, all threads will agree on the order in which writes to shared states have occurred

C++'s abstract machine?

- Memory ordering (cont'd)
  - There are a number of synchronization relationships described by the memory model of C++'s abstract machine
    - "*[…] an atomic store-release synchronizes with a load-acquire that takes its value from the store*"
    - "*[Note: The specifications of the synchronization operations define when one reads the value written by another. For atomic objects, the definition is clear. All operations on a given mutex occur in a single total order. Each mutex acquisition "reads the value written" by the last mutex release. — end note]*"
      - http://eel.is/c++draft/intro.races#6

C++'s abstract machine?

- Memory ordering (cont'd)
  - "*Sequenced before is an asymmetric, transitive, pair-wise relation between evaluations executed by a single thread, which induces a partial order among those evaluations*"
    - http://eel.is/c++draft/intro.execution#15
  - "*An evaluation A carries a dependency to an evaluation B if the value of A is used as an operand of B, unless [... some exceptions listed...]*"
    - http://eel.is/c++draft/intro.races#7

C++'s abstract machine?

- Memory ordering (cont'd)
  - "*An evaluation A inter-thread happens before an evaluation B if*
    - *A synchronizes with B, or*
    - *A is dependency-ordered before B, or*
    - *for some evaluation X*
      - *A synchronizes with X and X is sequenced before B, or*
      - *A is sequenced before X and X inter-thread happens before B, or*
      - *A inter-thread happens before X and X inter-thread happens before B*."
    - http://eel.is/c++draft/intro.races#9
    - Operations on distinct threads that are not inter-thread happens before are not ordered a priori
      - They can even occur at the same time

C++'s abstract machine?

- How about the execution stack, where arguments are put when a function is called?

C++'s abstract machine?

- How about the execution stack, where arguments are put when a function is called?
  - The C++ abstract machine has no such concept
  - Some platforms have more than one stack per thread
  - Some mechanisms (e.g. coroutines) can be stackless
- The word "stack" occurs a few times in the standard
  - Stack unwinding, when an exception is thrown
    - For an example, see http://eel.is/c++draft/except.ctor
  - The `<stack>` header and `std::stack`
    - See http://eel.is/c++draft/headers among other places

C++'s abstract machine?

- How about registers, where the CPU does its actual work?

C++'s abstract machine?

- How about registers, where the CPU does its actual work?
  - The C++ abstract machine has no such concept
  - There are a few occurrences of the word "register" in the standard
    - The `register` keyword, which is now unused (since C++17) but remains reserved
      - http://eel.is/c++draft/lex.key
    - "*When an object of class type X is passed to or returned from a function, if each copy constructor, move constructor, and destructor of X is either trivial or deleted, and X has at least one non-deleted copy or move constructor, implementations are permitted to create a temporary object to hold the function parameter or result object. [...]. [Note: This latitude is granted to allow objects of class type to be passed to or returned from functions in registers. — end note]*"
      - http://eel.is/c++draft/class.temporary#3
      - It's a non-normative note, not part of the abstract machine

Impact on programming

- Correct C++ programs conform to the rules of the abstract machine
  - Programs that rely on implementation-specific or unspecified behavior are at best non-portable
    - Even between compiler versions of a given cvendor
    - Your vendor will probably be careful, but you are beyond what the standard can guarantee
  - Programs that contain undefined behavior are essentially broken
    - They are time bombs

Impact on programming

- For example, take the following program
  - https://wandbox.org/permlink/dOtIzm4W8oBdun5Q

Impact on programming

- For example, take the following program
  - https://wandbox.org/permlink/dOtIzm4W8oBdun5Q
  - Why do we get the differences in results?
  - Why do we get the differences in timing?

Impact on programming

- For example, take the following program
  - https://wandbox.org/permlink/dOtIzm4W8oBdun5Q
  - Why do we get the differences in results?
  - Why do we get the differences in timing?
  - What happens if we use a higher optimization level? Why?

Impact on programming

- For example, take the following program
  - https://wandbox.org/permlink/dOtIzm4W8oBdun5Q
  - Why do we get the differences in results?
  - Why do we get the differences in timing?
  - What happens if we use a higher optimization level? Why?
  - This code is really bad… How could we make it better?

Impact on language design

- Should you plan to submit a proposal to the standard committee…
  - Restrict yourself to features that are part of the language
    - How to optimize register allocation makes no sense given C++'s abstract machine
    - Such proposals are better suited for your favorite vendors
- Should you want to add a feature that does not fit with C++'s abstract machine…
  - Think of ways not to pollute the standard
    - If you add registers to the machine, for example, is this a plus? Are there actual platforms that would be better represented if the standard did not mention registers altogether?

Tools you should know and use

- Matt Godbolt's Compiler Explorer
  - http://godbolt.org
- Fred Tingaud's Quick C++ Benchmark
  - http://quick-bench.com/
- Tim Song's CppWp, an online reference to C++'s Draft International Standard
  - http://eel.is/c++draft/

References

- Working Draft, Standard for Programming Language C++
  - http://wg21.link/n4640 (Feb. 2017)
- http://en.cppreference.com/w/cpp/language/memory_model
- Bjarne Stroutrup (2005), Abstraction and the C++ machine model
  - http://www.stroustrup.com/abstraction-and-machine.pdf

References

- David Chisnall (2011), High Performance Computing in C/C++ (*sic.*), The C Abstract Machine Model
  - http://cs.swan.ac.uk/~csdavec/HPC/1IntroAndPointers.pdf
- Edaqua Mortoray (2012)
  - https://mortoray.com/2012/06/18/abstract-machines-interpreters-and-compilers/
- Samvit Kaul (2014)
  - https://www.linkedin.com/pulse/20140703184017-137482-programming-the-c-c-abstract-machine-with-a-nop

References

- My own site (sorry for the color scheme)
  - http://h-deb.clg.qc.ca/
  - It's in French
- If you are interested in virtual machines (not technically the same as abstract machines), consider
  - http://h-deb.clg.qc.ca/Liens/Suggestions-lecture.html#virtual_machines

Questions? Comments? Threats?